UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

RICARDO KERSCHBAUMER

PROPOSIÇÃO DO PARADIGMA ORIENTADO A NOTIFICAÇÕES NO DESENVOLVIMENTO DE CIRCUITOS LÓGICO-DIGITAIS RECONFIGURÁVEIS

TESE DE DOUTORADO

CURITIBA
2018

RICARDO KERSCHBAUMER

PROPOSIÇÃO DO PARADIGMA ORIENTADO A NOTIFICAÇÕES NO DESENVOLVIMENTO DE CIRCUITOS LÓGICO-DIGITAIS RECONFIGURÁVEIS

Tese de doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de “Doutor em Ciências” – Área de Concentração: Engenharia de Computação.

Orientador: Prof. Dr. Carlos Raimundo Erig Lima
Coorientador: Prof. Dr. Jean Marcelo Simão

CURITIBA

2018

Dados Internacionais de Catalogação na Publicação

K41p
2018

Kerschbaumer, Ricardo
Proposição do paradigma orientado a notificações no desenvolvimento de circuitos lógico-digitais reconfiguráveis / Ricardo Kerschbaumer.-- 2018.
378 f. : il. ; 30 cm

Disponível também via World Wide Web
Texto em português com resumo em inglês
Tese (Doutorado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Industrial Elétrica, Curitiba, 2018
Bibliografia: f. 166-175

1. Arquitetura de computador. 2. Arquitetura de redes de computadores. 3. Paradigma orientado a notificações. 4. VHDL (Linguagem descritiva de hardware). 5. Arranjos de lógica programável em campo. 6. Computadores - Equipamento de entrada e saída. 7. Engenharia elétrica - Teses. I. Lima, Carlos Raimundo Erig. II. Simão, Jean Marcelo. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial. IV. Título.

CDD: Ed. 23 – 621.3

Biblioteca Central da UTFPR, Câmpus Curitiba
Bibliotecário: Adriano Lopes CRB-9/1429

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CURITIBA

TERMO DE APROVAÇÃO DE TESE Nº 176

A Tese de Doutorado intitulada **"Proposição do Paradigma Orientado a Notificações no Desenvolvimento de Circuitos Lógico Digitais Reconfiguráveis",** defendida em sessão pública pelo(a) candidato(a) **Ricardo Kerschbaumer**, no dia 30 de agosto de 2018, foi julgada para a obtenção do título de Doutor em Ciências, área de concentração Engenharia de Computação, e aprovada em sua forma final, pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial.

BANCA EXAMINADORA:

Prof(a). Dr(a). Paulo Cézar Stadzisz - Presidente – (UTFPR)
Prof(a). Dr(a). Volnei Antônio Pedroni – (UTFPR)
Prof(a). Dr(a). Antônio Augusto Medeiros Fröhlich – (UFSC)
Prof(a). Dr(a). Marco Aurélio Wehrmeister – (UTFPR)
Prof(a). Dr(a). Fabiano Silva - (UFPR)

A via original deste documento encontra-se arquivada na Secretaria do Programa, contendo a assinatura da Coordenação após a entrega da versão corrigida do trabalho.

Curitiba, 30 de agosto de 2018.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus por me propiciar a sabedoria, a tranquilidade e a força necessários para seguir em frente e buscar meus objetivos, sem desanimar com as adversidades do caminho. Agradeço ainda a Deus pela minha saúde e pela saúde de meus entes queridos, mantendo-os ao meu lado durante esta jornada.

Agradeço a minha esposa Ivane e ao meu filho Caetano que suportaram a minha ausência durante estes anos de idas e vindas. Sem seu apoio, incentivo e compreensão esta jornada não seria possível.

Agradeço também aos meus pais que nunca mediram esforços para me propiciar um futuro melhor, principalmente através do estudo. Assim possibilitam que eu chegasse onde estou hoje.

Quero agradecer também aos meus orientadores, Carlos Raimundo Erig Lima e Jean Marcelo Simão que conduziram e orientaram meus trabalhos com excelência e presteza. Seus ensinamentos extrapolaram os limites do mundo acadêmico, trazendo para mim valiosos conhecimentos para toda a vida.

Agradeço também ao Instituto Federal Catarinense que me permitiu o afastamento de minhas atividades como professor para me dedicar integralmente as atividades do doutorado.

A Universidade Tecnológica Federal do Paraná em especial o Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial também têm meus agradecimentos por me fornecerem a oportunidade de evoluir em meus estudos.

E por fim, não posso deixar de agradecer também aos meus colegas de trabalho e de estudo cujas contribuições foram muito importantes no desenvolvimento deste trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

RESUMO

KERSCHBAUMER, Ricardo. **Proposição do Paradigma Orientado a Notificações no Desenvolvimento de Circuitos Lógico-Digitais Reconfiguráveis.** 2018. 378 f. Tese de Doutorado - Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI). Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2018.

As arquiteturas usuais de computação estão tendo dificuldades em acompanhar a crescente demanda por capacidade de processamento. As FPGAs vêm se mostrando uma alternativa interessante a estas arquiteturas, principalmente para aplicações que demandam considerável poder de processamento e paralelismo de execução. Mesmo com características promissoras, a utilização das FPGAs é dificultada por seu modelo de programação e pelas linguagens tradicionais de síntese de *hardware*, o que demanda acentuado conhecimento técnico. Alternativamente, uma forma mais fácil de aproveitar o potencial das FPGAs é através da utilização de ferramentas de síntese em alto nível. Estas ferramentas tornam mais fácil a programação das FPGAs, porém muitas vezes os circuitos gerados utilizam mais recursos, são mais lentos e exploram menos paralelismo do que circuitos descritos através de linguagens tradicionais de síntese de *hardware*. Uma solução para alguns destes problemas é apresentada no Paradigma Orientado a Notificações (PON). O PON apresenta características de evitar redundâncias e tender a desacoplamento fino de partes do código, o que viabiliza paralelização e mesmo distribuição, algo particularmente interessante para a síntese de *hardware* digital. Isto se dá por meio de entidades lógico-causais e facto-execucionais que colaboram por meio de notificações pontuais. Ademais, elas emergem de programação em alto nível orientada a regras, sendo que existem arquétipos e linguagem para tal no tocante a *software*. Isto posto, este trabalho apresenta uma implementação do PON onde todos os elementos deste paradigma são modelados em blocos de lógica reconfigurável, utilizando linguagem VHDL. Essa nova implementação do PON para *hardware* digital, chamada de PON-HD 1.0, foi desenvolvida para facilitar a síntese em FPGA. Com o PON-HD 1.0 é possível gerar código VHDL para FPGA diretamente de um programa PON escrito em linguagem de alto nível. Esta linguagem e respectivo compilador se chama LingPON-HD 1.0, também proposta no âmbito deste trabalho e inspirada na linguagem precedente do PON para *software*. Para avaliar o desempenho e a estabilidade dos circuitos gerados com esta tecnologia do PON-HD 1.0, foram realizados alguns experimentos comparativos com linguagens tradicionais de síntese de *hardware*. Esses experimentos demonstraram que esta tecnologia do PON-HD 1.0 permite criar, com considerável rapidez e facilidade, circuitos digitais confiáveis com desempenho e paralelismo adequados, tudo a luz dos comparativos realizados. Como conclusão, os resultados demonstram a viabilidade do PON como paradigma e ferramental para o desenvolvimento adequado para o âmbito de FPGAs.

Palavras-chave: Síntese de lógica reconfigurável. FPGA. Arquiteturas paralelas. Paradigma orientado a notificações - PON.

ABSTRACT

KERSCHBAUMER, Ricardo. **Proposition of the Notification Oriented Paradigm in the Development of Reconfigurable Digital Logic Circuits.** 2018. 378 f. Tese de Doutorado - Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI). Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2018.

Usual computer architectures have shown issues in following the growing demand for processing. The FPGAs are an interesting alternative to these architectures, especially for applications that require considerable processing power and execution parallelism. Even with promising features, the use of FPGAs is complicated by its programming model and by the traditional hardware synthesis languages, which demands great technical knowledge. Alternatively, an easier way to enjoy the potential of FPGAs is using high-level synthesis tools. These tools make easier the FPGAs programming, however usually the generated circuits demand more resources, are slower, and exploit less parallelism than circuits described using traditional hardware synthesis languages. A solution to some of this problems is shown in the Notification Oriented Paradigm (NOP). The NOP presents features of avoiding redundancies and provide fine decoupling of parts of the code, which enables parallelization and even distribution, something particularly interesting for digital hardware synthesis. This is done through logical-causal and factual-executional entities that collaborate by means of punctual notifications. In addition, they emerge from high-level rule-oriented programming. Moreover, there are a framework and language for NOP software. That said, this work presents an implementation of NOP where all elements of this paradigm are modeled in reconfigurable logic blocks, using VHDL language. This new solution of NOP for digital hardware, called the PON-HD 1.0, was developed to facilitate the synthesis for FPGA. With the PON-HD 1.0 you can generate VHDL code for FPGA directly from a NOP program written in high-level language. This language and its compiler are called LingPON-HD 1.0, also proposed as part of this work and inspired by the previous language of NOP for software. In order to evaluate the performance and stability of circuits generated with the technology of PON-HD 1.0, some comparative experiments were carried out with traditional hardware synthesis languages. These experiments have shown that the technology of PON-HD 1.0 allows to create, with considerable speed and ease, trusted digital circuits with appropriate performance and parallelism, based on the performed comparisons. In conclusion, the results demonstrate the feasibility of the NOP as a paradigm and toolchain for suitable development in FPGAs scope.

Key words: Reconfigurable logic synthesis, FPGA, Parallel architectures, Notification Oriented Paradigm - NOP.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

Abreviaturas	Latim	Significado
cf.	*confer*	conforme
i.e.	*id est*	isto é
e.g.	*exempli gratia*	por exemplo
et al.	*et alii ou et aliae*	e outros ou e outras

LISTA DE SIGLAS, ACRÔNIMOS

Sigla	Original	Tradução
API	*Application Programming Interface*	Interface de Programação de Aplicativos
ARQPON	Arquitetura de computação PON	*NOP Computer Achitecture*
ASIC	*Application Specific Integrated Circuits*	Circuito Integrado de Aplicação Específica
CON	Controle Orientado a Notificações	*Notification Oriented Control*
CPLD	*Complex Programmable Logic Device*	Dispositivo Lógico Programável Complexo
CPU	*Central Processing Unit*	Unidade Central de Processamento
DON	Desenvolvimento Orientado a Notificações	*Notification Oriented Development*
DOR	Desenvolvimento Orientado a Regras	*Rules Oriented Development*
DSP	*Digital Signal Processing*	Processador Digital de Sinais
FBE	*Fact Base Element*	Elemento da Base de Fatos
FIFO	*First-In First-Out*	Primeiro a Entrar, Primeiro a Sair
FILO	*First-In Last-Out*	Primeiro a Entrar, Último a Sair
FPGA	*Field Programmable Gate Arrays*	Matrizes de Portas Programáveis em Campo
FSM	*Finite State Machines*	Máquinas de Estado Finito
GPU	*Graphics Processing Unit*	Unidade de Processamento Gráfico
HD	*Hardware* Digital	*Digital Hardware*
HDL	*Hardware Description Language*	Linguagem de descrição de *hardware*
IEEE	*Institute of Electrical and Electronics Engineers*	Instituto de Engenheiros Eletricistas e Eletrônicos
MON	Modelagem Orientada a Notificações	*Notification Oriented Modeling*
NOCA	*Notification Oriented Computer Architecture*	Arquitetura de Computador Orientada a Notificações
NOM	*Notification Oriented Metodology*	Metodologia Orientada a Notificações
NOP	*Notification Oriented Paradigm*	Paradigma Orientado a Notificações
PD	Paradigma Declarativo	*Declarative Paradigm*

PF	Paradigma Funcional	*Functional Paradigm*
PI	Paradigma Imperativo	*Imperative Paradigm*
PL	Paradigma Lógico	*Logical Paradigm*
PLL	*Phase-Locked Loop*	Loop Fechado de Fase
PON	Paradigma Orientado a Notificações	*Notification Oriented Paradigm*
POO	Programação Orientada a Objetos	*Object Oriented Programming*
RON	Requisitos Orientados a Notificações	*Notification Oriented Requirements*
RTL	*Register Transfer Level*	Nível de transferência de Registradores
RUP	*Rational Unified Process*	Processo Unificado Rational
SBR	Sistemas Baseados em Regras	*Rule Based Systems*
SPLD	*Simple Programmable Logic Device*	Dispositivo Lógico Programável Simples
SRAM	*Static Random Access Memory*	Memória de Acesso Aleatório Estática
V-REP	*Virtual Robot Experiment Platform*	Plataforma de Experiência de Robô Virtual
VHDL	*VHSIC Hardware Description Language*	Linguagem de descrição de *hardware* VHSIC
VHSIC	*Very High Speed Integrated Circuits*	Circuito Integrado de Velocidade Muito Alta
YAML	*Yet Another Markup Language*	Ainda Outra Linguagem de Marcação

SUMÁRIO

1 Introdução

1.1 Contextualização

A contextualização deste trabalho de pesquisa se divide em quatro momentos relativos à lógica reconfigurável em *hardware*, paradigmas de programação, paradigma orientado a notificações (PON) e PON em *hardware* digital. As subseções seguintes tratam destes temas no tocante a contextualização.

1.1.1 Contextualização sobre lógica reconfigurável em *hardware*

As arquiteturas atuais de computação, à luz do modelo de von Neumann e afins, estão tendo dificuldades em acompanhar a crescente demanda por poder de processamento, principalmente com a crescente importância da computação nas mais variadas áreas (CROSBIE, 2010; QIAN *et al*., 2005).

Os avanços tecnológicos que vem resultando no crescimento do poder de processamento dependem, historicamente, principalmente do aumento da frequência de operação (*clock*) dos processadores usuais e do aumento da densidade de componentes integrados nos semicondutores. Esta forma de ampliação da capacidade de processamento não vem se sustentando, principalmente, por problemas com relação à dissipação do calor gerado no interior dos processadores (FAN *et al*., 2018).

Entretanto, o aumento da frequência de *clock* não é o único fator que contribui para o aumento da capacidade de processamento dos processadores. Evoluções na microarquitetura e mecanismos de aceleração de acesso à memória também têm um papel importante no aumento da capacidade de processamento, porém não são suficientes para atender esta crescente demanda por poder de processamento (BORKAR; CHIEN, 2011).

Outra alternativa amplamente utilizada nesta computação tradicional, baseada em von Neumann, é a adoção de múltiplos núcleos (processadores) em uma mesma pastilha, os assim chamados processadores *multi-core*. Esta abordagem deu origem às contemporâneas arquiteturas de computação paralela, as quais, com o aumento do número de unidades de processamento, permitiriam o aumento do desempenho dos computadores. Ainda nesta linha

de arquitetura paralelas contemporâneas, surgiram também as *Graphics Processing Units* (GPUs). As GPUs são processadores especialmente desenvolvidos para processamento gráfico, porém, em função de sua grande capacidade de processamento paralelo, vêm sendo utilizadas para executar uma infinidade de algoritmos que necessitam grande poder de processamento.

Entretanto, é importante, e mesmo fundamental, salientar que o paralelismo dessas arquiteturas depende das características do *software* e de sua capacidade em aproveitar o paralelismo propiciado pelo *hardware*, o que não é nada evidente (BORKAR; CHIEN, 2011). Mesmo com todas estas tecnologias com *multi* ou *many* cores, é notório que está ficando cada vez mais difícil atender à crescente demanda por poder de processamento justamente porque as técnicas de software e seus algoritmos têm dificuldades em explorar o paralelismo de maneira apropriada (CROSBIE, 2010; QIAN *et al*., 2005). Ademais, há também problemas como o chamado "*memory wall*" que em suma é uma crescente diferença de desempenho entre memórias e processadores (LINHARES, 2015).

Neste contexto, há a necessidade de novas arquiteturas de computação alternativas ao modelo von Neumann e afins, incluindo suas evoluções com vários núcleos ou *cores*. Justamente, uma alternativa computacional são as chamadas Matrizes de Portas Programáveis em Campo ou, em idioma inglês, *Field Programmable Gate Arrays* (FPGAs), as quais vem se mostrando uma opção interessante (CROSBIE, 2010; QIAN *et al*., 2005; BORKAR; CHIEN, 2011). Diferente dos processadores usuais oriundos da visão arquitetural von Neumann, não raramente chamados genericamente de *Central Processing Unit* (*CPU*), as FPGAs não executam um conjunto sequencial de instruções, permitindo o processamento de forma paralela, através de circuitos digitais diretamente sintetizados no *hardware*. Na Figura 1, neste contexto, é apresentada uma ilustração destes dois diferentes modelos de processamento.

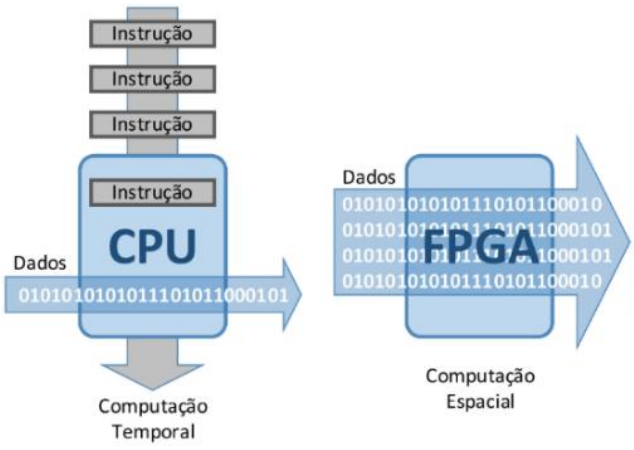

Figura 1 - Modelo de processamento em uma CPU e em uma FPGA.

Fonte: Autoria própria.

Observa-se um crescente interesse por parte dos fabricantes de microprocessadores nas FPGAs, pois sua natureza reconfigurável, seu consumo de energia menor do que das CPUs e sua grande capacidade de processamento propiciada pelo paralelismo de execução dos algoritmos permitem sua utilização em vários sistemas computacionais, com ganho de performance e redução do consumo de energia (OLDS, 2016). Este ganho de performance é em comparação com as CPUs tradicionais. Na verdade, a flexibilidade das FPGAs permite desde sistemas de todo paralelo até sistemas sequenciais, inclusive pela emulação de CPUs (LEONG, 2008).

De fato, as FPGAs são uma alternativa interessante pois apresentam características de performance e eficiência superior aos processadores tradicionais e mesmo as GPUs com software apropriado, porém mantendo flexibilidade superior aos ASICs (AWAD, 2009). A Figura 2 apresenta estas características nos principais tipos de sistemas computacionais.

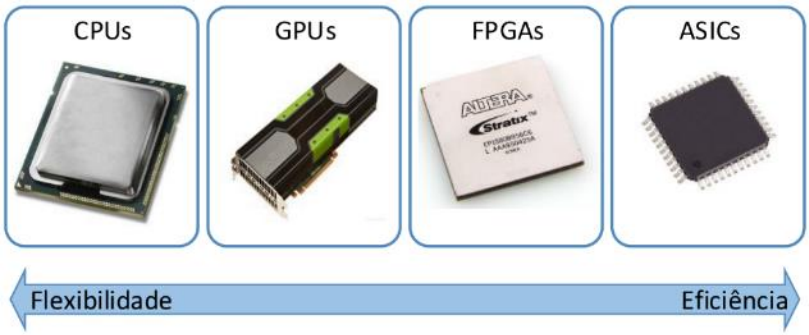

Figura 2 - Características de eficiência e flexibilidade das FPGAs.

Fonte: Autoria própria.

Contudo, as FPGAs não apresentam apenas vantagens uma vez que, por exemplo, sua utilização é dificultada por seu modelo de programação.

O modelo de programação das FPGAs é complexo, exigindo do desenvolvedor conhecimentos avançados de *hardware* digital e de linguagens específicas de descrição de *hardware,* (HDL - *Hardware Description Languages*) como VHDL - (VHSIC *Hardware Description Language* ou Linguagem de descrição de *hardware* VHSIC (*Very High Speed Integrated Circuits*)) ou Verilog (AWAD, 2009). Como alternativa à utilização de linguagens de descrição de *hardware*, o potencial das FPGAs pode ainda ser aproveitado através da utilização de ferramentas de síntese em alto nível (MEEUS *et al*., 2012). Estas ferramentas não exigem do desenvolvedor conhecimentos tão profundos de *hardware*.

A principal vantagem destas ferramentas é a utilização de linguagens em nível de abstração um tanto mais elevado que as linguagens de programação específicas para *hardware*,

reaproveitando princípios que antes se aplicavam essencialmente à engenharia de *software*. Mais especificamente, reaproveita-se, atualmente, principalmente a fase de programação ou implementação, o que inclui certo reaproveitamento de linguagens de programação clássicas como C e C++ (BUNKER; GOPALAKRISHNAN; MCKEE, 2004). Isto dito, neste âmbito são utilizadas linguagens derivadas do C e do C++ ou mesmo linguagens proprietárias que, ainda assim, exigem considerável conhecimento técnico. Neste caso, linguagens derivadas do C e do C++ facilitam certo reaproveitamento dos códigos já existentes feitos outrora para *software* (BUNKER; GOPALAKRISHNAN; MCKEE, 2004).

Apesar das vantagens citadas, o uso das chamadas "ferramentas de síntese em alto nível" apresentam também desvantagens importantes. Neste sentido, muitas vezes os circuitos gerados por estas ferramentas são mais lentos e ocupam mais recursos na FPGA do que os circuitos criados diretamente em linguagens tradicionais de síntese de *hardware* como VHDL e Verilog (MARCONDES; FRÖHLICH, 2008; MEEUS *et al*., 2012; NANE *et al*., 2016). Ademais, essas linguagens dificultam a paralelização dos algoritmos, salientando as linguagens derivadas do C e do C++, mas não apenas elas (MEEUS *et al*., 2012). Isto acontece devido à falta de informação temporal no código e a falta de um modelo de concorrência ou paralelismo efetivamente apropriado (WINDH *et al*., 2015).

O que se observa é que não apenas as linguagens de programação usais, como C, C++ e suas derivadas, mas também os principais paradigmas de programação que as regem possuem deficiências que dificultam a distribuição e a paralelização de execução das aplicações. Em sua maioria, paradigmas esses idealizados outrora para o âmbito de *software* monoprocessado para arquitetura von Neumann e afins (SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009; SIMÃO *et al*., 2012a; LINHARES, 2015). Em tempo, entende-se por paradigma de programação, mais do que um conjunto de técnicas e conceitos, mas uma forma própria de estruturar o pensamento na construção dos programas (VAN ROY; HARIDI, 2004; VAN ROY, 2009; BANASZEWSKI, 2009).

1.1.2 Contextualização sobre Paradigmas de Programação

Segundo Gabbrielli e Martini (2010), os principais paradigmas de programação atuais podem ser classificados como Paradigma Imperativo (PI) e Paradigma Declarativo (PD). O PI pode ser entendido como compreendendo as abordagens chamadas procedimental e a abordagem orientada a objetos, sendo esta mais rica em poder de abstração que aquela. O PD,

mais especificamente, compreenderia o Paradigma Funcional (PF) e o Paradigma Lógico (PL) (SCOTT, 2000). Entretanto, em compreensões menos hierarquizadas, como a de Van Roy (2009), pode-se considerar que há intersecções entre o PI e o PD, estando parte do PF em uma dada intersecção, a título de exemplo.

O PI é o paradigma mais antigo surgido a luz do pensamento sequencial, sendo apropriado para a arquitetura computacional de von Neumann (SCOTT, 2000). Isto dito, em suma, os programas desenvolvidos segundo o PI dependem de linguagens de programação repletas de detalhes técnicos e com lógicas complexas de construção orientadas a laços de repetição e estruturas de decisão, não raro geradores de redundâncias de processamento e acoplamentos de partes de código. De fato, esta abordagem tende a produzir acoplamento entre a lógica do programa e os estados por ele processados, o que dificulta o desacoplamento fino de código em módulos. Tal acoplamento ou falta de modularidade fina dificulta a tarefa de paralelização e distribuição de execução do código (HUGHES; HUGHES, 2003).

O PD, ao seu turno, surge subsequentemente o PI sendo uma camada de abstração e mesmo organização sobre ele. No PD existe uma distinção clara entre os dados que formam a Base de Fatos e as decisões lógico-causais (não raro chamadas de regras) que formam a Base de Regras, sendo que o ‘laço de repetição’ que compara essas bases é automatizado em uma máquina de inferência. Assim, no PD o programador ou desenvolvedor se preocupa essencialmente em definir o que deve ser feito, definindo os estados (fatos) dos elementos pertinentes a aplicação e as relações lógico-causais (regras) que determinam a atuação sobre estes estados (LINHARES, 2015; SCOTT, 2000).

Em tempo, este paradigma também apresenta problemas de acoplamento e distribuição, assim como o PI inclusive por ser uma abstração sobre este. Neste sentido, no PD isso acontece devido aos mecanismos de inferência, que são normalmente monolíticos e centralizados. Apesar de haver várias iniciativas de paralelização e distribuição no PD, os resultados são limitados pelas características intrínsecas da arquitetura e do processo de inferência monolítico-acoplador (BANASZEWSKI, 2009).

Maiores detalhes sobre paradigmas de programação e suas limitações podem ser encontrados em outros trabalhos do grupo de pesquisa relacionado ao PON (BANASZEWSKI, 2009; RONZCKA, 2012; VALENÇA, 2012; XAVIER et al., 2014; LINHARES, 2015), bem como em literaturas outras (GABBRIELLI; MARTINI, 2010; SCOTT, 2000; VAN ROY, 2009).

Uma solução para alguns dos problemas destes paradigmas é apresentada com o Paradigma Orientado a Notificações (PON), considerado na próxima subseção.

1.1.3 Contextualização sobre Paradigma Orientado a Notificações (PON)

O Paradigma Orientado a Notificações (PON) foi proposto inicialmente e embrionariamente por Simão (2001, 2005) como uma solução de controle discreto para sistemas de manufatura, subsequentemente sendo generalizado como solução de inferência e paradigma de programação por Simão e Stadzisz (2002, 2008, 2009). Em suma, o PON é baseado em processo de inferência constituído de sub-entidades pequenas, ditas inteligentes e desacopladas, que colaboram entre si através de notificações pontuais a fim realizar o cálculo lógico-causal de forma desacoplada, evitando redundâncias de processamento (SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009).

Ademais, o PON permite conceber sistemas em alto nível, similarmente ao PD, mas com outra lógica de composição e inferência nas aplicações decorrentes. Este novo conceito para conceber, compor e executar aplicações apresentado pelo PON é baseado em entidades-regras, compostas de sub-entidades colaborativas, as quais são notificáveis a partir de entidades factuais, o que é explicado a seguir por meio de exemplo (SIMÃO; STADZISZ, 2009).

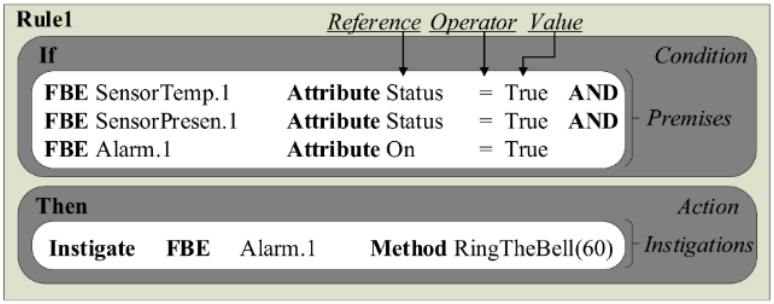

Figura 3 - Exemplo de regra lógico-causal.
Fonte: Adaptado de Simão e Stadzisz (2009).

Apenas a título de elucidação, a Figura 3 apresenta um exemplo de regra lógico-causal de um sistema de correlação de sensores. É a partir destas organizações em regras e entidades factuais que se organiza de forma emergente os sistemas de inferência colaborativa por notificações do PON, o qual será detalhado na sequência deste documento, mais especificamente na seção 2.5. A título de introdução, a Figura 4 apresenta um esboço da inferência colaborativa apresentada pelo PON.

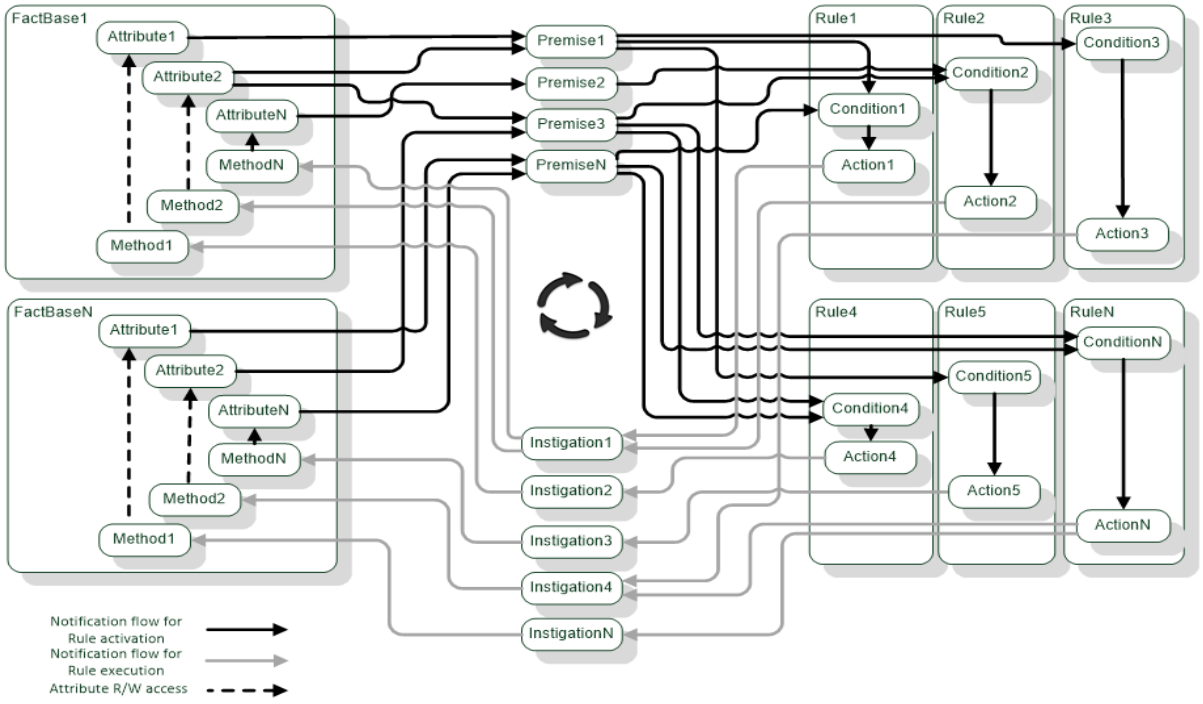

Figura 4 - Representação do Ciclo de Inferência por Notificações do PON.
Fonte: Adaptado de Linhares (2015).

Assim, o PON como paradigma de desenvolvimento tem o objetivo de tornar mais fácil a tarefa de criação de aplicativos (por permitir concepção em alto nível), tornando o código mais otimizado (por evitar redundâncias) e paralelizável/distribuível (por garantir desacoplamento) (SIMÃO; STADZISZ, 2009). Tais características tornam o PON interessante também para o desenvolvimento de aplicações em *hardware* além das aplicações em *software* (LINHARES, 2015).

O PON em si não está vinculado a nenhuma plataforma específica. Assim sendo, várias implementações vêm sendo realizadas nas mais variadas plataformas. Dentre estas implementações pode-se destacar um *framework* PON C++, proposto a partir de Simão (2005) {cf. Simão e Stadzisz (2008)} e subsequentemente refeito por Banaszewski (2009) {cf. Simão *et al*. (2017)} e depois ainda por Valença (2012) e mesmo por Ronszcka (2012) {cf. Simão *et al*. (2012c), Ronszcka *et al*. (2017) e Simão *et al*. (2017a)}. Na sequência várias versões deste *framework* foram desenvolvidas, algumas para linguagens outras como C# e Java (HENZEN, 2015) inclusive com versões prototipais para sistemas distribuídos (OLIVEIRA *et al.*, 2018; BARRETO *et al.*, 2018), enquanto outras ainda em C++ mas para *multi-threads/multi-core* (BELMONTE *et al.* 2016), sistemas distribuídos (TALAU, 2016), lógica nebulosa/fuzzy

(MELO; SIMÃO; FABRO, 2014; MELO, 2016;) e mesmo redes neurais (SCHÜTZ, 2015; SCHÜTZ, 2018).

Subsequentemente, inclusive para aumentar a facilidade de utilização do PON no desenvolvimento de programas, foi desenvolvido o protótipo de uma linguagem de programação específica para o PON e respectivo compilador, capaz de traduzir tal linguagem em um código-alvo, conforme detalhado em Ferreira (2015). Esta linguagem e mesmo tecnologia é chamada LingPON 1.0 e permite escrever aplicações para *software* monoprocessado em conformidade com este paradigma, de forma fácil e rápida (FERREIRA, 2015) (Ronszcka, 2018).

O PON também tem representação em ambientes paralelos e distribuídos. Isso é natural levando em conta seu modelo no que diz respeito ao desacoplamento implícito das entidades e a concorrência inerente das notificações. Tais características favorecem a distribuição e paralelização das aplicações (WEBER *et al*. 2010; BELMONTE 2012). Estas características vêm sendo exploradas por pesquisadores e afins no âmbito do PON em *software*, por hora no tocante ao *Framework* PON (BARRETO *et al.,* 2018; BELMONTE; SIMÃO; STADZISZ, 2012; BELMONTE, 2012; BELMONTE *et al.*, 2016; TALAU, 2016; OLIVEIRA *et al*., 2018), mas também sendo vislumbrando no âmbito da LingPON (Ronszcka, 2018).

Em tempo, as características do PON não são interessantes apenas para o desenvolvimento de *software* em si, sendo que o próprio projeto do *software* pode ser feito segundo o PON. Seguindo este raciocínio Wiecheteck (2011) propôs um método para projeto de *software* PON, denominado Desenvolvimento Orientado a Notificações (DON) (WIECHETECK *et al.*, 2011). Ainda com base no DON surgiu o Desenvolvimento Orientado a Regras (DOR) que foi motivo de um pedido de patente (SIMÃO; STADZISZ; WIECHETECK, 2012). Aliado a isso existe uma nova abordagem de engenharia de *software* do PON chamada Modelagem Orientada a Notificações (MON) (MENDONÇA, 2015) e a nascente abordagem de Requisitos Orientados a Notificações (RON) (SIMÃO *et al.*, 2016; NOVAES *et al*., 2018). Ainda, houve também esforços no âmbito de engenharia de testes para com o PON (KOSSOSKI; SIMÃO; STADZISZ, 2014; KOSSOSKI, 2015).

Estas abordagens relativas à Engenharia de *Software* podem, ao que tudo indica, também ser úteis a Engenharia de *Hardware.* Em todo o caso, o PON em si tem sido vislumbrado no âmbito de *hardware* (LINHARES, 2015). Isto é descrito na próxima subseção.

1.1.4 Contextualização sobre PON em *Hardware*

A bem da verdade, as características de paralelização e distribuição do PON não são devidamente aproveitadas nas plataformas tradicionais de computação, dado que o PON prevê um nível de desacoplamento fino de suas entidades. Tal distribuição, mesmo empregando algum paralelismo como em *multi-core*, sempre incorre em sequencializações, como no caso do processamento em cada *core*. Assim, abordagens do PON em *Hardware* Digital (HD) constituíram-se em tema de pesquisa (KERSCHBAUMER *et al.*, 2015; KERSCHBAUMER *et al.*, 2018a; KERSCHBAUMER *et al.*, 2018b; PORDEUS *et al.*, 2016).

Dentre todas as implementações do PON em *hardware*, a que tem maior relação com este trabalho é a implementação do PON em *hardware* digital realizada por Witt *et al.* (2011). Nesta implementação, todos os elementos que compõem o PON são modelados em blocos de lógica reconfigurável (WITT *et al.*, 2011). Esta implementação se mostrou funcional, permitindo a execução de aplicações PON puramente em *hardware*, constituindo-se no *framework* PON-HD prototipal.

Apesar de funcional, este *framework* ou arquétipo não apresenta mecanismos de determinismo e de resolução de conflitos, tornando sua utilização um tanto complexa, exigindo do desenvolvedor a implementação destes mecanismos manualmente. (WITT *et al.*, 2011). Também não foram realizadas otimizações nos elementos de *hardware*, prejudicando a performance dos circuitos resultantes. Outro ponto negativo desta implementação é que o desenvolvedor deve conectar manualmente todos os blocos de *hardware* de forma a montar a cadeia de notificações e criar o comportamento desejado para a aplicação.

Para melhorar esta característica, Jasinski (2012) propôs uma solução para a geração de *hardware* em VHDL a partir de especificação em PON via linguagem YAML (*Yet Another Markup Language*). A linguagem YAML é uma linguagem de serialização de dados amigável para humanos e padronizada para todas as linguagens de programação (EVANS, 2017). Nesta solução, a aplicação PON é descrita em YAML (portanto em uma linguagem de alto nível), sendo posteriormente convertida para arquivos VHDL (JASINSKI, 2012).

Também com o objetivo de melhorar a execução de aplicações PON, com a ajuda de *hardware* reconfigurável, Peters *et al*. (2012) desenvolveu uma arquitetura na qual o PON é implementado parte em *software* e parte em *hardware*, como um co-processador. Este co-processador é responsável pelas avaliações lógico-causais. Experimentos demostraram um

aumento de performance em comparação a aplicações desenvolvidas puramente em *software com PON, na época o Framework C++ 1.0* (PETERS *et al*., 2012; PETERS, 2012).

Outro viés do uso do PON em *hardware* foi uma arquitetura de computação completa orientada a notificações chamada de ARQPON. Para executar de forma adequada as aplicações PON essa nova arquitetura de computação foi proposta por Linhares (2015) em sua tese de doutoramento. Esta arquitetura foi denominada ARQPON, originando um microprocessador PON completo implementado em uma FPGA (LINHARES 2015; LINHARES; SIMAO; STADZISZ, 2015). Mais recentemente, também houve o desenvolvimento de um simulador para o ARQPON, já que as FPGAs disponíveis ao grupo de pesquisa não possuíam recursos suficientes para testar escalabilidade e estabilidade, propriedades estas que o simulador permitiu demostrar (PORDEUS, 2017).

Isto posto, muito embora o COPON e o ARQPON ensejem novas e instigantes pesquisas, uma solução efetiva que explore o potencial do PON diretamente sintetizado em *hardware*, de forma purista sem elementos outros que os dele mesmo, ainda não havia sido apresentada, sendo um dos pontos motivadores do trabalho aqui apresentado. Assim, a efetiva utilização do PON em *Hardware* Digital, ou simplesmente PON-HD, a partir de linguagem de desenvolvimento de alto nível e tecnicamente viável, apresenta-se como objeto de pesquisa particularmente pertinente. Isto serviria, ademais e neste contexto, para verificar se o PON é adequado para a elaboração de circuitos em lógica reconfigurável.

Nos próximos capítulos todos os assuntos que foram introduzidos aqui serão discutidos de forma mais aprofundada. Na seção seguinte serão apresentadas as motivações deste trabalho e na sequência os objetivos do mesmo.

1.2 Motivações

Primeiramente, há de se considerar que as arquiteturas atuais e usuais de computação estão tendo dificuldades em acompanhar a crescente demanda por poder de processamento e as FPGAs vêm se mostrando uma alternativa para atender a esta demanda (CROSBIE, 2010; QIAN *et al*., 2005; BORKAR; CHIEN, 2011). Igualmente, entretanto, há de se considerar que a utilização das FPGAs é dificultada por seu modelo de programação (AWAD, 2009) e a utilização de ferramentas de síntese em alto nível, as quais deveriam permitir aproveitar mais facilmente o potencial das FPGAs, não o fazem em plenitude, principalmente em função das limitações dos paradigmas vigentes de desenvolvimento (MEEUS *et al*., 2012). Outro aspecto

importante a se considerar é que as FPGAs e as ferramentas de síntese em alto nível associadas têm, apesar das dificuldades, um importante papel no desenvolvimento da computação.

O PON, por outro lado, vem apresentado resultados positivos em várias plataformas. Seu processo de inferência baseado em entidades pequenas e desacopladas, que colaboram entre si através de notificações pontuais para conceber e executar aplicações, trazem várias vantagens na execução das aplicações (SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009).

Tendo em mente a importância das FPGAs e das ferramentas de síntese em alto nível e as deficiências apresentadas por estas ferramentas, a utilização do PON no desenvolvimento em alto nível de *hardware* digital apresenta-se como uma ideia muito promissora. Motivando, assim, este trabalho de doutorado.

1.3 Objetivos

Nesta seção são apresentados os objetivos gerais e específicos que nortearam o desenvolvimento deste trabalho de doutorado.

1.3.1 Objetivo geral

Desenvolvimento e análise do Paradigma Orientado a Notificação (PON) para a geração de VHDL, visando a subsequente síntese de *Hardware* Digital (HD) dito notificante por meio da proposição de ferramental PON-HD específico, bem como a verificação das vantagens e desvantagens das propriedades intrínsecas do PON na qualidade dos circuitos digitais obtidos, isto por meio de experimentos comparativos com outras abordagens de projeto de *Hardware* Digital em alto nível.

1.3.2 Objetivos específicos

1) Realizar revisão dos aspectos técnicos e teóricos a respeito dos dispositivos de lógica reconfigurável e suas técnicas de programação, das chamadas ferramentas de síntese em alto nível em geral, dos paradigmas de desenvolvimento e principalmente do Paradigma Orientado a Notificações a fim de identificar e caracterizar a lacuna existente referente à

síntese em alto nível e como o PON pode ser empregado para o desenvolvimento em alto nível de circuitos em lógica reconfigurável.

2) Desenvolver um conjunto de técnicas e ferramentas que permitam a utilização do PON no desenvolvimento em alto nível de circuitos em lógica reconfigurável, por meio de VHDL a luz das notificações que permita a partir deste gerar circuitos ditos notificantes. Este conjunto de ferramentas denominado PON-HD 1.0 será composto por um *framework* para VHDL chamado *Framework* PON-HD 1.0, por uma linguagem chamada LingPON-HD 1.0 e seu compilador e por um simulador para o PON-HD 1.0.

3) Tornar a implementação do PON em *hardware* digital tecnologicamente viável inclusive por circuitaria resultante com *clock* e utilização de recursos apropriados, bem como pela programação de mais alto nível, isto por meio da programação em regras, na linguagem de programação LingPON-HD 1.0 e seu compilador, permitindo que desenvolvedores sem experiência em síntese de *hardware* desenvolvam circuitos digitais em alto nível.

4) Realizar testes e experimentos para verificar se o PON é apropriado ao desenvolvimento de circuitos em *hardware* digital, por meio da geração de VHDL intermediário orientado a notificações.

5) Verificar se a performance do PON-HD 1.0 é apropriada quando comparada a outras abordagens de síntese de *hardware* em alto nível, especificamente o VHDL e o Vivado HLS (Vivado *high level synthesis* da Xilinx), em termos de velocidade de operação e utilização de recursos da FPGA.

6) Apresentar os resultados obtidos à comunidade acadêmica para avaliação e discussão dos mesmos.

Estes são os objetivos definidos para este trabalho de doutorado. O cumprimento de cada um destes objetivos, parcial ou totalmente, será assunto tratado nos capítulos deste trabalho, sendo feito o fechamento a respeito no capítulo de conclusão.

1.4 Contribuições e resultados

Este trabalho de doutorado apresenta um ferramental que permite a utilização do PON na síntese em alto nível de *hardware* digital através de um conjunto de ferramentas específico composto por um *Framework* PON-HD 1.0, uma linguagem PON-HD 1.0, um compilador PON-HD 1.0 e um simulador PON-HD 1.0. Este ferramental recebe o nome de Ferramental PON-HD 1.0 ou simplesmente PON-HD 1.0.

A Figura 5 apresenta um panorama das implementações atuais do PON visando posicionar este presente trabalho de pesquisa no panorama do PON. Neste sentido, os blocos destacados indicam as áreas onde este trabalho de doutorado está contribuindo através do ferramental PON-HD 1.0. As setas indicam a evolução do paradigma como um todo.

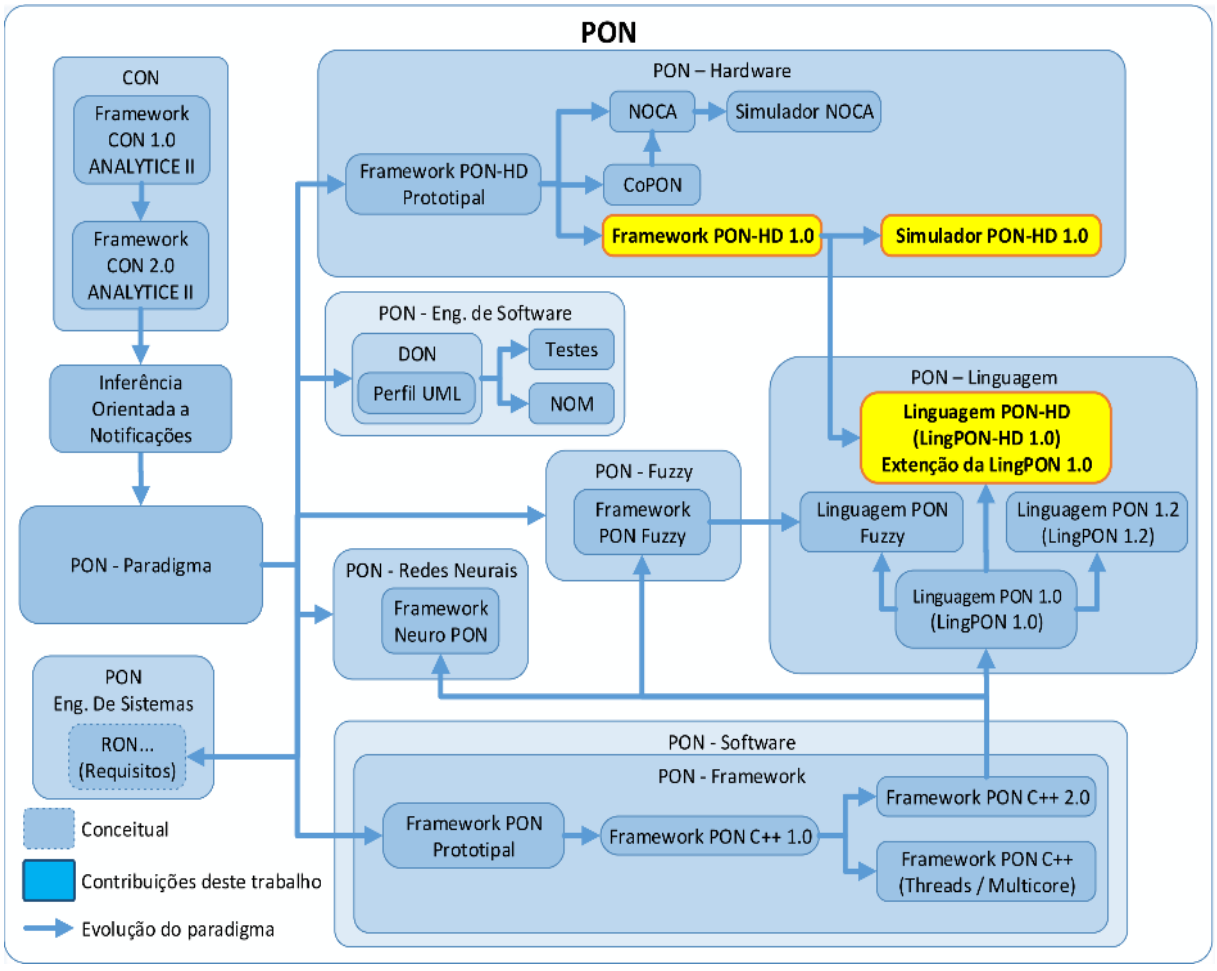

Figura 5 - Panorama atual do PON.

Fonte: Autoria própria.

Neste contexto, a principal contribuição do PON-HD 1.0 é verificar se o Paradigma orientado a notificações é apropriado para o desenvolvimento de circuitos digitais em lógica reconfigurável, apresentando-se como uma alternativa vantajosa para o desenvolvimento de soluções baseadas em FPGAs.

Além disso, como ferramental de alto nível de abstração, o PON-HD 1.0 permite que desenvolvedores com poucos conhecimentos sobre *hardware* reconfigurável, ou mesmo sobre circuitos digitais, sintetizem circuitos através de regras de alto nível, aumentando a capacidade dos mesmos de gerar soluções não convencionais para os mais diversos problemas nas áreas da engenharia.

1.5 Organização do documento

Nesta seção será apresentada a organização de cada um dos capítulos deste documento. A capítulo atual apresentou uma contextualização, apresentou também as motivações, os objetivos, as contribuições e os resultados esperados para este trabalho de doutorado.

No capítulo 2 será apresentada a fundamentação teórica, iniciando por uma breve apresentação dos dispositivos de lógica reconfigurável e suas formas de programação. Na sequência serão discutidas a ferramentas de síntese em alto nível e os paradigmas de desenvolvimento. O capítulo termina com uma ampla discussão sobre o Paradigma Orientado a Notificações e suas implementações.

O desenvolvimento do trabalho de doutorado é apresentado no capítulo 3, onde é realizada uma descrição detalhada do PON-HD 1.0. Na sequência, os elementos do conjunto de ferramentas que compõe o PON-HD 1.0 são apresentados em detalhes.

No capítulo 4 são apresentados os experimentos realizados. Cada um dos experimentos é descrito em detalhes, com ênfase no objetivo do experimento e na metodologia adotada. Após cada experimento, são apresentados os resultados obtidos, acompanhados de uma discussão e interpretação dos mesmos.

As conclusões deste trabalho de doutorado são apresentadas no capítulo 5. Inicialmente, é realizada uma discussão das principais contribuições e da viabilidade desta proposta de doutorado. Na sequência, são discutidos resultados obtidos no capítulo anterior à luz dos objetivos. As dificuldades e limitações encontradas no decorrer deste trabalho são então apresentadas. Para concluir, é feita uma discussão sobre os trabalhos futuros.

2 Fundamentação teórica

Neste capítulo serão abordados alguns assuntos que servem como base para as pesquisas realizadas no doutorado. Serão abordados, ainda que de forma bastante resumida, os dispositivos de lógica reconfigurável dado que suas características de operação e programação são fundamentais para o processo de desenvolvimento em alto nível de circuitos.

Por sua vez, o Paradigma Orientado a Notificações (PON) e sua adaptação para o *hardware* são o cerne deste trabalho, assim a abordagem deste tema ocupa boa parte do capítulo. O trabalho realizado por Fernando Augusto de Witt e outros (WITT *et al.*, 2011) deu origem ao PON em *Hardware* Digital e foi objeto de pedido de patente também é abordado, principalmente por ter sido o ponto de partida para as pesquisas realizadas.

Ainda, considerando que o PON-HD 1.0 deve ser uma ferramenta de desenvolvimento de circuitos de alto nível, outras ferramentas de mesma classe são abordadas, enfatizando-se sua classificação e a avaliação dos circuitos por elas gerado. Isto permite a comparação do PON-HD 1.0 com outras ferramentas correlatas.

2.1 Dispositivos de lógica reconfigurável

Os dispositivos de lógica reconfigurável são dispositivos lógicos de uso geral cujo *hardware* pode ser configurado conforme necessário. Existem diversos tipos destes dispositivos dentre os quais pode-se destacar: SPLD (*Simple Programmable Logic Device*); CPLD (*Complex Programmable Logic Device*); e FPGA (*Field Programmable Gate Array*). Cada um desses dispositivos tem suas características e aplicações específicas. As CPLDs por exemplo, possuem baixo custo, enquanto as FPGAs são mais voltadas a projetos complexos e de elevado desempenho (PEDRONI, 2010).

Os dispositivos de lógica reconfigurável, particularmente as FPGAs, apresentam uma grande gama de ferramentas de auxílio ao projeto, que permitem a utilização de bibliotecas de componentes previamente desenvolvidos para acelerar o desenvolvimento de um projeto. De outro modo, estas ferramentas de auxílio de projeto sistematizam o desenvolvimento dos mesmos, facilitando a integração de blocos construtivos e a análise e detecção de erros.

Além dos blocos de lógica configurável é comum aos dispositivos de lógica reconfigurável conterem outros tipos de blocos construtivos, como memória SRAM (*Static*

Random Access Memory), DSP (*Digital Signal Processing*) e PLL (*Phase-Locked Loop*). Estes blocos construtivos são úteis no desenvolvimento de projetos grandes e complexos (PEDRONI, 2010).

Outrossim, considerando a predominância das FPGAs no mercado de dispositivos lógicos reconfiguráveis, deste ponto em diante os dispositivos lógicos reconfiguráveis serão simplesmente denominados FPGAs.

2.2 Programação de dispositivos de lógica reconfigurável - FPGAs

Tradicionalmente os circuitos digitais são projetados através de diagramas esquemáticos. Similarmente, os circuitos das FPGAs podem ser projetados através de diagramas esquemáticos. Porém, esta abordagem é impraticável em grandes projetos. A solução é utilizar linguagens de descrição de *hardware* (*hardware description languages* - HDLs) baseadas em texto (LAMERES, 2017). Este assunto é tratado na próxima subseção.

2.2.1 Linguagens de descrição de *hardware*

As linguagens de descrição de *hardware* evoluíram de forma a permitir além da descrição do *hardware* também sua simulação em diferentes níveis de abstração. A utilização de linguagens de descrição de *hardware* traz algumas vantagens, como a facilidade de trabalhar em um nível mais alto de abstração e não ficar dependente de uma arquitetura específica (LAMERES, 2017).

Outra vantagem da utilização das linguagens de descrição de *hardware* é o processo automatizado de síntese que permite que as ferramentas de CAD criem automaticamente os circuitos a nível de *hardware*. A utilização de bibliotecas contendo componentes e funções predefinidas também é uma característica importante das linguagens de descrição de *hardware* (LAMERES, 2017).

Atualmente existem duas linguagens de descrição de *hardware* dominantes no mercado, notadamente VHDL e Verilog (LAMERES, 2017). Como todo o desenvolvimento deste trabalho foi realizado em VHDL não serão apresentados maiores detalhes sobre Verilog. O VHDL foi escolhido por ser amplamente utilizado pelo grupo de pesquisa do PON.

O VHDL (VHSIC *Hardware Description Language*) é uma linguagem de descrição de *hardware* independente de tecnologia e fabricante, destinada a síntese e simulação de circuitos digitais. Ela é resultado de uma iniciativa do Departamento de Defesa dos Estados Unidos na década de 1980 e foi a primeira linguagem de descrição de *hardware* padronizada pela IEEE (*Institute of Electrical and Electronics Engineers*) (PEDRONI, 2010).

Uma característica importante do VHDL é a utilização efetiva de bibliotecas. Um conjunto de bibliotecas padrão é fornecido junto com as ferramentas de síntese. Ainda, a possibilidade de ampliar as funcionalidades da linguagem através de novas bibliotecas é também importante para o desenvolvimento deste trabalho.

Mesmo com as vantagens das linguagens de descrição de *hardware*, a concepção de projetos grandes e complexos não é tarefa fácil, consumindo muito tempo e recursos (COUSSY *et al*., 2009). Neste contexto e com o objetivo de elevar o nível de abstração na criação de circuitos digitais surgiu a síntese em alto nível, que será discutida na próxima subseção.

2.2.2 Síntese em alto nível

A síntese em alto nível tem o objetivo de elevar o nível de abstração do projeto permitindo assim a rápida geração de *hardware* (COUSSY *et al*., 2009). O *hardware* sintetizado em alto nível normalmente é otimizado em termos de desempenho, área e/ou consumo de energia (COUSSY *et al*., 2009).

A crescente complexidade das aplicações vem forçando o aumento dos níveis de abstração de forma a acelerar a automação tanto da síntese como dos processos de verificação do *hardware*. A síntese em alto nível têm sido fator chave na evolução do processo de projeto ou *design*, permitindo assim que os projetistas explorassem o espaço de *design* de forma eficiente e rápida (COUSSY *et al*., 2009).

Observa-se que a síntese em alto nível não demanda que o projetista domine profundamente os conceitos de eletrônica digital normalmente necessários para a descrição dos circuitos. Espera-se que o projetista tenha capacidade de descrever a solução do problema usando a linguagem de alto nível escolhida, sem a necessidade de acompanhar como a linguagem irá gerar uma possível solução em forma de circuito digital.

Para possibilitar a síntese em alto nível são necessárias ferramentas de síntese em alto nível. A próxima seção apresenta uma discussão mais profunda sobre estas ferramentas.

2.3 Ferramentas de síntese em alto nível

As ferramentas de síntese em alto nível visam preencher a lacuna entre a concepção de algoritmos a serem executados diretamente em *hardware* e o projeto do *hardware* em si (MEEUS *et al*., 2012). Estas ferramentas automatizam as etapas mais complexas da geração de circuitos digitais em *hardware*, utilizando técnicas de otimização, visando melhorar o desempenho e, ao mesmo tempo, mantendo um elevado nível de abstração.

2.3.1 Importância das ferramentas de síntese em alto nível

As arquiteturas tradicionais de computação vêm encontrando dificuldades em acompanhar a crescente demanda por capacidade de processamento e as FPGAs vêm se mostrando uma opção (CROSBIE, 2010; QIAN *et al*., 2005; BORKAR; CHIEN, 2011). Porém, sua programação é tarefa complexa, normalmente exigindo conhecimentos avançados de *hardware* (AWAD, 2009).

Ademais, a complexidade de implementação dos algoritmos nas FPGAs se acentua pela possibilidade de paralelismo. Isto faz com que a complexidade de implementação em FPGA seja maior do que nos microprocessadores sequenciais, exigindo assim mais esforço dos desenvolvedores (AWAD, 2009). Este paralelismo das operações nas FPGAs, supostamente natural, permitiria transferir tarefas computacionais complexas que requerem movimentação de grandes volumes de dados e alto poder de processamento dos microprocessadores para as FPGAs. Entretanto, programar as FPGAs é em si complexo, ainda mais se for de maneira efetivamente paralela.

As ferramentas de síntese em alto nível vêm resolver esta questão. As linguagens de entrada e a facilidade de utilização destas ferramentas permitem que o projeto de *hardware* seja executado por desenvolvedores sem experiência com projeto de *hardware* (MEEUS *et al*., 2012). O processo de verificação, que é uma tarefa importante e complexa, também é facilitado pelo uso de ferramentas de síntese em alto nível (MEEUS *et al*., 2012). Ainda, a complexidade do trabalho realizado pelas ferramentas de síntese em alto nível está relacionada justamente a natureza paralela das operações realizadas nas FPGAs, permitindo alcançar níveis razoáveis de paralelismo em certas circunstâncias.

Alguns fornecedores de ferramentas de síntese em alto nível alegam que a utilização destas ferramentas diminui a probabilidade de erros, se comparado a codificação em linguagens

tradicionais de descrição de *hardware* como VHDL e Verilog. Isso acontece porque as ferramentas geram os circuitos de uma forma sistematizada que é composta por várias fases de verificação (MEEUS *et al*., 2012).

A título de informação a Tabela 1 apresenta as principais ferramentas de síntese em alto nível em utilização no momento (NANE et al. 2016), apresentando seu fabricante, suas linguagens de entrada e saída, a possibilidade de verificação ou simulação do *design* e o suporte ou não a dados do tipo ponto flutuante.

Tabela 1 - Principais ferramentas de síntese em alto nível e suas características.

Ferramenta	Proprietário	Entrada	Saída	Verificação / Simulação	Ponto Flutuante
eCXite	Y Explorations	C	VHDL / Verilog	Sim	Não
CoDeveloper	Impulse Accelerated	Impulse-C	VHDL / Verilog	Sim	Sim
Catapult-C	Calypto *Design* Systems	C / C++ / SystemC	VHDL / Verilog / SystemC	Sim	Não
Cynthesizer	FORTE	SystemC	Verilog	Sim	Sim
Bluespec	BlueSpec Inc.	Bluespec SystemVerilog	System Verilog	Não	Não
CHC	Altium	Subconjunto C	VHDL / Verilog	Não	Sim
CtoS	Cadence	SystemC / Transaction level modeling / C++	Verilog / SystemC	Não	Não
DK Design Suite	Mentor Graphics	Handel-C	VHDL / Verilog	Não	Não
GAUT	U. Bretagne	C / C++	VHDL	Sim	Não
MaxCompiler	Maxeler	MaxJ	RTL	Não	Sim
ROCCC	Jacquard Comp.	Subconjunto C	VHDL	Não	Sim
Synphony C	Synopsys	C / C++	VHDL / Verilog / SystemC	Sim	Não
Cyber-WorkBench	NEC	BDL	VHDL / Verilog	Sim	Sim
LegUp	U. Toronto	C	Verilog	Sim	Sim
Bambu	PoliMi	C	Verilog	Sim	Sim
DWARV	TU. Delft	Subconjunto C	VHDL	Sim	Sim
VavadoHLS	Xilinx	C / C++ / SystemC	VHDL / Verilog / SystemC	Sim	Sim

Fonte: Adaptado de Nane *et al*., (2016).

2.3.2 As linguagens utilizadas nas ferramentas de síntese em alto nível

A principal característica de uma ferramenta de síntese em alto nível é sua linguagem de entrada. A linguagem utilizada pela ferramenta deve ser compatível com as linguagens de projeto de algoritmos, facilitando assim a migração dos códigos já existentes que executam na forma de *software* em sistema monoprocessado. Uma ferramenta de síntese em alto nível de boa qualidade deve também permitir a captura do projeto com certo nível de abstração. O tamanho do projeto gerado e sua latência após a síntese devem ser razoáveis. Ainda, é igualmente importante que a ferramenta e seus recursos sejam fáceis de usar (MEEUS *et al*., 2012).

Por outro lado, as linguagens de programação tradicionais não são adequadas para o desenvolvimento de projetos envolvendo paralelismo (GUPTA; SOHI, 2011; SIMÃO; STADZISZ, 2009). Assim, as linguagens tradicionais não seriam ideais para aplicações que visem explorar as potencialidades das FPGAs (WINDH *et al*., 2015). Não obstante, grande parte da comunidade desenvolvedora segue usando ferramentas de sínteses baseadas em linguagens usuais (MEEUS *et al*., 2012).

As linguagens baseadas em C ou C++ não são as mais apropriadas para as ferramentas de síntese em alto nível, devido principalmente à falta de informação temporal no código e a falta de um modelo de concorrência/paralelismo apropriado (WINDH *et al*., 2015). Porém, a adoção de linguagens baseadas em C ou C++ traz um bom nível de abstração, facilitando a adaptação dos algoritmos existentes e permitindo uma curva de aprendizado mais rápida (WINDH *et al*., 2015).

Existem linguagens específicas que são cada vez mais usadas para essa finalidade. São exemplos de tais linguagens VHDL, Verilog, SystemC e Bluespec (WINDH *et al*., 2015). No entanto, essas linguagens dificultam a reutilização ou a adaptação de algoritmos originalmente escritos em linguagens como C e C++ (WINDH *et al*., 2015). Além disso, essas linguagens são mais difíceis de se usar que as tradicionais, pois obrigam o desenvolvedor a ter um bom conhecimento do *hardware* e estar ciente do paralelismo, expressando-o explicitamente.

Na realidade, existem várias linguagens com características melhores ou piores conforme o contexto e a aplicação. Como as ferramentas de síntese em alto nível variam muito em relação à linguagem de entrada, é importante escolher a ferramenta adequada para cada projeto (WINDH *et al*., 2015). O ideal seria uma linguagem de altíssimo nível que abstraia o

paralelismo, gerando-o automaticamente para o desenvolvedor. Entretanto, até agora não há uma solução definitiva para tornar mais fácil a programação de FPGAs (WINDH *et al*., 2015).

2.3.3 Classificação das ferramentas de síntese em alto nível

Para melhor compreender o funcionamento das ferramentas de síntese em alto nível é apropriado classificá-las segundo suas características. As principais características de uma ferramenta de síntese em alto nível são: facilidade de implementação, nível de abstração, tipos de dados, ajuste do *design*, capacidade de verificação, qualidade dos resultados, documentação e curva de aprendizado (MEEUS *et al*., 2012). Meeus *et al.* (2012) classificou as principais ferramentas disponíveis na época de sua análise segundo estes critérios utilizando um sistema de pontuação que vai de 0 a 5 pontos.

A classificação da **facilidade de implementação** está intimamente relacionada com a linguagem de programação utilizada. As ferramentas de síntese em alto nível visam preencher a lacuna que existe entre o projeto (*design*) de algoritmos e o projeto de *hardware*, permitindo assim que desenvolvedores com pouca experiência em circuitos digitais projetem *hardware*. Para isso é preferível que a linguagem utilizada seja próxima das linguagens em que os algoritmos são tradicionalmente escritos, como C ou C++, evitando assim que se tenha que reescrever tudo em uma linguagem diferente (MEEUS *et al*., 2012). Para este critério foram atribuídos 5 pontos às ferramentas em que o código fonte original poderia ser usado diretamente em sua linguagem nativa com pequenas ou nenhuma modificação. Quanto mais modificações são necessárias no algoritmo menor sua pontuação. É importante observar que Meeus *et al*. (2012) não estão preocupados neste quesito com a expressividade da linguagem utilizada, mas sim na portabilidade dos códigos já existentes.

A classificação segundo o **nível de abstração** fornecido pela ferramenta de síntese em alto nível também é importante. Não é desejável forçar o desenvolvedor a utilizar linguagens direcionadas ao *design* de *hardware* que carecem de expressividade e flexibilidade. Linguagens de programação excessivamente restritivas podem tornar a implementação de um determinado comportamento bastante difícil (MEEUS *et al*., 2012). Assim é importante que o desenvolvedor possa expressar o comportamento desejado da forma mais livre possível. É claro que algumas restrições são necessárias, porém, estas restrições não devem dificultar a descrição do comportamento dos algoritmos (MEEUS *et al*., 2012). Para este critério 5 pontos são atribuídos a ferramentas que permitem a descrição de algoritmos facilmente, sem a preocupação de

sincronismo e temporização. A pontuação vai decaindo e atinge o mínimo para ferramentas onde o algoritmo deve ter seu sincronismo ou temporização explícito no código-fonte de alto nível ou o *design* é realizado a nível de blocos (MEEUS *et al*., 2012).

Quando se trata de *design* de *hardware* o conceito de **tipos de dados** é um tanto diferente. *Softwares* executados por um processador geralmente possuem tipos de dados bem definidos, como inteiros ou ponto flutuante. Estes tipos de dados estão geralmente relacionados com as instruções que o processador é capaz de executar. No *hardware*, no entanto, o tipo de dado básico é o *bit* e o desenvolvedor tem toda a liberdade de formar conjuntos de bits de qualquer tamanho para representar qualquer tipo de dado. Em consequência, as ferramentas de síntese em alto nível devem permitir ao desenvolvedor a utilização de vários tipos de dados, facilitando assim o *design* dos algoritmos. Ademais, suporte a tipos de dados mais complexos como ponto fixo e ponto flutuante é importante na transcrição dos algoritmos já existentes para o *hardware* (MEEUS *et al*., 2012). Para este critério são atribuídos 5 pontos a ferramentas com suporte tanto o ponto flutuante como o ponto fixo. A pontuação decai conforme o suporte a tipos de dados diminui (MEEUS *et al*., 2012).

O critério relacionado a **ajustes no *design*** diz respeito a flexibilidade que a ferramenta fornece com relação a ajustar como o código deve ser gerado. Deve ser possível escolher entre otimizações para velocidade ou para consumo de recursos por exemplo. Meeus *et al.* (2012) atribuiu 5 pontos neste critério a ferramentas que permitem fazer estes ajustes através de interface gráfica ou *scripts*. A pontuação vai diminuindo conforme estes ajustes ficam dificultados ou são inexistentes nas ferramentas.

Levando em consideração o tamanho e a complexidade dos projetos de *hardware* atuais, a **capacidade de verificação** é uma tarefa importante no ciclo do projeto. As ferramentas de síntese em alto nível devem fornecer mecanismos de teste e simulação de forma a facilitar o processo de validação do *design*. A verificação pode ser facilitada através da geração automática de *testbenches* e da integração com ferramentas proprietárias de simulação (MEEUS *et al*., 2012). Para este critério foram atribuídos 5 pontos a ferramentas que geram bancos de testes fáceis de usar. A pontuação diminui conforme o suporte à verificação também diminui na ferramenta (MEEUS *et al*., 2012).

Outro fator importante é a **qualidade dos resultados** gerados pelas ferramentas de síntese em alto nível. Características como uso de recursos da FPGA e latência dos circuitos gerados ajudam a determinar qualidade dos circuitos gerados por estas ferramentas. É desejável que os circuitos sejam rápidos e utilizem poucos recursos das FPGAs. É também importante que estas ferramentas permitam ajustar através de parâmetros as restrições do projeto,

permitindo assim o equilíbrio entre consumo de recursos e latência, específicos para cada caso (MEEUS *et al*., 2012). Com relação a qualidade dos resultados Meeus *et al.* (2012) classificou as ferramentas segundo o tamanho dos circuitos gerados para os algoritmos testados. Foram atribuídos 5 pontos a ferramenta que gerou o menor circuito e a pontuação decaiu conforme o tamanho do circuito aumentou. É importante salientar aqui que para os resultados de Meeus *et al.* (2012) a velocidade de *clock* e a latências dos circuitos não foram levados em consideração.

Como estas ferramentas geralmente possuem interfaces complexas com características específicas é conveniente que seja disponibilizada uma **documentação** apropriada. Esta documentação deve cobrir não apenas a ferramenta em si, mas também deve explicar como escrever adequadamente o código fonte dos projetos (MEEUS *et al*., 2012). Neste quesito foram atribuídos 5 pontos as ferramentas com documentação completa e fácil de entender. A pontuação foi diminuindo conforme a documentação foi ficando mais complicada de entender ou inexistente (MEEUS *et al*., 2012).

O último critério é a **curva de aprendizado**. É desejável que as ferramentas de síntese em alto nível possuam uma curva de aprendizado razoavelmente plana, facilitando assim sua adoção. Uma interface gráfica intuitiva que guia o desenvolvedor durante o processo de *design* facilita este processo (MEEUS *et al*., 2012). Foram atribuídos 5 pontos para ferramentas com uma curva de aprendizado mais plana, onde as restrições da linguagem sejam fáceis de entender e as mensagens de erro apresentem clareza. A pontuação diminui conforme a ferramenta se mostra difícil de dominar (MEEUS *et al*., 2012).

A Tabela 2 apresenta o resultado das avaliações realizadas por Meeus *et al*., (2012).

Tabela 2 - Classificação das ferramentas de síntese em alto nível.

	Facilidade de implementação (Algoritmos C, C++)	Nível de abstração	Tipos de dados	Ajuste do *design*	Capacidade de verificação	Qualidade dos resultados	Documentação	Curva de aprendizado
AccelDSP	*****	****	*****	****	****	**	***	*****
Agility Compiler	****	***	*	***	*	**	****	**
AutoPilot	*****	*****	****	*****	*****	*****	*****	*****
BlueSpec	***	***	*	*	-	*****	****	**
Catapult C	*****	*****	****	*****	*****	****	*****	*****
Compaan	****	*****	*****	****	*	*	**	***
C to Silicon	****	****	****	**	****	***	****	*
CyberWorkBench	****	*****	****	*****	****	*****	***	****
DK Design	***	***	****	**	***	*	***	**
Impulse CoDeveloper	**	***	****	***	*	*	*	***
ROCCC	*	****	*	***	*	*	****	**
Synphony C	*****	*****	****	***	*****	***	****	****

Fonte: Adaptado de Meeus *et al*., (2012).

2.3.4 Qualidade dos resultados

É ainda importante medir a qualidade dos resultados das ferramentas de síntese em alto nível. Para a avaliação da qualidade de um *hardware* gerado por uma ferramenta de síntese em alto nível, três métricas podem ser utilizadas: a máxima frequência de operação, que é a máxima frequência em MHz em que um circuito pode operar; a latência, que é o número de ciclos de *clock* necessários para completar uma tarefa; e o uso de recursos da FPGA, incluindo elementos lógicos, blocos DSP, blocos de memória e outros recursos de *hardware* (NANE *et al.*, 2016).

A frequência de operação é uma medida importante, pois determina a velocidade com que o circuito pode operar, impactando assim diretamente no seu desempenho. Esta frequência de operação é determinada pelo formato do circuito, que por sua vez é determinado pela ferramenta de síntese. Assim, as estratégias adotadas pelas ferramentas de síntese são determinantes no desempenho dos circuitos gerados.

Outro fator que impacta no desempenho é a latência. Assim como a frequência de operação, a latência também é determinada pelas estratégias adotadas pelas ferramentas de síntese na geração dos circuitos. Uma mesma tarefa pode ser executada de várias formas diferentes, por vários circuitos diferentes. Cada circuito pode necessitar de um número diferente de ciclos de *clock* para realizar a tarefa e assim demorar mais ou menos (NANE *et al.*, 2016).

Enquanto a frequência de operação e a latência tem relação com o desempenho, a utilização de recursos da FPGA tem relação com o custo do circuito. Quanto mais recursos o circuito necessita para executar determinada tarefa, maior será o tamanho da FPGA necessária para contê-lo e consequentemente maior será seu custo. Também neste caso, as estratégias adotadas pelas ferramentas de síntese determinam a utilização de recursos da FPGA em um determinado circuito.

Existe uma relação de compromisso entre o desempenho (máxima frequência de operação e latência) e a utilização de recursos da FPGA. Para melhorar o desempenho é necessário realizar mais tarefas ao mesmo tempo, assim mais circuitos são necessários e consequentemente mais recursos são consumidos. Este conceito é válido tanto para elementos lógicos como para blocos DSP e blocos de memória. As ferramentas de síntese em alto nível devem permitir ao desenvolvedor ajustar os parâmetros de síntese de forma a escolher entre desempenho e consumo de recursos conforme as necessidades de cada projeto (MEEUS *et al.*, 2012).

Outro ponto importante relacionado a qualidade dos resultados das ferramentas de síntese em alto nível é a independência de uma arquitetura específica. Desta forma é possível escolher a arquitetura ou o fabricante mais apropriado ao projeto. Isto é possível separando a etapa de *design* da etapa de síntese do *hardware*. Enquanto o *design* é realizado em alto nível através de ferramentas e linguagens genéricas a síntese é realizada por ferramentas específicas de cada arquitetura ou fabricante (MEEUS *et al*., 2012).

2.4 Os paradigmas de desenvolvimento

Para que se possa compreender melhor o Paradigma Orientado a Notificações (PON), elemento central desta pesquisa, é importante que se discuta um pouco mais sobre paradigmas de desenvolvimento. Esses paradigmas de desenvolvimento surgiram para ou no âmbito de *software*, com suas linguagens e mesmo métodos de desenvolvimento salientando a programação. Entretanto, naturalmente, tais paradigmas acabaram sendo reutilizados para *hardware* configurável como FPGA, quando começou a se utilizar linguagens para a elaboração de *hardware* e não mais "apenas" blocos construtivos (WANG; PING; WANG, 2006).

David Watt (2004) explica que um paradigma de desenvolvimento, consiste na seleção de um conjunto de conceitos de programação, como tipo de dados, variáveis, escopo, abstração, concorrência e controle, utilizados em sinergia para formar um estilo de programação. Peter Van Roy e Seif Haridi (2004), por sua vez, definem que um paradigma de programação é um sistema formal que determina como a programação é realizada.

Mais do que possuir seu próprio conjunto de técnicas e conceitos para programação, um paradigma de desenvolvimento possui sua própria forma de estruturar o pensamento na construção dos programas (VAN ROY; HARIDI, 2004; VAN ROY, 2009; BANASZEWSKI, 2009). Neste âmbito, buscando classificar os paradigmas de desenvolvimento, Peter Van Roy e Seif Haridi (2004) desenvolveram uma taxonomia para estes paradigmas. A Figura 6 apresenta um diagrama que ilustra esta taxonomia.

Para classificar os paradigmas, segundo esta taxonomia, são estabelecidos quatro conceitos relacionados a linguagem de programação, que são: registros (*record*), recipientes com escopo léxico (*closures*), independência (concorrência) e estado nomeado (*named state*).

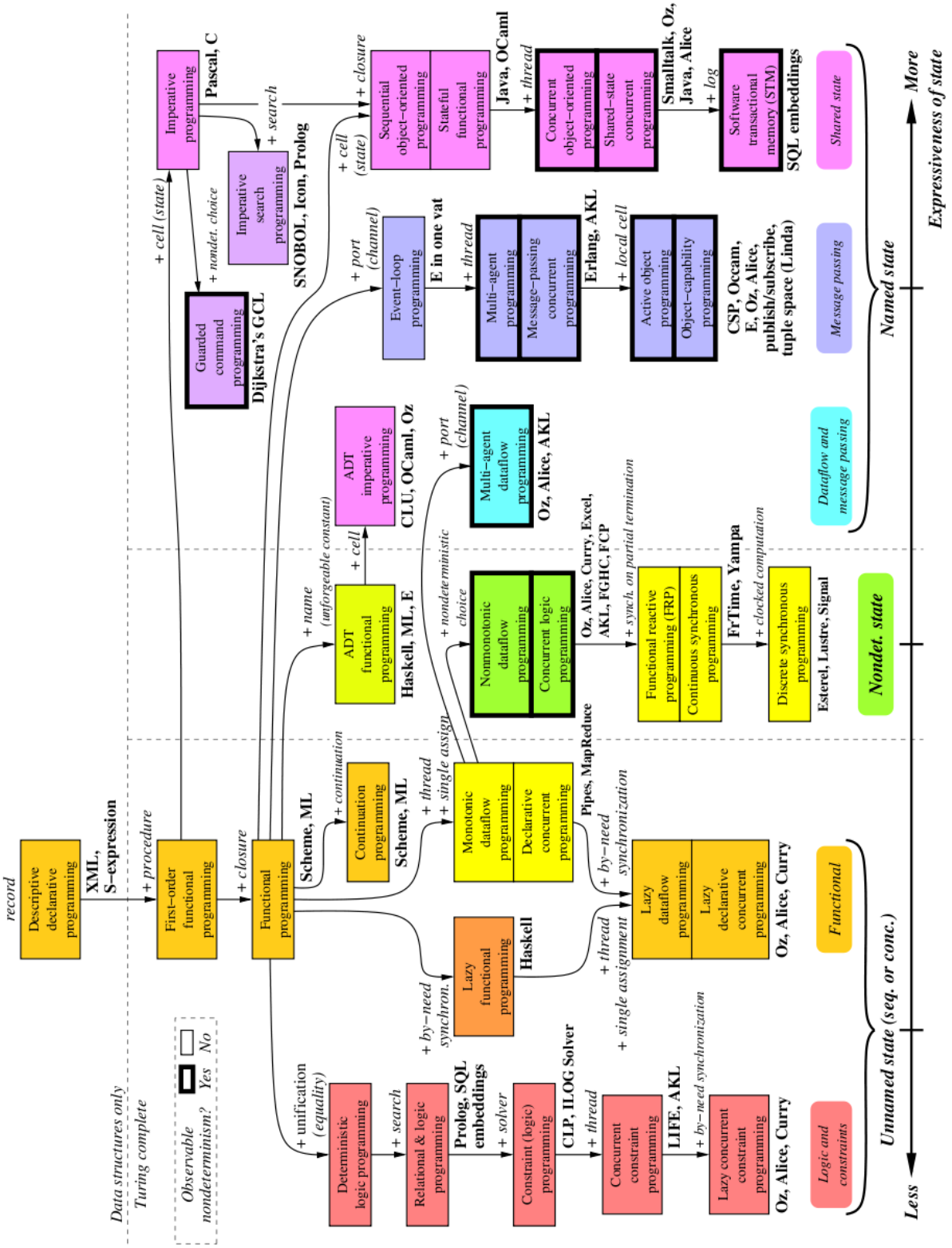

Figura 6 - Taxonomia de paradigmas de programação.

Fonte: Adaptado de VAN ROY, 2009.

Registros são as estruturas de dados que através de referências permitem indexar os itens de dados que compõe as estruturas de dados (VAN ROY, 2009). Recipientes com escopo léxico (*closures*) são os procedimentos, módulos, objetos, funções, classes, e afins nos quais se viabilizam criar blocos e estruturas de controle como SE (*if*) e ciclos repetitivos como ENQUANTO (*while*) (VAN ROY, 2009). A independência, por sua vez é a capacidade de construir *software* em partes independentes, dando suporte à concorrência (VAN ROY, 2009). Ainda um estado nomeado (*named state*) representa um componente computacional com um nome, que possui a capacidade de armazenar informações, sendo variáveis exemplos de componentes com estados nomeados (VAN ROY, 2009).

É também importante salientar que o eixo horizontal da Figura 6 gradua os paradigmas conforme a expressividade do estado nomeado. A esquerda, estão localizados os paradigmas com menor expressividade e a direita os de maior expressividade. Já no eixo vertical os paradigmas são graduados segundo suas características, iniciando na parte superior com as características básicas (*record* e *closure*) e incluindo mais características específicas conforme são localizados mais a baixo.

Não é objetivo deste trabalho se aprofundar em relação aos paradigmas de desenvolvimento, portanto serão discutidos apenas os principais. De fato, os principais paradigmas de programação atuais podem ser classificados de maneira geral e assaz simplista em dois grandes tipos, nomeadamente Paradigma Imperativo (PI) e Paradigma Declarativo (PD) (GABBRIELLI; MARTINI, 2010).

A seguir serão apresentadas as principais características de cada um destes paradigmas, de forma a fundamentar os conceitos apresentados nas próximas seções.

2.4.1 Paradigma Imperativo

O Paradigma Imperativo (PI) pode ser entendido como compreendendo as abordagens chamadas procedimental e a abordagem orientada a objetos. Apesar de algumas diferenças estruturais, ambas se caracterizam por definir *software* como uma sequência lógica de instruções que, quando executada, processa e altera os estados (dados) do programa de acordo uma lógica imperativamente pré-estabelecida.

A forma de programação utilizada no PI induz ao uso de linguagens de programação com muito detalhamento técnico e lógicas complexas para construção dos programas. Existe ainda um acoplamento natural entre a lógica do programa e os estados por ele processados. Este

acoplamento dificulta a distribuição e a paralelização de execução do programa, pois fica muito complicado processar partes do programa de forma simultânea e independente, dada a forte dependência entre estados e lógica no programa (SOMMERVILLE, 2004; PAES e HIRATA, 2008; WATSON *et al.*, 2009).

Para melhor explicar a redundância apresentada por este paradigma, um trecho de programa é apresentado na Figura 7.

```
1 Enquanto (estado4 < valor5)
2   Se (estado1 = valor1)
3     então
4       Procedimento1
5   Fim Se
6   Se (estado2 = valor2 E estado3 = valor3)
7     então
8       Procedimento2
9   Fim se
10  Se (estado1 = valor1 OU estado1 = valor4)
11    então
12      Procedimento3
13  Fim se
14 Fim Enquanto
```

Figura 7 - Exemplo de redundância no Paradigma Imperativo.

Fonte: Adaptado de LINHARES, 2015, p. 54

É possível observar que a comparação entre o “estado1” e o “valor1” acontece duas vezes, na linha 2 e na linha 10. Mesmo as comparações sendo idênticas e mesmo que o valor do “estado1” não tenha sido alterado nas linhas intermediárias, o resultado lógico da primeira comparação não pode ser aproveitado na segunda, dada a estrutura imposta pelo programa. Esta deficiência é conhecida como redundância estrutural. Tal característica produz avaliações idênticas repetidas em função de questões estruturais do programa (FORGY, 1982).

Também é possível observar na linha 1 da Figura 7 um laço de repetição que força a repedida reavaliação de todas as condições “Se”. Estas reavaliações acontecem independentemente de os dados envolvidos nestas condições terem sido alterados ou não. Se não ouve alteração nos dados não se justifica uma nova avaliação. Esta reavaliação desnecessária é conhecida como redundância temporal (LINHARES, 2015).

Ademais, o acoplamento existente neste paradigma também pode ser observado na Figura 7. Mesmo não estando explicito no exemplo, os procedimentos “Procedimento1”, “Procedimento2” e “Procedimento3” podem alterar as variáveis “Estado1”, “Estado2”, “Estado3” ou “Estado4” que por sua vez são utilizadas nas avaliações das relações causais. Esta

estrutura cria uma forte dependência entre os procedimentos e as relações causais para com os dados, provocando ainda uma imposição da ordem em que as avaliações são executadas. Estas características de acoplamento dificultam a distribuição e a execução paralela do código (LINHARES 2015).

Boas práticas de programação como orientação a objetos, orientação a aspectos, orientação a eventos e padrões, todas a luz de princípios de coesão e desacoplamento gerando encapsulamento, podem mitigar tais problemas apresentados no código pedagógico de exemplo. Tais problemas podem ser ainda mais mitigados se a programação for precedida de fases apropriadas de engenharia de *software* a luz daqueles princípios. Ainda que mitigados, tais problemas não são eliminados, continuando a ser um fator crítico para sistemas maiores. Ademais, tais práticas mitigantes exigem conhecimentos mais avançados do desenvolvedor, havendo disparate entre o número de profissionais formados e a demanda crescente (GABBRIELLI; MARTINI, 2010). Essas questões são tratadas em maiores detalhes em trabalhos como (LINHARES *et al*., 2014; SIMÃO *et al*., 2014).

2.4.2 Paradigma Declarativo

O Paradigma Declarativo (PD), por sua vez, pode ser entendido como compreendendo o Paradigma Funcional (PF) e o Paradigma Lógico (PL) (SCOTT, 2000). Neste paradigma o programador se preocupa essencialmente em definir o que deve ser feito, não existindo uma distinção clara entre dados e instruções. Os estados de um programa (fatos) e as relações causais que determinam a atuação sobre os estados (regras), são declarados em um nível elevado de abstração, como dados. A aplicação das regras sobre os fatos é normalmente realizada por um mecanismo de inferência separado do programa, de forma transparente a ele (FORGY, 1982).

Os Sistemas Baseados em Regras (SBR) são exemplos dos conceitos do PD. Nestes sistemas o conhecimento é armazenado em uma base de fatos, sob a qual um mecanismo de inferência aplica um conjunto de regras lógico-causais. Estes sistemas estão fundamentados no processamento cognitivo humano, que similarmente tem acesso ao conhecimento sobre determinado problema e sobre este conhecimento define regras de forma a produzir uma resposta apropriada para o problema (NEWELL; SIMON, 1972 apud BANASZEWSKI, 2009; LINHARES, 2015).

De forma resumida, a forma como as variáveis ou os atributos da base de fatos são alterados é determinada pelas conclusões que as regras abstraem dos próprios atributos

(BANASZEWSKI *et al*., 2007). Assim, os SBRs são geralmente compostos por uma base de fatos, uma base de regras e uma máquina de inferência. Estas bases de fatos e de regras são armazenadas em repositórios ou em memória. Esta máquina de inferência é separada das bases e define um modelo de processamento que aplica as regras sobre os fatos, gerando novos fatos ou atualizando os existentes criando assim um ciclo de inferência (LINHARES, 2015).

Assim como no PI, os SBR possuem problemas de acoplamento e distribuição, isto acontece porque os mecanismos de inferência são normalmente monolíticos e centralizados. Existem várias iniciativas de paralelização e distribuição dos SBR, mas tais iniciativas são limitadas pelas características intrínsecas da arquitetura e do processo de inferência (BANASZEWSKI, 2009). A utilização de estruturas de dados dispendiosas também é uma das causas de sobrecarga de processamento nestes sistemas (LINHARES *et al*., 2014; SIMÃO *et al*., 2014).

Maiores detalhes sobre os paradigmas de desenvolvimento e suas deficiências podem ser encontrados nas obras referenciadas (GABBRIELLI; MARTINI, 2010; LINHARES *et al*., 2014; SIMÃO *et al*., 2014).

2.5 Fundamentos do Paradigma Orientado a Notificações (PON)

Nas próximas seções é apresentado o Paradigma Orientado a Notificações (PON). Neste contexto serão tecidas algumas comparações gerais entre o PON e outros paradigmas de programação. Ainda, a estrutura e o processo de inferência do PON serão discutidos em detalhes. Também serão apresentadas as várias plataformas onde o PON é materializado (i.e. implementado) e utilizado, enfatizando o PON para *hardware* digital que serve como base para este trabalho.

2.5.1 Origens do PON

Os conceitos que fundamentaram o surgimento do Paradigma Orientado a Notificações (PON) foram inicialmente propostos por Jean Marcelo Simão em sua dissertação de mestrado (SIMÃO, 2001) e em artigos relacionados (SIMÃO; STADZISZ, 2002). Eles foram ainda mais aprofundados e detalhados em sua tese de doutorado (SIMÃO, 2005).

Nestes trabalhos o autor propõe mecanismos de Controle Orientado a Notificações (CON) para sistemas de produção, mais precisamente para sistemas de manufatura. Estes mecanismos utilizam um metamodelo de controle discreto e holônico do CON de forma a aplicar instâncias deste sobre simulação realística de sistemas de manufatura em uma ferramenta chamada ANALYTICE II, ferramenta esta desenvolvida pelo seu grupo de pesquisa. Isso permitiu simular realisticamente (quase emular por assim dizer) os chamados Sistemas Inteligentes de Manufatura, também chamados de Sistemas Holônicos de Manufatura.

Neste metamodelo de CON, as colaborações entre entidades "inteligentes" de manufatura são organizadas inclusive por meio de uma abordagem holônica (i.e. integradora e sinérgica), na qual cada entidade de manufatura é integrada a um sistema computacional por meio de um recurso virtual e tem suas colaborações regidas por um sistema de controle "inteligente". Este metamodelo de CON trabalha via colaboração das entidades holônicas de manufatura com entidades holônicas de controle, tudo por meio de notificações pontuais, sendo tal mecanismo de notificação detalhado na próxima subseção.

Na verdade, este metamodelo de controle holônico atua sobre a ferramenta ANALYTICE II de forma tal a modifica-la para ser um simulador fidedigno de Sistemas de Manufatura Holônica. Este ferramental foi submetido a diversos testes e análises, via instância de controle holônico orientado a notificações, com os resultados documentados em diversos trabalhos (SIMÃO, 2005; SIMÃO *et al*., 2008; SIMÃO, 2001; SIMÃO; STADZISZ, 2002; SIMÃO; STADZISZ; MOREL, 2006).

Os sistemas discretos de controle e manufatura, holônicos ou não, não foram a única aplicação deste metamodelo de CON. O autor percebeu, subsequentemente, que poderiam aplicá-lo em várias áreas, como: concepção de controle discreto em geral; inferências discretas em geral; e mesmo *software* de maneira genérica. Desta maneira, a aplicação do metamodelo de notificações à construção de programas foi, a luz do tempo, denominada Paradigma Orientado a Notificações (PON) (SIMÃO; STADZISZ, 2008).

2.5.1.1 O PON como paradigma de desenvolvimento

A essência do PON é seu processo de inferência baseado em sub-entidades pequenas, inteligentes e desacopladas, que colaboram entre si através de notificações pontuais a fim realizar todo o cálculo lógico-causal. Outramente dito, como paradigma de desenvolvimento, o PON apresenta um novo conceito para conceber e executar aplicações, com base em entidades-

regras, compostas de sub-entidades colaborativas, as quais são notificáveis a partir de entidades factuais. (SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009).

O objetivo do PON como paradigma de desenvolvimento é tornar mais fácil a tarefa de criação e organização de projetos e artefatos no tocante a aplicativos ou sistemas. Além da facilidade de composição de artefatos (como código e circuitos) o PON busca tornar estes otimizados e distribuíveis (SIMÃO; STADZISZ, 2009). O PON também busca resolver problemas de redundância e centralização, presentes nas abordagens atuais de processamento lógico-causal, resolvendo assim as questões de mau uso da capacidade de processamento e o acoplamento dos paradigmas atuais (SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009).

2.5.2 Estrutura e processo de inferência do PON[1]

Com o objetivo de exemplificar os conceitos fundamentais do PON, um exemplo de aplicação foi elaborado. Este exemplo será apresentado de forma completa na seção 3.1, onde se fala do PON em *hardware* digital, porém, uma regra lógico-causal deste exemplo é apresentada na Figura 8 dado que nada o obsta. Esta regra lógico-causal, é apresentada em um formato amigável ou em alto nível, sendo tratada no PON com uma entidade computacional especial chamada *Rule* (SIMÃO; STADZISZ, 2008). Assim, no PON, as expressões causais são representadas por regras lógico-causais usuais, as quais são naturais aos desenvolvedores dos paradigmas atuais de programação ou até mesmo naturais em si.

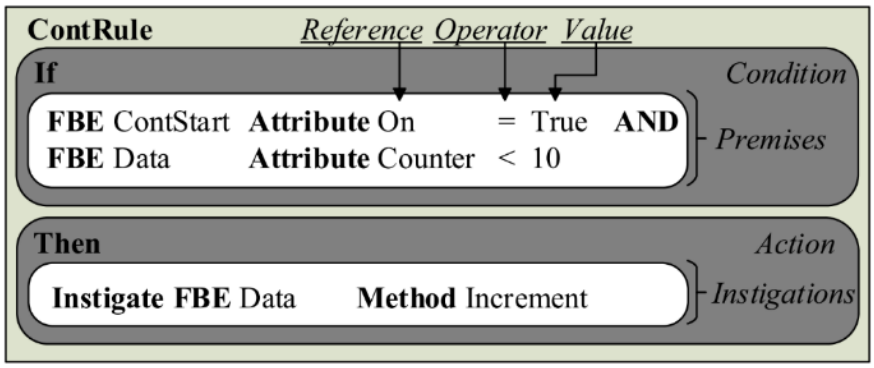

Figura 8 - A representação de uma *Rule.*

Fonte: Autoria própria.

[1] Esta subseção está fundamentada principalmente nos trabalhos de Simão e Stadzisz (2008), Simão e Stadzisz (2009), Linhares *et al*. (2014) e Simão *et al*. (2014).

Conforme a teoria do PON, cada *Rule* é composta por duas entidades, uma *Condition* e uma *Action* (SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009). Esta estrutura e estruturas relacionadas são mostradas através do diagrama de blocos em SysML na Figura 9.

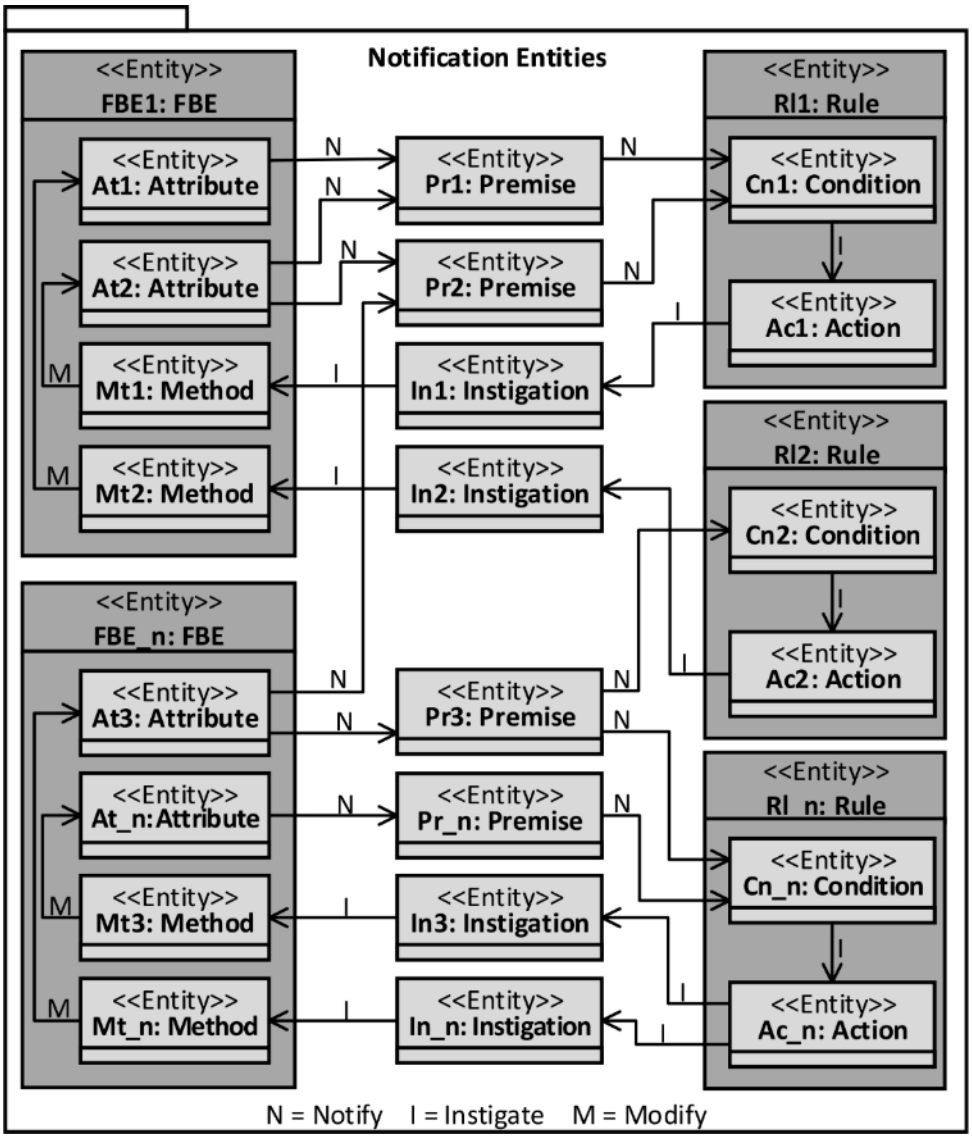

Figura 9 - Diagrama de blocos em SysML do ciclo de notificações do PON.

Fonte: Autoria própria.

Em uma *Rule*, a *Condition* é responsável pela tomada de decisão, enquanto a *Action* é responsável pela conclusão relativa a eventual decisão positiva. Assim, pode-se dizer que a *Condition* e a *Action* são entidades que trabalham juntas para representar e tratar o conhecimento causal da *Rule*. Para que isso aconteça, a *Condition* avalia os estados de elementos factuais, enquanto a *Action* instiga a execução dos serviços relativos a *Rule* (SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009). No exemplo apresentado na Figura 8, os

elementos factuais avaliados pela *Condition*, são “ContStart” e “Data”. A *Action*, por sua vez, instiga um método chamado “Increment”.

No PON, cada elemento avaliado nas *Rules* é representado por um tipo de entidade chamado *Fact_Base_Element (FBE)* (SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009). No exemplo, “ContStart” e “Data” são exemplos de *FBEs*. Ainda, cada *FBE* é composto por um conjunto não vazio de ‘atributos’. No exemplo, “ContStart” e “Data” são exemplos de *FBEs.* Cada atributo, por sua vez, é representado por um outro tipo de entidade chamada *Attribute*. No exemplo, “On” e “Counter” são respectivamente *Attributes* das *FBEs* “ContStart” e “Data”.

Os estados dos *Attributes* são fatos analisáveis em um processo de inferência nas *Conditions*. Esta avaliação dos estados dos *Attributes* se dá por meio de outras entidades chamadas *Premises* (SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009). Na *Rule* do exemplo, a *Condition* é composta por duas *Premises*. A primeira analisa se o *Attribute* “On” é “*True*” e a segunda analisa se o *Attribute* “Counter” é menor que 10. A relação entre estes elementos do PON é facilmente observada no diagrama da Figura 9.

Para que uma *Rule* seja aprovada, ou seja, se torne verdadeira, é necessário que cada uma das *Premises* da *Condition* sejam inferidas como verdadeira. Assim a *Condition* torna-se verdadeira aprovando sua *Rule* (SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009). Uma *Rule* aprovada pode então ativar sua *Action*.

A *Action*, por sua vez, é composta por entidades chamadas *Instigations*. Na *Rule* do exemplo, a *Action* contém uma única *Instigation*. A função das *Instigations* é instigar *Methods*. Cada *Method*, por sua vez, é outra entidade da *FBE* que permite a execução dos serviços da *mesma*. Como o *Method* é responsável pela parte execucional, sua chamada costuma alterar os estados dos *Attributes* de uma *FBE*, alimentando assim o processo de inferência (LINHARES *et al*., 2014; SIMÃO; STADZISZ, 2008; SIMÃO *et al*., 2014).

É possível afirmar que o processo de inferência do PON é diferenciado, uma vez que as *Rules* têm sua inferência efetuada pela colaboração ativa de suas entidades notificantes (SIMÃO; STADZISZ, 2008). O conceito de notificações pontuais entre elementos colaboradores permite que para cada alteração no estado de um *Attribute* de uma *FBE*, a avaliação do estado deste *Attribute* ocorra apenas nas *Premises* relacionadas e então somente nas *Conditions* relacionadas e pertinentes (LINHARES *et al*., 2014; SIMÃO *et al*., 2014).

2.5.3 Inferência Orientada a Notificações do PON[2]

O grande diferencial do PON é o processo de inferência orientada a notificação. Para compreender esse processo é necessário compreender a natureza e a composição da *Premise*. A *Premise* relaciona o estado de um ou dois *Attributes,* entregando como resultado um valor booleano.

Cada *Premise,* como pode ser visto no exemplo da na Figura 8, é composta por: (a) uma referência ao valor de um *Attribute*, que é chamado de *Reference* e recebido por notificação; (b) um operador relacional, que é chamado de *Operator* e é usado para fazer comparações; e (c) outro valor, que é chamado de *Value* e pode ser uma constante ou o valor de outro *Attribute*, recebido também por notificação (LINHARES *et al*., 2014; SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009; SIMÃO *et al*., 2014).

Quando um ou dois *Attributes* (ou seja, *Reference* ou *Value*) notificam uma *Premise,* um cálculo lógico é realizado. Este cálculo é realizado comparando-se *Reference* e *Value*, usando o *Operator*. Quando o valor booleano de uma *Premise* notificada é alterado, as *Conditions* relacionadas também são notificadas, assim uma *Premise* colabora com a avaliação causal de um conjunto de *Conditions* (LINHARES *et al*., 2014; SIMÃO; STADZISZ, 2008; SIMÃO; STADZISZ, 2009; SIMÃO *et al*., 2014).

A *Condition* notificada, por sua vez, calcula seu valor booleano por meio de um operador lógico, como uma conjunção ou uma disjunção, dos valores das *Premises* em questão. No caso de uma conjunção (operador AND), por exemplo, quando todas as *Premises* que integram a *Condition* são satisfeitas, a *Condition* em si é satisfeita. No caso de uma disjunção (operador OR), também como exemplo, quando uma das *Premises* que integra a *Condition* é satisfeita, a *Condition* em si é satisfeita. Utilizando ainda o conceito de notificações, uma vez que uma *Condition* é satisfeita, ela notifica a respectiva *Rule* para que seja aprovada.

Uma *Rule* quando aprovada notifica sua *Action*, a qual por sua vez notifica cada uma das suas *Instigations*. As *Instigations* então notificam cada um de seus *Methods*, instigando-os a executar. É importante salientar que quando uma *Rule* é aprovada, é necessário verificar se não existe nenhum conflito com outra *Rule* antes que a mesma possa ser executada (LINHARES *et al*., 2014; SIMÃO; STADZISZ, 2008; SIMÃO *et al*., 2014). O diagrama de blocos ilustrado

[2] Esta subseção está fundamentada principalmente nos trabalhos de Simão e Stadzisz (2008), Simão e Stadzisz (2009), Linhares *et al.* (2014) e Simão *et al.* (2014).

na Figura 9 apresenta a colaboração exercida por meio de notificações entre as entidades do PON. A Figura 4 apresentada na introdução e repetida aqui na Figura 10 e representa de outra forma o ciclo de inferência por notificações do PON.

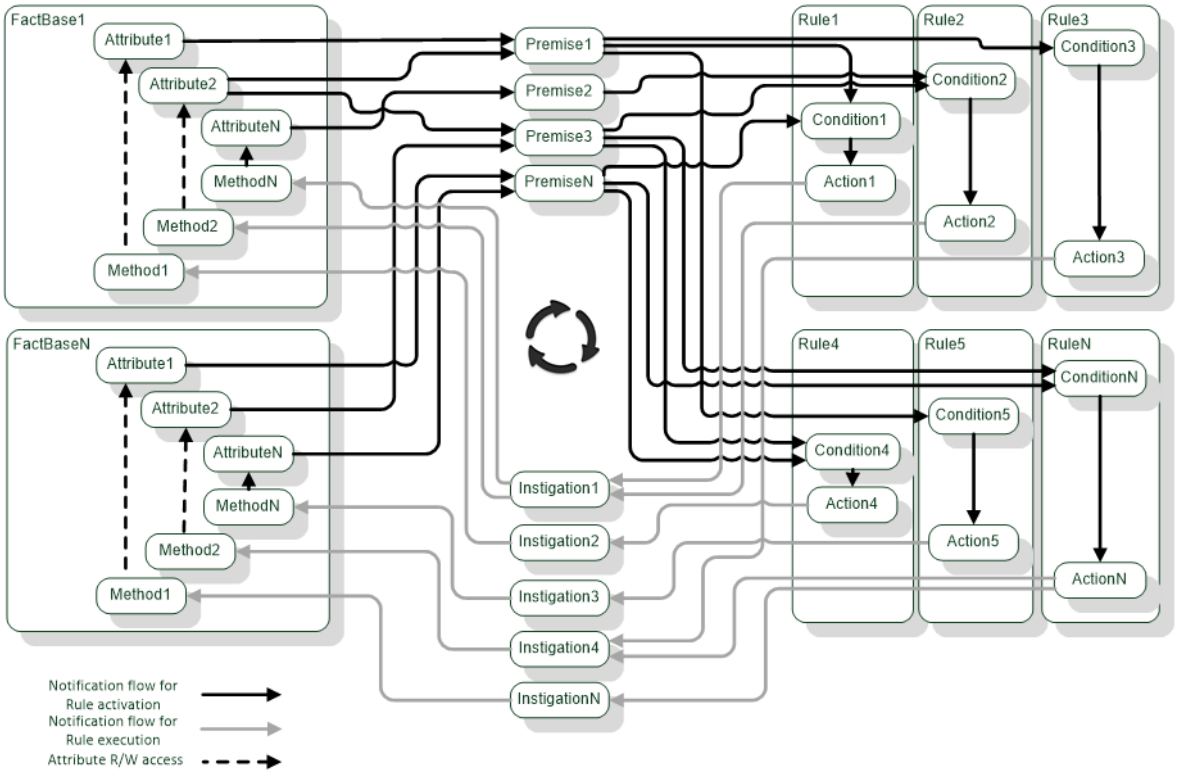

Figura 10 - Ciclo de inferência por Notificações do PON.
Fonte: Adaptado de Linhares (2015).

Uma característica que deve ser ressaltada com relação as entidades colaborativas do PON é que cada elemento notificante (por exemplo, um *Attribute*) conhece quais elementos estão interessados em seu resultado (por exemplo, *Premises*). Assim somente os elementos que realmente necessitam destes resultados são notificas evitando desperdício de processamento (LINHARES *et al*., 2014; SIMÃO; STADZISZ, 2008; SIMÃO *et al*., 2014).

2.5.4 A natureza do PON[3]

Graças à natureza do PON e sua cooperação por meio de notificações pontuais entre entidades factuais, lógicas e causais, o PON evita os dois principais tipos de redundâncias verificadas em muitas linguagens de programação. Isto acontece, pois no PON, cada entidade factual *Attribute* é avaliado por um conjunto de entidades lógicas e causais, ou seja, as *Premises* e as *Conditions*, apenas na mudança de seu estado. Evitando tais redundâncias, é possível evitar desperdício de processamento e mesmo acoplamento entre partes do código (LINHARES *et al*., 2014; SIMÃO; STADZISZ, 2008; SIMÃO *et al*., 2014).

Como discutido anteriormente (na subseção 2.4.1), os dois principais tipos de redundâncias são a redundância temporal e a redundância estrutural. A redundância temporal consiste na avaliação desnecessária de expressões lógico/causais. Por exemplo, uma expressão dada "se (A > 3) então" não aprovada devido ao valor atual da variável "A", sendo reavaliada outra(s) vez(es) em um laço de repetição sem ter havido mudanças na variável "A". A redundância estrutural, por sua vez, é a repetição de expressões lógicas no âmbito de expressões causais. Por exemplo, uma expressão lógica dada "(A > 3)" repetida no âmbito de duas ou mais expressões "se então" (LINHARES *et al*., 2014; SIMÃO; STADZISZ, 2008; SIMÃO *et al*., 2014).

O PON resolve a questão da redundância temporal eliminando a pesquisas sobre elementos passivos. Como foi apresentado, os *Attributes* são elementos reativos em relação a atualização de seu estado. Além disso, os *Attributes* notificam pontualmente apenas as *Premises* que estão interessadas na atualização de seu estado. Assim, evita-se que outras partes e até mesmo outras expressões causais sejam avaliadas ou reavaliadas desnecessariamente (LINHARES *et al*., 2014; SIMÃO; STADZISZ, 2008; SIMÃO *et al*., 2014).

A natureza do PON também resolve a questão da redundância estrutural, pois uma *Premise* pode ser compartilhada por duas ou mais *Conditions*. Assim, a *Premise* realiza o cálculo lógico apenas uma vez e compartilha o resultado lógico apenas com as *Conditions* relacionadas, evitando assim reavaliações desnecessárias e a repetição de *Premise*s com o mesmo conteúdo (LINHARES *et al*., 2014; SIMÃO; STADZISZ, 2008; SIMÃO *et al*., 2014).

[3] Esta subseção está fundamentada principalmente nos trabalhos de Simão e Stadzisz (2008), Simão e Stadzisz (2009), Linhares *et al.* (2014), Simão *et al.* (2014), Peters *et al.* (2012), Belmonte *et al.* (2012), Talau (2016) e Barreto (2016).

Como as redundâncias estruturais e, principalmente, as redundâncias temporais são evitadas com o PON, sua utilização possibilita obter uma melhora no desempenho geral das aplicações (SIMÃO; STADZISZ, 2009; FERREIRA, 2015; LINHARES *et al*., 2014; RONSZCKA *et al.*, 2015). Além de melhorar o desempenho, o PON é também aplicável no desenvolvimento de aplicações paralelas ou distribuídas. Isto ocorre devido ao "desacoplamento" de suas entidades (ou acoplamento mínimo entre entidades para ser mais preciso) permitido pela orientação a notificações. Quando uma entidade é notificada, não há grande diferença, se está se encontra na mesma região de memória, na memória do mesmo computador ou na mesma sub rede, é apenas necessário saber o endereço da entidade notificável (LINHARES *et al*., 2014; SIMÃO; STADZISZ, 2008; SIMÃO *et al*., 2014; PETERS *et al.*,2012; BELMONTE *et al.*, 2012; TALAU, 2016; BARRETO, 2016).

2.5.5 Resolução de Conflitos

Quando duas ou mais *Rules* necessitam acessar um mesmo *FBE* e demandam exclusividade de acesso sobre ele, ocorrem conflitos de execução. Neste caso, as *Rules* concorrem para adquirir acesso exclusivo a este *FBE*, sendo que somente uma destas *Rules* deve ser executada por vez afim de garantir determinismo e consistência (BANASZEWSKI, 2009).

Para resolver estes conflitos é necessário determinar uma estratégia de execução para as *Rules* envolvidas, visando alcançar o fluxo de execução pretendido. Idealmente este processo deve acontecer de forma automatizada, evitando todos os inconvenientes relacionados ao conflito entre *Rules*. A organização da execução de *Rules* aprovadas segundo alguma estratégia pré-estabelecida é uma das estratégias possíveis. (BANASZEWSKI, 2009).

As estratégias de resolução de conflitos dependem do ambiente de execução dos programas. Em ambientes monoprocessados, a resolução de conflitos deve organizar a execução das *Rules*, de forma que apenas uma *Rule* seja executada por vez. Em ambientes multiprocessados, a resolução de conflitos procura evitar o acesso concorrente a um determinado recurso a fim de manter a consistência da execução do programa (BANASZEWSKI, 2009).

Em ambientes monoprocessados pode-se adotar o modelo centralizado de resolução de conflitos. Neste modelo é empregada uma entidade concentradora composta por uma estrutura de dados linear (e.g. pilha, fila ou lista), que armazena uma lista de *Rules* a serem

executadas. Nesta estrutura são armazenadas referências para as *Rules* aprovadas (BANASZEWSKI, 2009). A Figura 11 apresenta esta estrutura, que recebe as *Rules* na ordem em que elas são aprovadas, podendo reorganizá-las de acordo com a estratégia adotada (BANASZEWSKI, 2009).

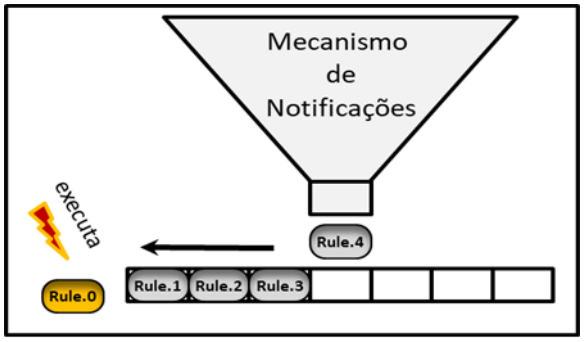

Figura 11 - Modelo centralizado de resolução de conflitos.
Fonte: adaptado de BANASZEWSKI, 2009, p. 109

A solução do PON para ambientes monoprocessados permite adotar algumas estratégias predefinidas que são: *BREADTH* que se baseia no escalonamento *FIFO* (*First-In First-Out*); *DEPTH* que se baseia no escalonamento *FILO* (*First-In Last-Out*); e PRIORITY que organiza as *Rules* de acordo com as prioridades definidas nas mesmas (BANASZEWSKI, 2009).

Ainda em ambientes monoprocessados o modelo descentralizado de resolução de conflitos pode ser adotado. Neste modelo não há o conceito de conjunto de conflito. Assim, não há a ordenação das execuções das *Rules* como constatado no modelo centralizado. Como não se utiliza um conjunto de conflito, este modelo é limitado quanto as estratégias suportadas (BANASZEWSKI, 2009). Visando simplificar o processo de resolução de conflitos, esta estratégia trabalha com o imediatismo da execução das *Rules*. Neste modelo, uma *Rule* é executada no mesmo instante em que é aprovada (BANASZEWSKI, 2009). A Figura 12 apresenta esta estratégia.

Uma vantagem da execução imediata é que a aprovação desnecessária de *Rules* não ocorre, porque toda vez que uma *Rule* é aprovada ela é imediatamente executada. Esta aprovação desnecessária de *Rules* ocorre quando uma *Rule* é aprovada e enquanto está aguardado para ser executada é desaprovada por mudanças nas condições (BANASZEWSKI, 2009). O modelo descentralizado é adotado como padrão (*default)* na solução do PON para ambientes monoprocessados. Este tipo de conflito se não tratado adequadamente pode provocar instabilidades indesejadas na aplicação.

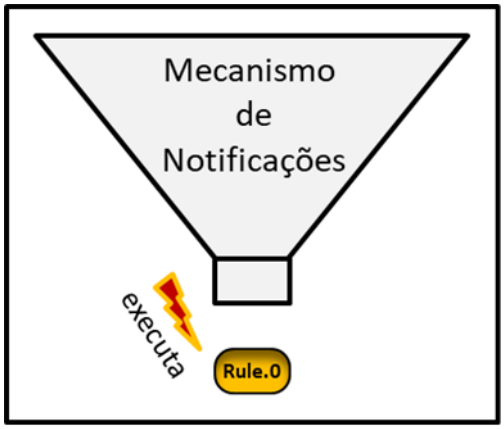

Figura 12 - Modelo descentralizado de resolução de conflitos.
Fonte: adaptado de BANASZEWSKI, 2009, p. 109

2.5.6 PON comparativamente a outros paradigmas de programação

Com o objetivo de analisar o PON sob o viés de paradigma de programação é necessário compará-lo a outros paradigmas. O PON, além de inovar no tocante ao processo de inferência lógico-causal utiliza e melhora alguns conceitos e técnicas presentes no Paradigma Imperativo (PI) e outras presentes no Paradigma Declarativo (PD). Portanto, estes dois paradigmas, nomeadamente PI e PD, serão usados como comparação.

Resumidamente, no imperativo o código é escrito de forma a definir sequencialmente as atividades a serem realizadas pelo dispositivo computacional, inclusive aquelas de decisão ou inferência lógico-causal (GABBRIELLI; MARTINI, 2010). No paradigma declarativo, por sua vez, o código é declarado em nível mais alto que no imperativo, sendo a sequência de execução ditada por uma máquina de inferência explícita. Outrossim, tanto as abordagens imperativas quando as declarativas apresentam um conjunto de imperfeições (GABBRIELLI; MARTINI, 2010).

Estes paradigmas são movidos por inferência monolítica implícita (no caso de PI) ou explícita (no caso de PD), que de alguma forma realiza pesquisas sobre elementos passivos em uma base de fatos (por exemplo, variáveis, objetos, vetores etc.) a fim de testar expressões lógico/causais (por exemplo, a instrução *if - then* ou regras semelhantes). Como já foi mencionado, isto frequentemente resulta, tanto no PI quanto no PD, em acoplamento de código e redundâncias, o que implica em dificuldades na distribuição do processamento e sobrecargas, como detalhado em (FORGY, 1982; SIMÃO; STADZISZ, 2009).

Com uma visão diferente e inovadora no tocante à inferência, o PON aproveita a forma de expressão dos SBR do PD, representando o conhecimento na forma de fatos e regras. O PON também aproveita a flexibilidade de programação e alguns recursos de abstração apresentados pela Orientação a Objetos do PI, tais como classes e objetos (BANASZEWSKI, 2009; LINHARES, 2015).

Particularmente, o PON traz a Orientação a Objetos e a Orientação a Eventos em uma configuração muito específica. Nesta, cada atributo (i.e., 'variável') de objeto ou instância ('módulo') é capaz de notificar mudanças nos seus estados (i.e., uma ocorrência de evento) a um grupo específico de interessados (nomeadamente 'regras'). Ainda, cada método de objetos (i.e., 'função') pode ser executado a partir de uma instigação externa oriunda de evento procedente (e.g. decorrência de uma relação lógico-causal de uma regra aprovada). Como pode ser observado na Figura 13, o PON se encontra entre o Paradigma Orientado a Objetos e os Sistemas Baseados em Regras (POO e SBR, respectivamente), transpassado por uma profunda e exacerbada orientação a eventos em uma configuração muito peculiar, tão peculiar que é denominado orientação a eventos.

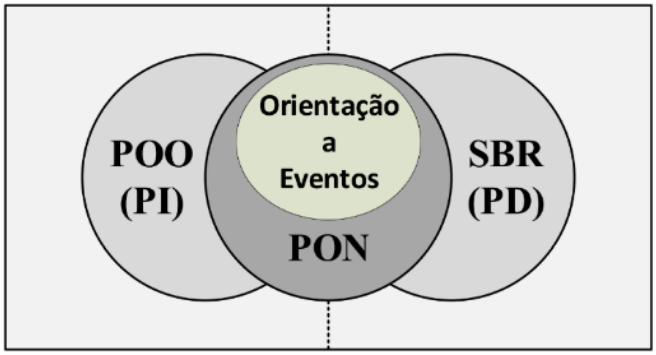

Figura 13 - Relação entre PON, PI e PD.
Fonte: adaptado de BANASZEWSKI, 2009, p. 90

Apesar de estar fundamentado em princípios dos dois paradigmas, o PON possui características que justificam sua classificação como um paradigma em si, justamente no âmago da chamada orientação a notificações. O PON impõe uma nova forma de estruturação e de elaboração da lógica de programas, impondo a orientação a notificações na expressão da dinâmica do programa e da sua lógica de causa e efeito, no nível mais básico de abstração possível (LINHARES, 2015). Diferentemente do PI, onde a lógica do programa está fortemente ligada a sequência de execução, no PON, a dinâmica é expressa através de notificações pontuais que potencialmente podem ser executadas em paralelo dado o desacoplamento implícito que

proporcionam (LINHARES, 2015). O PON também se diferencia dos SBRs onde a sequência de execução é dependente do mecanismo de inferência utilizado, sendo este mecanismo orientado a buscas em estruturas de dados como árvores e afins.

O PON tem o potencial de melhorar o funcionamento dos programas com relação a desempenho e capacidade de distribuição, exatamente onde o PI e PD são deficientes, como já foi mencionado. As características que tornam o PON tão interessante são: seu processo de inferência por notificações; e seu modelo de representação de relações causais, baseado em entidades coesas, reativas e desacopladas (LINHARES, 2015).

As alterações ocorridas nos estados são imediatamente passadas as partes pertinentes ao contexto do programa, isso devido ao processo de inferência por notificações. É importante salientar que a propagação ocorre somente quando os estados são alterados, ou seja, o desempenho global do sistema é melhorado evitando-se as buscas desnecessárias sobre elementos passivos (LINHARES, 2015).

Estruturalmente o PON é composto por elementos reativos, interconectados de tal forma que cada elemento notifica pontualmente apenas os elementos que estão interessados na alteração de seu estado. Esta característica possibilita um ganho teórico em termos de aproveitamento do potencial de computação, o que tende a melhorar o desempenho relativo de aplicações PON quando comparadas a aplicações similares em PI ou PD (BANASZEWSKI, 2009; LINHARES, 2015).

O desacoplamento entre os elementos do PON teoricamente favorece sua distribuição em diversos nós computacionais sem afetar a coesão do programa. Para viabilizar esta distribuição basta apenas que seja fisicamente possível propagar as notificações entre os nós computacionais envolvidos. Da mesma forma é possível realizar o processamento paralelo destas notificações, desde que mecanismos de resolução de conflitos sejam estabelecidos.

2.5.7 As plataformas do PON

O modelo estrutural do PON não tem relação implícita com nenhuma plataforma específica. Assim sendo, várias implementações vêm sendo realizadas com o PON em diferentes plataformas. As seções a seguir apresentam as principais implementações.

2.5.7.1 *Framework* PON C++

A primeira implementação do PON foi proposta inicialmente por Jean M. Simão através de um *framework* que implementa o metamodelo de Controle Orientado a Notificações (CON) na ferramenta ANALYTICE II (SIMÃO, 2005). Este *framework* serviu de base para o *framework* PON C++, proposto também por Jean M. Simão, chamado de *framework* prototipal ou Biblioteca Referencial de Programação Orientada a Notificações Simão *et al*. (2012c). Subsequentemente, uma versão mais efetiva do *framework* foi refeita por Banaszewski (2009), sendo esse chamado de *framework* PON (C++) Original ou mais corretamente *Framework* PON (C++) 1.0.

Uma nova versão do *framework* com uma série de otimização foi desenvolvida por Valença (2012) e com o auxílio de Ronszcka (2012), tendo sido chamada de *Framework* PON (C++) otimizado, ou mais corretamente *Framework* PON (C++) 2.0. Esta nova versão do *framework* PON C++ apresentou significativos ganhos de performance, adotando estruturas de dados especialmente adaptadas para o PON, além de estruturação orientada a padrões de projeto. Ademais, uma interface gráfica chamada *Wizard* foi desenvolvida para a criação em alto nível de *FBEs*, *Rules* e afins.

Muitos experimentos vêm sendo realizados com o auxílio destas três implementações na forma de *framework* C++, sendo que o os resultados têm sido documentados e publicados (BANASZEWSKI, 2009; BATISTA *et al*., 2011; RONSZCKA *et al*., 2011; LINHARES *et al*., 2011; SIMÃO *et al*., 2012a; RONSZCKA, 2012; VALENÇA, 2012; SANTOS, 2017).

Ainda outros pesquisadores têm trabalhando em outras versões do *framework* PON. Algumas destas versões foram derivações para lidar com diferentes ambientes, como aplicações *multi-thread* (derivação do *framework* Prototipal) (WEBER *et al.*, 2010), plataformas multi-core (derivação do *Framework* 1.0) (BELMONTE; SIMÃO; STADZISZ, 2012; BELMONTE, 2012; BELMONTE, 2016), sistemas *fuzzy* (derivado do *Framework* 2.0) (MELO; SIMÃO; FABRO, 2014; MELO, 2016) e redes neurais (derivado do *framework* 2.0) (SCHÜTZ, 2015). Além disso, versões do *framework* foram reconstruídas em outras linguagens de programação, como Java e C# (HENZEN, 2015).

Cada um destes *Frameworks* ou Arquétipos PON é composto por uma estrutura orientada a objetos definida por um conjunto de classes em linguagem orientada a objetos. Em cada *framework* PON, cada elemento do modelo de notificações do PON possui elementos correspondentes na forma de classes e objetos.

As versões mais avançadas destes *Frameworks* PON foram concebidas para oferecer uma interface de programação (API) de relativamente fácil utilização para a implementação de aplicações que seguem o modelo do PON. O *Framework* (C++) 2.0 é um exemplo disso, por definir as abstrações necessárias para compor os *FBEs* e suas respectivas *Rules* (LINHARES 2015; VALENÇA, 2012). A Figura 14 apresenta um pequeno exemplo de código escrito para o *Framework* PON (C++) 2.0.

```
1   fbe Archer
2       attributes
3           boolean atHasFired false
4           integer atCount 0
5       end_attributes
6       methods
7           method mtFire1(atHasFired = true)
8           method mtFire2(atCount = atCount + 1)
9           method mtFire3(atCount = atCount + atCount)
10          method mtFire4() begin_method //código
11  específico em C/C++ end_method
12      end_methods
13  end_fbe
```

Figura 14 - Exemplo de aplicação em LingPON para *Framework* PON 2.0.
Fonte: FERREIRA, 2015, p. 79

Entretanto, cada uma destas implementações, apesar de funcional não é profundamente fiel ao PON, pois o mecanismo de notificações acontece de forma sequencial, via estruturas de dados clássicas. Portanto e em suma, tais implementações não apresentam paralelismo efetivo ainda. Mesmo nas implementações em multi-thread para multi-core (BELMONTE; SIMÃO; STADZISZ, 2012; BELMONTE, 2012; BELMONTE, 2016), bem como os protótipos com distribuição, ainda assim tem processamento sequencializado em cada núcleo ou unidade de processamento.

Contudo, elas têm servido como uma plataforma de validação da factibilidade do PON e validação de certas propriedades dele, como o evitar de redundâncias estruturais e temporais, a possibilidade de baixar o tempo de processamento, o desacoplamento entre entidades, algum nível de paralelismo/distribuição e mesmo alguma facilidade de programação. Ademais elas têm permitido a prototipação de considerável número de aplicações PON, usando uma plataforma de computação convencional e programação em PI (SANTOS, 2017).

2.5.7.2 Linguagem e compilador PON

O PON como paradigma de programação vinha carecendo de melhores facilidades de programação, o que inclusive é um de seus propósitos. A utilização de *frameworks* apesar de facilitar a implementação das aplicações PON e permitir uma assaz clara visualização de seus componentes no código, não tem o mesmo apelo e organização que uma linguagem específica permitiria. Assim, com o objetivo de desenvolver uma linguagem de programação específica para o PON e um compilador, capaz de traduzir tal linguagem em um código-alvo, Cleverson A. Ferreira (2015) desenvolveu seu trabalho.

Com uma linguagem especialmente desenvolvida para o PON, chamada LingPON, é possível escrever aplicações em conformidade com este paradigma de forma fácil e rápida, apesar de sua gramática ainda não estar tão completa quanto poderia (FERREIRA, 2015). A Figura 14 da seção anterior apresenta um pequeno exemplo de aplicação escrito em LingPON, nele é possível observar as palavras-chave que representam os elementos do PON, como "*attributes*", "*methods*" entre outros.

Para que se possa usufruir dos benefícios da linguagem LingPON, é necessário um compilador que a traduza em algum código-alvo, que possa ser executado em alguma plataforma de computação. Este compilador foi chamado de compilador PON e é capaz de traduzir o código escrito em LingPON para três linguagens, dentre as quais pode-se destacar a linguagem C, a linguagem C++, código específico para o *Framework* PON (C++) 2.0 entre outros (FERREIRA, 2015).

Buscando evoluir a LingPON, de forma a facilitar o desenvolvimento de aplicações PON Leonardo Araújo Santos (SANTOS, 2017; SANTOS *et al*, 2017) desenvolveu melhorias na linguagem. Por meio do aprimoramento das agregações de entidades no LingPON, a luz do próprio PON, isso levou a redução de redundâncias e consequentemente a redução do número de linhas de programa no código fonte das aplicações feitas neste paradigma. Foram utilizados indicadores de complexidade de código como o número de linhas do programa e a quantidade de *tokens* para mensurar as melhorias de desenvolvimento.

Uma adaptação do compilador, pertinente a este trabalho, foi desenvolvida pelo autor deste trabalho de doutorado[4], onde o código em LingPON é diretamente traduzido para VHDL,

[4] Este trabalho iniciou-se na disciplina "TEC0301 - Tópicos Especiais Em Ec: Paradigma Orientado A Notificações" ministrada pelo professor Jean Marcelo Simão e teve continuidade no âmbito deste trabalho de doutorado.

de forma a ser executado diretamente em *Hardware*. Dado que isso se constitui em parte das contribuições deste trabalho.

2.5.7.3 O PON em ambientes paralelos e distribuídos

Se for levado em consideração a essência do PON, no que diz respeito ao desacoplamento das entidades e ao paralelismo das notificações, é possível observar características que favorecem a distribuição e paralelização das aplicações dado o explícito desacoplamento entre as entidades proporcionada pela orientação a notificações. Vários estudos vêm sendo realizados com o objetivo de aproveitas estas características, tanto em *software* como em *hardware*.

Um ambiente *multi-thread* para execução de aplicações foi proposto por Weber *et al*. (2010), nele cada entidade (*Rule, Condition, Premise* etc.) é executada em uma *thread* separada. Na mesma linha de raciocínio, Belmonte (2012) realizou melhorias no *framework* PON 1.0 para que suas *Rules* sejam executadas em *Threads* separadas (BELMONTE; SIMÃO; STADZISZ, 2012; BELMONTE, 2012; BELMONTE, 2016)

Um estudo mais aprofundado, relativo ao balanceamento de carga em sistemas multiprocessados foi realizado por Belmonte, Simão e Stadzisz (2012). O objetivo deste estudo foi melhorar o desempenho, através da melhoria da paralelização de execução dos elementos notificantes constituintes do PON (BELMONTE, 2012). Alguns ensaios mais recentes também começaram a demostrar a facilidade do PON em sistemas distribuídos (BELMONTE, 2016). Entretanto, tanto nos atuais trabalhos de paralelismo quanto de distribuição do PON em *software*, não há paralelismo/distribuição efetiva dado que cada núcleo ou ponto de execução é sequencial, a luz da tradicional arquitetura von Neumann.

Uma forma de executar com maior fidelidade o PON é por meio de um dispositivo de lógica reconfigurável (FPGA). Alguns trabalhos buscaram viabilizar a implementação de parte (somente cadeia de notificações) ou de toda uma aplicação PON diretamente em *hardware*. Dentre estes é possível destacar Witt *et al*. (2011), Jasinski (2012), Peters (2012) e Linhares (2015). Como as implementações do PON em *hardware* tem grande relação com o tema deste trabalho serão dedicadas seções específicas a cada uma delas.

2.5.8 Implementações do PON em *hardware* digital (PON-HD)

As características do PON permitem tirar proveito da paralelização e/ou distribuição de forma mais simples e eficiente do que o Paradigma Imperativo e o Paradigma Declarativo. Entretanto, a dinâmica de execução do PON, baseada em notificações, não é executada com eficiência no *hardware* dos sistemas computacionais atuais, devido principalmente ao seu modelo de execução sequencial (LINHARES 2015). Nas próximas subseções serão apresentadas implementações do PON em *hardware* digital.

2.5.8.1 O PON-HD Prototipal

As características de desacoplamento das entidades do PON permitem que sua execução aconteça de forma paralela, e desta forma, sua implementação em *hardware* traz potenciais ganhos de desempenho em relação a implementações convencionais realizadas puramente em *software* (WITT *et al*., 2011). Witt *et al.* (2011) propuseram que os elementos que compõe o PON podem ser modelados em blocos de lógica reconfigurável. Dado a pequena complexidade do funcionamento de cada uma das entidades do PON, estas podem ser modeladas em *hardware* como blocos lógicos (*AND, OR*, *NOT*) e blocos sequenciais (*latches* e *flip-flops*) (WITT *et al.*, 2011). Esta implementação veio posteriormente a ser chamada de PON-HD Prototipal.

Para melhor compreensão do funcionamento do PON em *hardware*, uma descrição da implementação de cada um dos seus componentes é necessária.

Uma *FBE*, na estrutura do PON, serve para agregar *Attributes* e *Methods* e desta forma possui um caráter organizacional, não necessitando de uma implementação em *hardware* (WITT *et al.*, 2011). Um *Method*, por sua vez, tem a função de alterar o valor dos *Attributes*. Assim, nesta implementação do PON em *hardware*, o *Method* é composto por um bloco de circuito digital sincronizado com o *clock*. Este bloco é concebido de forma a trabalhar com outros *Methods* em um barramento (WITT *et al.*, 2011). A Figura 15 apresenta o diagrama do *Method*.

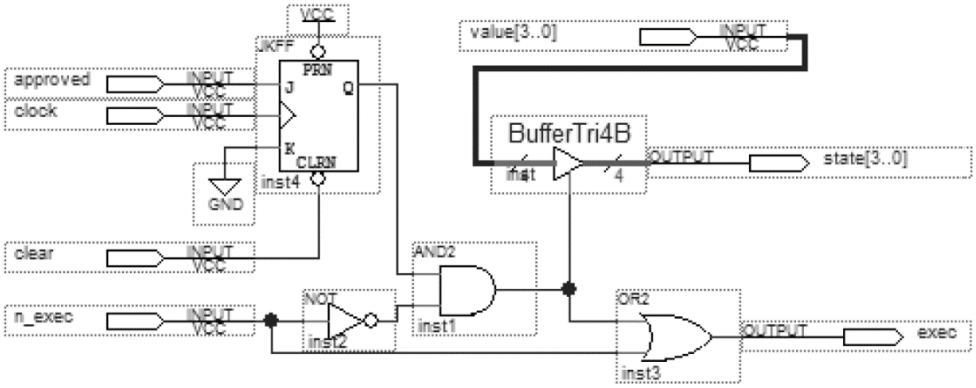

Figura 15 - *Method* no PON-HD Prototipal.

Fonte: Witt *et al.* (2011).

Os *Attributes*, por sua vez, têm a responsabilidade de armazenar os dados, e assim devem ser compostos por *flip-flops* do tipo D. Também é de responsabilidade dos *Attributes* a notificação das *Premises*, quando da alteração de seu conteúdo. A sincronização deste circuito é realizada através do *clock*, permitindo assim que a transmissão dos dados entre os *Methods* e os *Attributes* aconteça (WITT *et al.*, 2011). A Figura 16 apresenta o diagrama do *Attribute*, nesta implementação.

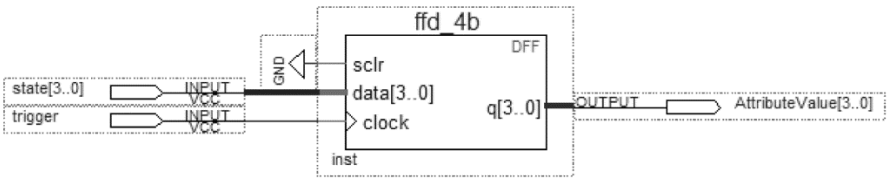

Figura 16 - *Attribute* no PON-HD Prototipal.

Fonte: Witt *et al.* (2011).

Quando da alteração dos *Attributes*, as *Premises* são notificadas e devem realizar seus cálculos lógicos. Em *hardware*, as *Premises* são compostas por blocos comparadores assíncronos. Estes blocos possuem duas entradas que devem ser conectadas a dois *Attributes*, ou a um *Attribute* e uma constante de forma a realizar suas comparações. Como saída, este bloco fornece um sinal, que assume os valores verdadeiro ou falso (WITT *et al.*, 2011). A Figura 17 apresenta o diagrama da *Premise*, nesta implementação.

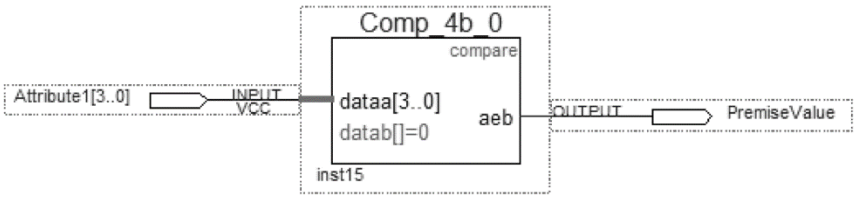

Figura 17 - *Premise* no PON-HD Prototipal.

Fonte: Witt *et al.* (2011).

Os sinais fornecidos pelas *Premises* são então conectados as *Conditions*. Como as *Conditions* apenas realizam operações lógicas sobre os resultados das *Premises* (*NOT, AND* e *OR*), não é necessária a construção de um bloco específico para este fim, basta utilizar portas lógicas normais (WITT *et al.*, 2011).

A função de uma *Rule* é analisar o estado lógico das *Conditions* que a compõe. Se o resultado for verdadeiro, uma *Action* deve ser ativada. No *hardware*, bastaria apenas conectar a saída da *Condition* à entrada da *Action*. Porém, para evitar que a *Action* seja executada repetidamente de forma incorreta, o sinal de ativação deve permanecer ativo por apenas um ciclo de *clock*. Assim, um circuito composto por dois *flip-flops* tipo D e sincronizado pelo *clock* foi concebido para este fim (WITT *et al.*, 2011). Este circuito pode ser visualizado na Figura 18.

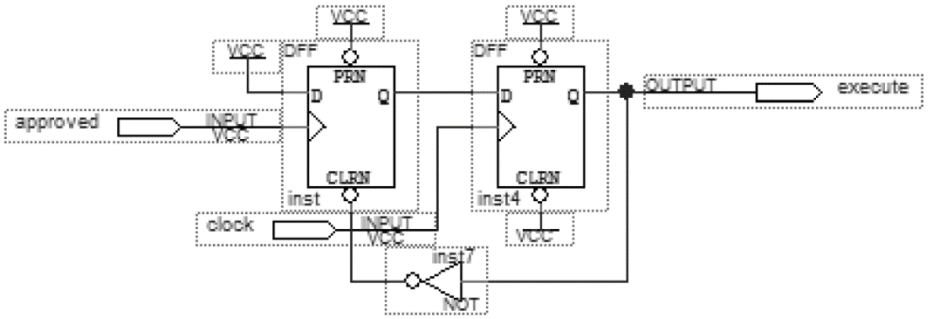

Figura 18 - Circuito de ativação da *Action*.

Fonte: Witt *et al.* (2011).

O modelo inicial do PON-HD Prototipal não prevê mecanismos de determinismo temporal e de resolução de conflitos para *Rules* que rodam em paralelo, desta forma, uma *Action* é responsável apenas por transferir o sinal de ativação da *Rule* para a *Instigation*, porém, mecanismos de resolução de conflitos podem potencialmente ser implementados neste bloco (WITT *et al.*, 2011). A função da *Instigation* no *hardware*, por sua vez, é transferir o sinal de uma ou mais *Actions* para um *Method*, uma porta OR é suficiente para desempenhar esta função (WITT *et al.*, 2011).

Esta implementação do PON em *hardware* valida a cadeia de notificações em termos funcionais, não sendo necessariamente otimizada em termos de desempenho. (WITT *et al.*, 2011). Por implementar de forma mais fiel a essência do PON e apresentar características bastante promissoras esta implementação do PON motivou um pedido de patente (SIMÃO *et al.*, 2012b).

É importante, para justificar a realização deste trabalho de pesquisa, enfatizar que apesar de se mostrar funcional, esta implementação apresenta deficiências com relação a performance, a implementação de um mecanismo de determinismo de evolução e de técnicas de sincronização de regras executadas em paralelo (resolução de conflitos) (WITT *et al.*, 2011). A dificuldade de implementação das aplicações também é um dos fatores negativos desta implementação, pois é necessário que o desenvolvedor conecte manualmente todos os blocos de forma a montar a cadeia de notificações e criar o comportamento necessário para a aplicação.

O potencial apresentado pela execução do PON em *hardware* digital pode apenas ser vislumbrado pela implementação do PON-HD Prototipal. Uma implementação mais robusta e completa se fez necessária para que seja possível continuar as pesquisas da viabilidade do PON como ferramenta de desenvolvimento de circuitos em lógica reconfigurável.

Na sequência, Jasinski (2012) propôs um *framework* para a geração de *hardware* em VHDL a partir de modelos em PON. Neste *Framework* a aplicação PON é descrita em um arquivo no formato YAML que posteriormente é convertido para arquivos VHDL. A aplicação final é resultado da compilação do arquivo VHDL gerado pelo *Framework* junto com outros arquivos em VHDL que contém a descrição dos componentes do PON em *Hardware*. O resultado é uma aplicação PON que roda em *hardware* digital (FPGA) (JASINSKI, 2012). Este *framework* foi desenvolvido como trabalho final para a disciplina de “LRH0087 - Lógica Reconfigurável Por *Hardware*”, ministrada pelo professor Carlos Raimundo Erig Lima em 2012. O relatório original deste trabalho pode ser encontrado na íntegra no Anexo A. Apesar de ter se mostrado funcional em algumas aplicações de teste, este projeto continua sofrendo com as mesmas limitações do PON-HD Prototipal, principalmente com relação a ausência de um mecanismo de resolução de conflitos e a performance.

2.5.8.2 Coprocessador PON (COPON)

Peters (2012) observou que como uma aplicação PON é composta de uma cadeia de pequenas entidades computacionais, comunicando-se somente quando necessário, sua

implementação direta em *hardware* é muito facilitada. Para aproveitar esta característica foi desenvolvida uma arquitetura onde o PON é implementado parte em *hardware* e parte em *software*. A parte implementada em *hardware* é um co-processador que é responsável pelas avaliações lógico-causais. A Figura 19 mostra como este co-processador foi implementado e integrado ao processador principal.

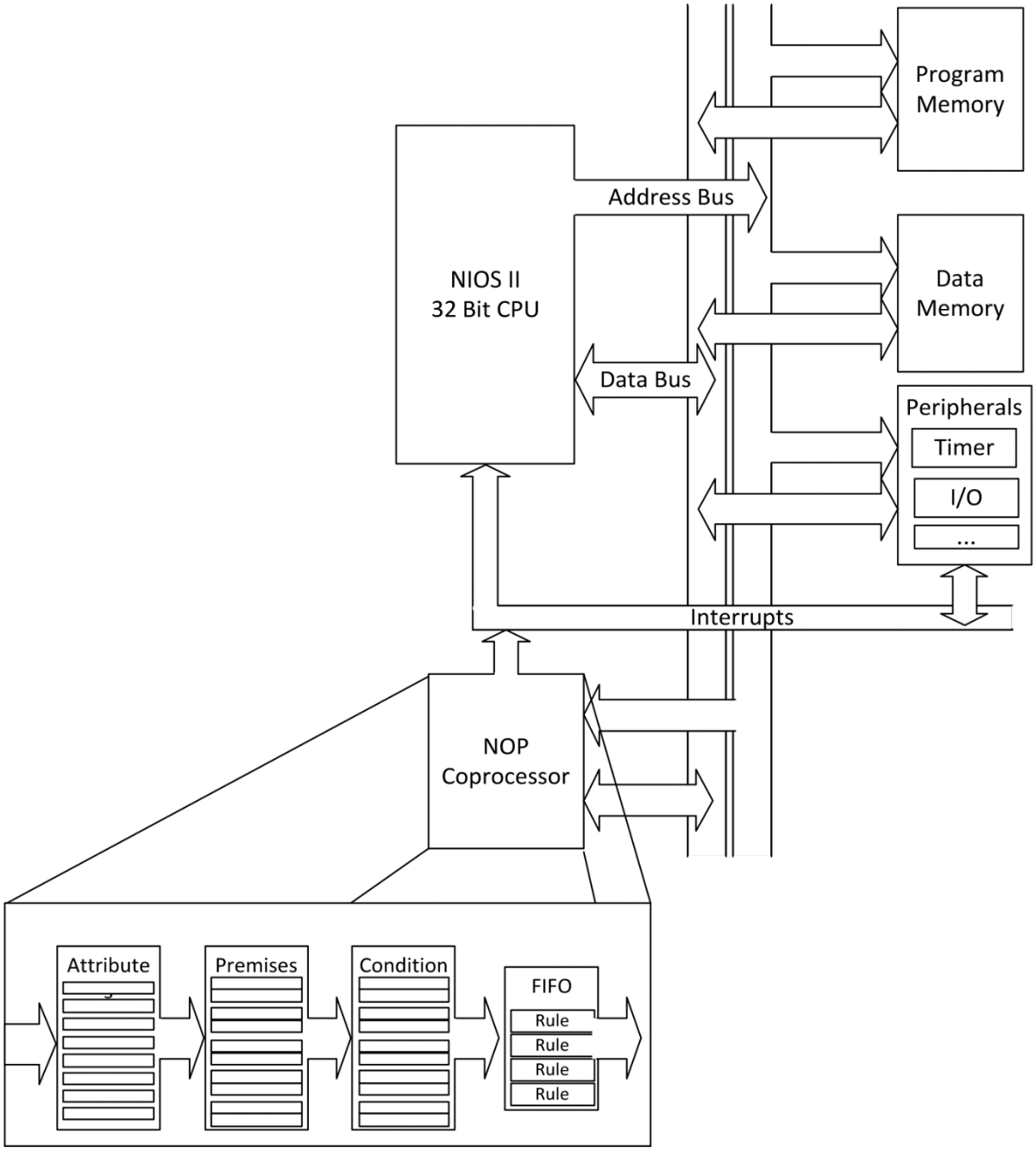

Figura 19 - Implementação do co-processador PON.

Fonte: Peters (2012).

A parte em *software* é um programa que lida com o processamento factual e execucional e é processado em um núcleo von Neumann comum. A Figura 20 apresenta em um fundo mais escuro os componentes do PON que foram implementados na forma de *hardware*, através deste co-processador.

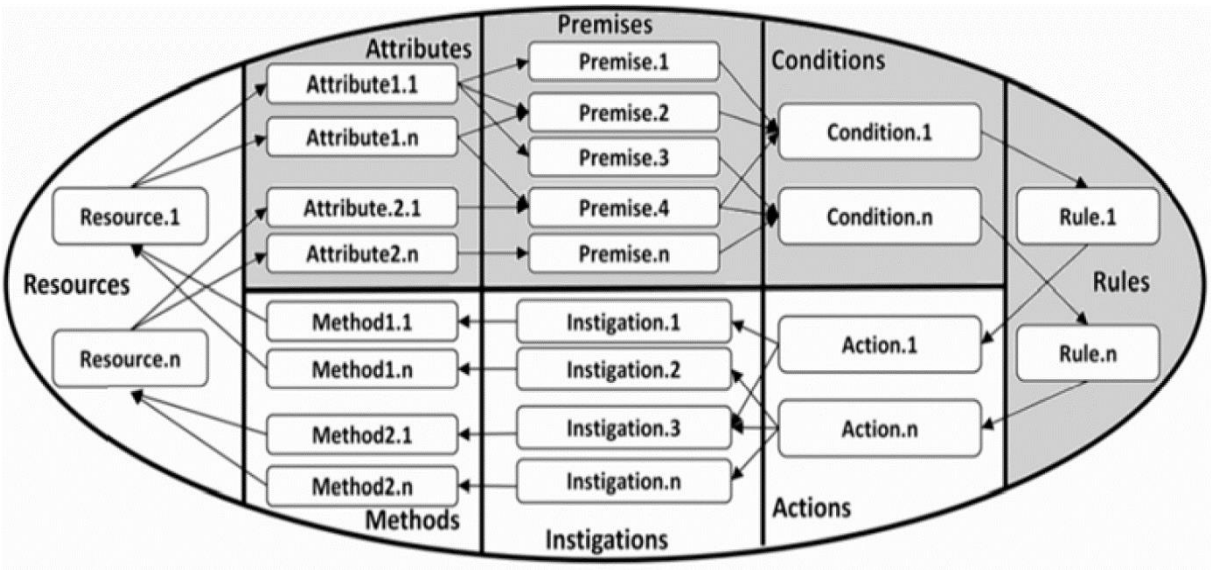

Figura 20 - Elementos do PON implementados no co-processador.
Fonte: Peters (2012).

Uma implementação desta arquitetura foi realizada em lógica reconfigurável (i. e. FPGA) utilizando linguagem VHDL. Os experimentos demostraram um decréscimo de até 96% no número de ciclos de *clock* utilizados por uma aplicação se comparada à implementação da mesma aplicação puramente em *software*, utilizando o *Framework* PON (C++) 1.0 (PETERS *et al*., 2012).

2.5.8.3 Arquitetura de computação PON (ARQPON)

Buscando uma forma mais eficiente de executar as aplicações PON, Linhares (2015) desenvolveu uma arquitetura de computação, denominada ARQPON, que é própria para a execução de *software* desenvolvido segundo o PON. Com base nesta arquitetura, um microprocessador PON completo foi implementado em uma FPGA. Esta implementação foi capaz de buscar em memória e executar aplicações PON desenvolvidas como um *software* de baixo nível, de forma semelhante a um processador von Neumann comum, porém com uma dinâmica de execução (propagação de notificações) mais próxima à dinâmica definida pelo

modelo teórico do PON (LINHARES 2015; LINHARES; SIMAO; STADZISZ, 2015). Esta nova arquitetura de computador orientada a notificações foi chamada de NOCA (*Notification Oriented Computer Architecture*).

Observando o potencial da arquitetura NOCA e a dificuldade de realizar experimentos nesta arquitetura (é necessária uma FPGA muito grande e potente), Pordeus (2017) propôs em sua dissertação de mestrado o desenvolvimento de um simulador. Este simulador recebeu o nome de NOCASim e é constituído da implementação da NOCA na forma de um *software* de computador, com o objetivo de simular o funcionamento desta arquitetura sem a dependência do *hardware* digital (PORDEUS, 2017).

2.5.9 Desenvolvimento Orientado a Notificações (DON)

Devido às características do PON, as abordagens tradicionais de modelagem, tanto do PD (SBR) quanto do PI, não são apropriadas. Para contornar esta deficiência, Wiecheteck (2011) propôs um método para projeto de *software* PON, denominado DON (Desenvolvimento Orientado a Notificações).

O DON é baseado no processo iterativo do RUP (*Rational Unified Process*), adaptando partes dele ao PON. Através do DON é possível aplicar um perfil UML que expressa os conceitos PON viabilizando sua aplicação na etapa de modelagem em processo de engenharia de *software* (MENDONÇA, 2015; MENDONÇA *et al.*, 2015; WIECHETECK, 2011). A Figura 21 apresenta as etapas DON.

O DON, por aplicar uma abordagem convencional de modelagem orientada a objetos, não favorece propriamente a modelagem de *software* PON. Mendonça (2015) estabeleceu uma Metodologia de Projeto de *Software* Orientado a Notificações (NOM). Esta metodologia é focada a orientação a regras, fatos e notificações. Nesta metodologia os elementos do PON são levados em consideração desde os primeiros níveis de modelagem (MENDONÇA, 2015).

Ainda relacionado ao desenvolvimento de *software*, Kossoski (2015) propôs em sua dissertação de mestrado um método de teste para projetos de *software* que empregam o PON no seu desenvolvimento. Este método desenvolvido para ser aplicado nas fases de teste unitário e teste de integração. Os resultados desta pesquisa mostram que o método contribuiu para facilitar os testes de programas desenvolvidos segundo o PON (KOSSOSKI, 2015; KOSSOSKI; SIMÃO; STADZISZ, 2014).

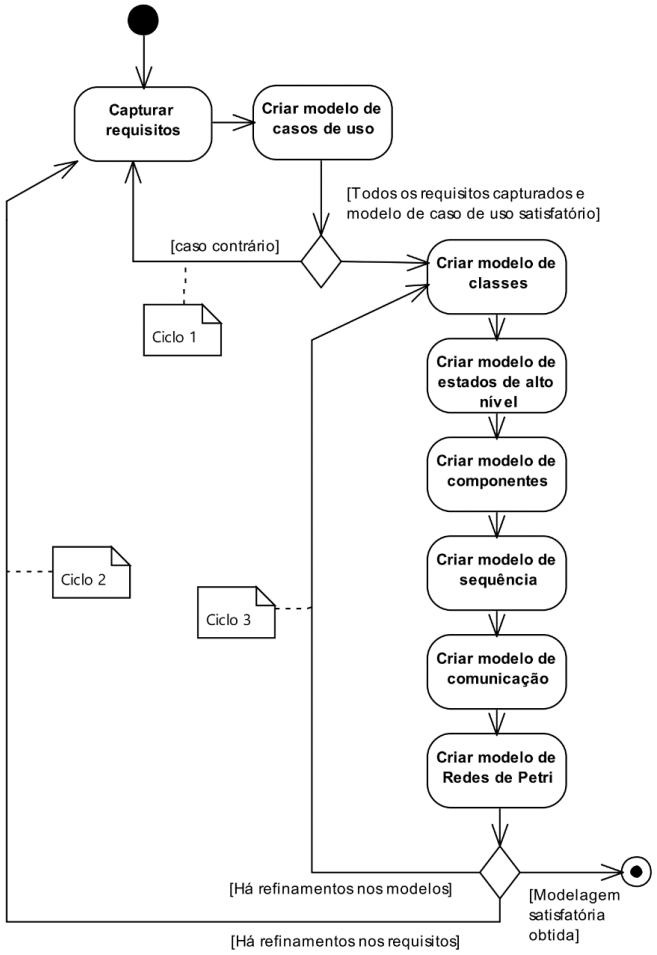

Figura 21 - Etapas do DON.

Fonte: Mendonça (2015).

2.6 Considerações sobre o capítulo

Neste capítulo, foram apresentados vários conceitos teóricos que servem de fundamentação a esta pesquisa de doutorado. Inicialmente foram apresentados alguns fundamentos sobre os dispositivos de lógica reconfigurável e sua programação, com o objetivo de justificar a importância destes componentes na atualidade.

Na sequência, a discussão foi sobre as ferramentas de síntese em alto nível e sua importância na programação de dispositivos de lógica reconfigurável. Esta seção tratou do esforço despendido na busca de novas ferramentas de síntese, bem como sua classificação e avaliação.

Observando as informações apresentadas nesta seção, nota-se que existe um esforço em desenvolver ferramentas de síntese em alto nível que consigam aproveitar os algoritmos já existentes, escritos em linguagens tradicionais como C e C++. Esta estratégia é por um lado

vantajosa, pois facilita o reaproveitamento de código, mas é por outro lado desvantajosa, pois traz para estas ferramentas muitos dos problemas inerentes a estas linguagens e aos paradigmas que elas pertencem. Características como dificuldade de paralelização do código, por exemplo, implicam em comprometimento de performance dos circuitos gerados por estas ferramentas se comparado a circuitos sintetizados diretamente em linguagens de síntese de *hardware* como VHDL ou Verilog. Observa-se, neste contexto, que a problemática da síntese em alto nível é permitir o desenvolvimento de *hardware* em um nível mais alto de abstração, sem o comprometimento da performance.

A seção sobre paradigmas de desenvolvimento apresenta um panorama geral sobre os principais paradigmas de programação, enfatizando principalmente o Paradigma Imperativo e o Paradigma Declarativo, bem como suas principais deficiências. Esta seção serve principalmente de fundamentação para a seção seguinte que fala sobre o Paradigma Orientado a Notificações.

O Paradigma Orientado a Notificações (PON) é discutido na seção seguinte. Iniciando pelas suas origens e comparando-o a outros paradigmas. Esta seção tratou ainda das vantagens que o PON apresenta em relação a outros paradigmas de desenvolvimento. A estrutura e o processo de inferência do PON foram então discutidos, com o objetivo de melhor esclarecer o funcionamento deste paradigma. Com o objetivo de mostrar o que já existe com relação ao PON, foram então apresentadas as várias materializações deste paradigma, nas mais variadas plataformas.

A seção seguinte apresenta o Paradigma Orientado a Notificações em *Hardware* Digital (PON-HD). Esta seção tem grande importância para este trabalho de doutorado, pois apresenta o PON-HD Prototipal, que serviu de base para o desenvolvimento do PON-HD 1.0 que será apresentado no decorrer dos próximos capítulos e faz parte das contribuições desta pesquisa de doutorado.

As limitações apresentadas pelo PON-HD Prototipal motivaram a proposição deste tema de pesquisa por parte dos orientadores da presente tese de doutorado, bem como, o desenvolvimento de uma nova implementação do PON em *hardware* digital, que contemple principalmente a resolução de conflitos implícita e uma performance melhorada. As diferenças entre a implementação prototipal dos componentes do PON em *hardware* e a implementação destes mesmos componentes realizada no âmbito deste trabalho de pesquisa serão mais profundamente discutidas junto a descrição do *framework* PON-HD 1.0.

Ainda com o objetivo de aproveitar melhor o potencial do PON para a síntese de *hardware* digital, a integração do LingPON ao PON-HD permite construir um ferramental de

desenvolvimento em alto nível. É com estas características que o ferramental proposto neste trabalho de doutorado pretende se encaixar entre as ferramentas de desenvolvimento de *hardware* digital em alto nível.

Por último e não menos importante é apresentado Desenvolvimento Orientado a Notificações (DON). Através do método DON as características do PON podem ser empregadas no desenvolvimento de *software*. Ainda nesta seção, foi apresentada a Metodologia de Projeto de *Software* Orientado a Notificações (NOM) que formaliza o processo de desenvolvimento de aplicações desenvolvidas segundo o PON. Um método de teste para projetos de *software* que empregam o PON no seu desenvolvimento foi então apresentado finalizando a seção.

É importante ressaltar que as técnicas apresentadas nesta seção sobre o desenvolvimento e os testes de aplicações desenvolvidas segundo o PON se aplicam também em parte a aplicações desenvolvidas para *hardware* digital com o PON-HD 1.0, que é o foco deste trabalho de doutorado.

Levando em consideração tudo o que foi apresentado é possível visualizar que este trabalho de pesquisa vem contribuir com o tema através da proposição de um ferramental que permite desenvolver *hardware* em alto nível, através de regras, sem comprometimento da performance dos circuitos gerados.

3 Desenvolvimento do trabalho

Este capítulo detalha as pesquisas realizadas com relação a aplicação do PON em *hardware* no âmbito do presente trabalho. Inicialmente é apresentado o PON-HD 1.0 propriamente dito. Os componentes que fazem parte do ferramental PON-HD 1.0 são então apresentados em detalhes. A utilização do PON no desenvolvimento em alto nível é discutida através da apresentação de um conjunto de ferramentas, desenvolvidas para este fim, as quais constituem a chamada tecnologia LingPON-HD 1.0.

Com o objetivo de utilizar o PON para a síntese de *hardware* digital foi definido um conjunto de diretrizes que o PON-HD 1.0 deve atender, o Apêndice A apresenta estes requisitos.

3.1 O PON em *Hardware* Digital 1.0 – PON-HD 1.0

Para facilitar a compreensão do funcionamento do PON-HD 1.0, é apresentado um pequeno exemplo, iniciando pela descrição de sua lógica de operação.

Resumidamente, a operação do PON-HD 1.0 é:

(a) os valores dos *Attributes* são comparados pelas *Premises*, resultando em um valor lógico (verdadeiro ou falso);

(b) se a operação lógica definida pela *Condition*, sobre as *Premises* que a compõem, apresentar um resultado verdadeiro, os respectivos *Methods* são ativados;

(c) os *Methods* ativados alteram o valor de outros *Attributes*, completando assim o ciclo de execução do PON no PON-HD 1.0.

Como exemplo foi escolhida uma aplicação bastante simples. A Figura 22 apresenta o circuito deste exemplo.

Trata-se de um contador que conta até dez, quando uma entrada de controle está ativa. O diagrama de circuito apresentado na Figura 22 foi obtido com a ferramenta “RTL Viewer” da Altera, e representa o resultado da compilação da aplicação desenvolvida com o PON-HD 1.0.

Cada um dos blocos apresentados na Figura 22 será discutido na próxima seção, onde os elementos do PON-HD 1.0 serão apresentados com mais detalhes.

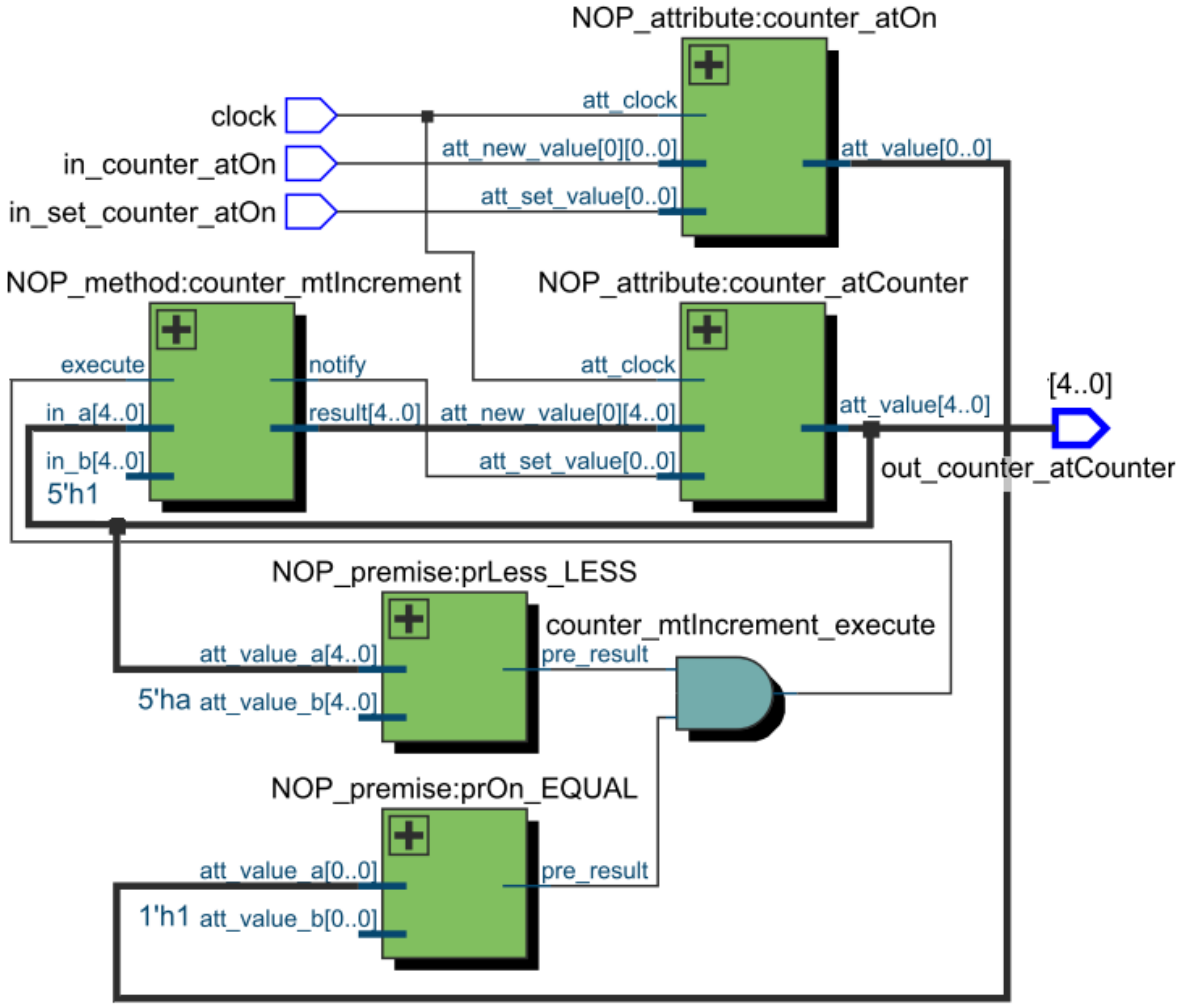

Figura 22 - Diagrama esquemático da implementação em *hardware*.

Fonte: Autoria própria.

Em todo caso, neste exemplo são utilizados números inteiros com sinal de 5 bits para a contagem e valores booleanos de 1 bit para o controle. O circuito do exemplo contém dois *Attributes*, o "counter_atCounter" e o “counter_atOn”. O " counter_atCounter" é responsável por armazenar o valor da contagem, enquanto o " counter_atOn" é responsável por armazenar o estado de ligado ou desligado do contador.

O circuito também contém duas *Premises*. A primeira *Premise*, denominada "prLess", verifica o valor da contagem. A segunda *Premise*, denominada "prOn", verifica se o contador está ativo. Mais precisamente, a *Premise* "prLess" verifica se o *Attribute* " counter_atCounter" é menor que a constante 10, sendo que suas entradas são: o valor do *Attribute* " counter_atCounter" e a constante 10. Por sua vez, a *Premise* "prOn" verifica se o valor do *Attribute* "counter_atOn" é verdadeiro, sendo que as suas entradas são: o valor do *Attribute* "counter_atOn" e a constante 1. O último componente a ser descrito é o *Method*

"counter_mtIncrement", que serve para executar o incremento do contador. Mais precisamente, ele age incrementando o valor do *Attribute* "counter_atCounter", sendo que as suas entradas são: o valor do *Attribute* "counter_atCounter" e a constante 1.

Em tempo, as entradas do circuito são: o sinal de *clock*, a entrada "in_counter_atOn" que habilita a contagem e a entrada "in_set_counter_atOn" que armazena o valor da entrada no *Attribute* "counter_atOn". A saída do circuito é o valor armazenado no *Attribute* "counter_atCount", que é neste caso composto por 5 bits.

Outrossim, o funcionamento do circuito é bastante simples. Enquanto o *Attribute* "counter_atOn" permanecer com o valor 0 o contador permanece parado. Assumindo que o valor inicial do *Attribute* "counter_atCount" é zero, quando um sinal de ativação é armazenado no *Attribute* "counter_atOn", as *Premises* "prLess" e "prOn" tornam-se verdadeiras e ativam o *Method* "counter_mtIncrement". Esse método incrementa o valor do *Attribute* "counter_atCount" a cada ciclo de *clock* até ele atingir o valor 10. Neste ponto, a *Premise* "prLess" torna-se falsa, inibindo, assim o *Method* "counter_mtIncrement" parando a contagem.

A seguir será apresentado um detalhamento de cada um dos componentes do *framework* PON-HD 1.0.

3.1.1 Os componentes do *framework* PON-HD 1.0

Buscando uma análise mais profunda do *framework* PON-HD 1.0, serão apresentados a seguir cada um dos componentes que foram desenvolvidos para compor a implementação em *hardware* digital do PON. O método utilizado na implementação destes componentes foi a criação de um componente VHDL para cada componente do *framework* PON-HD 1.0. Optou-se pela implementação dos componentes em VHDL pela grande compatibilidade com as ferramentas de síntese existentes no mercado.

Levando em conta o modelo teórico-estrutural do PON, é possível observar que alguns componentes não necessitam correspondentes físicos no *framework* PON-HD 1.0. Um exemplo é o FBE, que serve para agrupar *Methods* e *Attributes* e não tem função (até então) no *hardware* gerado pelo PON-HD 1.0. Para deixar mais claro quais componentes do PON possuem correspondentes físicos no *framework* PON-HD 1.0 e quais não possuem a Figura 23 apresenta o diagrama de blocos já apresentado Figura 9 com os elementos do PON que não possuem correspondentes físicos no *framework* PON-HD 1.0 marcados como “Conceptual” e com as bordas tracejadas.

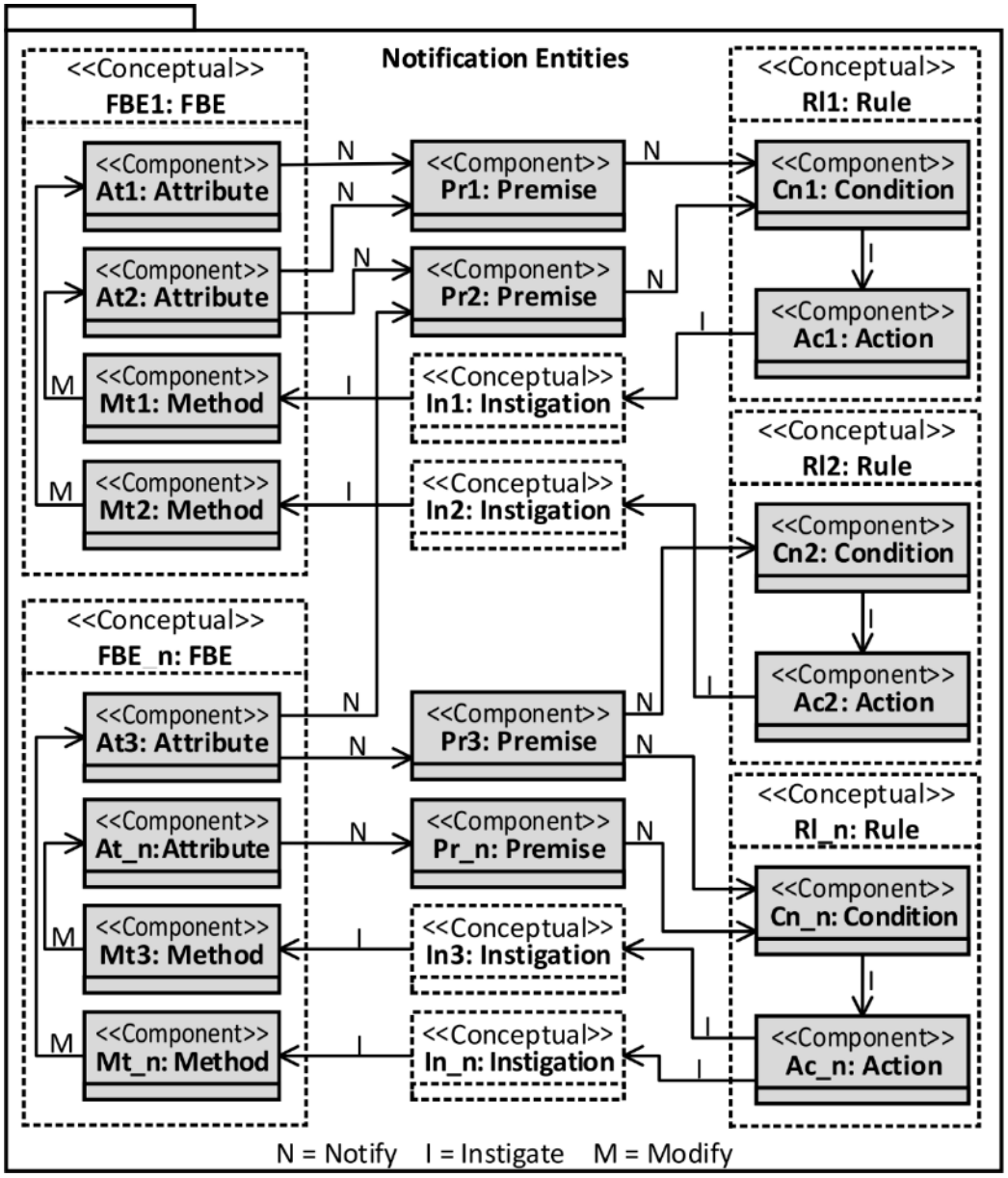

Figura 23 - Diagrama de blocos do PON-HD

Fonte: Autoria própria.

Outro exemplo de componente do PON que (até então) não possui correspondente no PON-HD 1.0 são as *Rules.* As *Rules*, assim como as *FBEs*, teriam no *framework* PON-HD 1.0 principalmente a função de agrupar os elementos e, portanto, não precisam ter nenhuma implementação em *hardware*. Cada *Rule* agrega uma *Action* e uma *Condition* e serve principalmente como uma abstração de alto nível, sendo sua função no PON ajudar a organizar o projeto. Entretanto, ele não se fez (ainda) pertinente em termos de *hardware*.

Os componentes do PON que possuem correspondentes no *framework* PON-HD 1.0 terão o detalhamento de seu funcionamento apresentado a seguir. Também serão apresentados detalhes sobre sua implementação no *hardware*, iniciando pelo *Attribute*.

Para facilitar a compreensão do funcionamento de cada um dos componentes é necessário inicialmente explicar o funcionamento de uma entidade do VHDL chamada "*generic*". Em suma, esta entidade é utilizada para declarar constantes genéricas globais (PEDRONI, 2010). No PON-HD 1.0 esta entidade é utilizada para parametrizar os componentes.

3.1.1.1 Attribute

Observando a estrutura do PON, pode-se notar que o *Attribute* é o elemento responsável por armazenar os dados. No *hardware* digital os elementos que naturalmente armazenam os dados são os registradores, assim, é também natural que no PON em *hardware* o *Attribute* seja composto por registradores. O *Attribute* executa o armazenamento dos dados nos registradores em sincronismo com o *clock*.

Como um dos requisitos do *framework* PON-HD 1.0 é que todo o ciclo de notificação aconteça em um único ciclo de *clock*. Para simplificar o processo, o *Attribute* é o único elemento sincronizado pelo *clock* no *framework* PON-HD 1.0. Todos os outros elementos são totalmente combinacionais. Essa estrutura permite que o *framework* PON-HD 1.0 apresente relativamente baixa latência, o que será objeto de discussão subsequentemente na seção 3.1.1.2 e nas seguintes.

Para permitir que qualquer tipo de dado digital seja armazenado, os *Attributes* devem possuir um número ajustável de bits. Assim, como o *Attribute* é implementado na forma de um componente VHDL, o número de bits que ele pode armazenar é configurável por meio de parâmetros do tipo "*generics*". O conteúdo armazenado nos *Attributes*, por sua vez, é utilizado pelas *Premises* e pelos *Methods* em suas computações.

Outra característica importante dos *Attributes* é que o número de *Methods* que podem atualizar o valor nele armazenado é variável, de acordo com as características da aplicação. Assim, o número de entradas que o *Attribute* suporta deve ser também ajustável através de parâmetros do tipo "*generics*". Ainda, como os *Methods* são capazes de atualizar o conteúdo dos *Attributes*, armazenando neles o resultado de suas computações o ciclo de inferência é completado por aqueles, ao bem da verdade.

Outrossim, devido as características do PON, dois ou mais *Methods* podem desejar atualizar um mesmo *Attribute* ao mesmo tempo. Neste caso, acontece um conflito de acesso ao *Attribute*. Um mecanismo de resolução de conflitos simples e eficiente foi implementado para solucionar este tipo de problema. A estratégia adotada foi implementar um mecanismo de prioridade, onde cada entrada tem uma prioridade diferente. Segundo este mecanismo, uma entrada de menor índice (Aquela que aparece antes no código em LingPON-HD 1.0 por exemplo) tem prioridade sobre uma entrada de maior índice. A Figura 24 apresenta a estrutura de um *Attribute* de 32 bits com duas entradas.

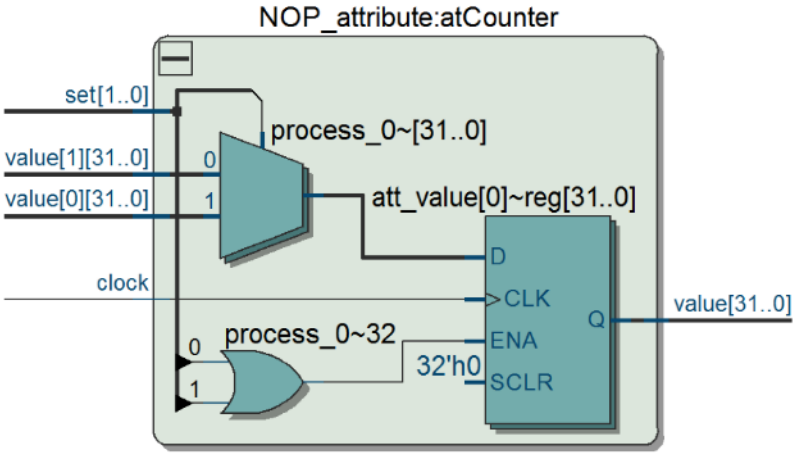

Figura 24 - Exemplo de *Attribute*.

Fonte: Autoria própria.

Nesta Figura 24 a entrada *value[0]* tem prioridade sobre a entrada *value[1]*, sendo que seria assim por diante caso houvessem mais entradas.

Os *Methods*, que por ter menor prioridade, não tem seus valores utilizados na atualização dos *Attributes* tem seu resultado perdido. Cabe, neste caso, ao desenvolvedor projetar o código em LingPON-HD de forma a prever esta situação.

Em tempo, a estrutura proposta para o *Attribute* exigiu a definição de um tipo muito específico de dados, pois as entradas podem variar em número (dependendo do número de *Methods* que acessam o *Attribute*) e podem também variar em número de bits (conforme as características do dado armazenado). Assim um tipo de dado de duas dimensões variáveis é necessário. O VHDL não possui nativamente este tipo de dados, porém, a partir da versão 2008 é possível definir este tipo de dados manualmente. A Figura 25 apresenta o código VHDL utilizado para definir o tipo de dados usado na entrada dos *Attributes*.

```
-- Package for array of STD_LOGIC_VECTOR on PORTS
LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE data_type_pkg IS
    TYPE data IS ARRAY(NATURAL RANGE <>) OF STD_LOGIC_VECTOR;
END PACKAGE data_type_pkg;
```

Figura 25 - Código VHDL do tipo de dados do *Attribute*.

Fonte: Autoria própria.

A estrutura do componente criado em VHDL para o *Attribute* permite também que se defina através de um parâmetro do tipo "*generics*" o seu valor inicial. Estes valores são assumidos pelos *Attributes* no início das computações, após a energização do circuito.

O código em VHDL do *Attribute* foi salvo como componente VHDL em um arquivo chamado "NOP_attribute.vhd". Este arquivo, conjuntamente com o arquivo "data_type_pkg.vhd", que define o tipo de dados utilizado, pode ser incluído em qualquer projeto permitindo assim o uso deste elemento do PON-HD 1.0. O conteúdo do arquivo "NOP_attribute.vhd" é apresentado no Apêndice B.

3.1.1.2 Method

No PON-HD 1.0 os *Methods* são normalmente responsáveis pelas computações outras que as lógico causais, como cálculos matemáticos, atuando sobre os valores dos *Attributes* e armazenando neles seus resultados. Segundo a implementação atual do *framework* PON-HD 1.0, proposto neste trabalho, qualquer circuito combinacional de duas entradas e uma saída pode ser implementado no *Method*. É também necessário que o número de bits das entradas e da saída sejam compatíveis com os *Attributes* envolvidos nas computações. Na versão atual, estão implementadas apenas operações aritméticas e lógicas simples. A Tabela 3 apresenta um resumo das operações que foram implementadas até o momento.

Tabela 3 - Operações realizadas pelos *Methods*.

	Operação	Descrição
0	=	Atribuição
1	+	Soma
2	-	Subtração
3	*	Multiplicação
4	/	Divisão

5	ABS	Valor absoluto
6	REM	Resto da divisão de A por B com o sinal de A
7	MOD	Resto da divisão de A por B com o sinal de B
8	NOT	Operação lógica NOT (Inversão de bits)
9	AND	Operação lógica E
10	NAND	Operação lógica E negada
11	OR	Operação lógica OU
12	NOR	Operação lógica OU negada
13	XOR	Operação lógica OU exclusivo
14	XNOR	Operação lógica OU exclusivo negada
15	SLL	Deslocamento lógico para a esquerda
16	SRL	Deslocamento lógico para a direita
17	SLA	Deslocamento aritmético para a esquerda
18	SRA	Deslocamento aritmético para a direita
19	ROL	Rotação para a esquerda
20	ROR	Rotação para a direita

Fonte: Autoria própria.

Nas operações aritméticas, realizadas pelos *Methods*, os dados dos *Attributes* são interpretados como números inteiros com sinal. Por sua vez, nas operações lógicas realizadas pelos *Methods*, os dados dos *Attributes* são interpretados como simples conjuntos de bits. Algumas destas operações exigem uma única entrada (como a operação de atribuição, por exemplo) e outras exigem duas entradas (como a operação de soma, por exemplo), conforme as características da operação a ser executada.

Isto posto, o *framework* PON-HD 1.0 ajusta o componente do *Method* conforme a necessidade. Ainda as entradas dos *Methods* podem ser valores constantes ou valores oriundos dos *Attributes*, conforme as características da aplicação. A Figura 26 mostra a estrutura de um *Method* que executa a operação de soma entre um *Attribute* e uma constante, ambos valores de 32 bits.

O PON-HD Prototipal, por sua vez, possui uma implementação completamente diferente para o *Method*. Como apresentado anteriormente, na Figura 15 é possível observar que a implementação do *Method* no PON-HD prototipal, utiliza um registrador para armazenar o sinal de aprovação do mesmo. Esta estrutura compromete a performance pois atrasa em um ciclo de *clock* a execução do *Method*.

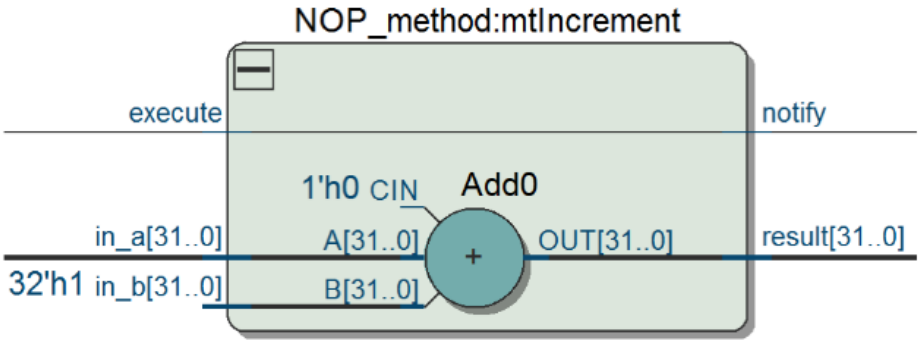

Figura 26 - Exemplo de *Method*.

Fonte: Autoria própria.

Assim como os *Attributes*, os *Methods* também foram implementados como componentes VHDL, desta forma a sua utilização nas aplicações é facilitada. Como os *Methods* realizam várias operações diferentes é necessário configurá-los para a operação desejada. Esta configuração é realizada através de um parâmetro do tipo "*generics*". O número de bits dos dados operados pelos *Methods* também deve ser informado, de forma que o componente possa realizar corretamente as operações. Um parâmetro do tipo "*generics*" é também usado para este fim.

Devida a forma escolhida para sua implementação, apenas uma operação pode ser executada em cada *Method*. Para operações mais complexas devem ser associados vários *Methods*.

O código em VHDL do *Method* foi salvo como componente VHDL em um arquivo chamado “NOP_method.vhd”. O conteúdo deste arquivo é apresentado no Apêndice C.

3.1.1.3 Premise

As operações lógico-relacionais estabelecidas no PON são realizadas no *framework* PON-HD 1.0 pelas *Premises*. Estas operações são realizadas pelas *Premises* sob o conteúdo de suas duas entradas, que podem ser valores oriundos dos *Attributes* ou constantes. O resultado das operações lógico-relacionais realizadas pelas *Premises* é um valor binário (verdadeiro ou falso) que indica se o condicional de teste foi satisfeito ou não.

A Tabela 4 apresenta as operações lógico-relacionais executadas pelas *Premises*.

Tabela 4 - Operações realizadas pelas *Premises*.

	Operação	Descrição
1	==	Verifica se os dois *Attributes* são iguais.
2	/=	Verifica se os dois *Attributes* são diferentes.
3	<	Verifica se o primeiro *Attribute* é menor que o segundo.
4	>	Verifica se o primeiro *Attribute* é maior que o segundo.
5	<=	Verifica se o primeiro *Attribute* é menor ou igual ao segundo.
6	>=	Verifica se o primeiro *Attribute* é maior ou igual ao segundo.

Fonte: Autoria própria.

A Figura 27 mostra a estrutura de uma *Premise* que executa a operação "menor que" entre um *Attribute* e uma constante, ambos valores de 32 bits.

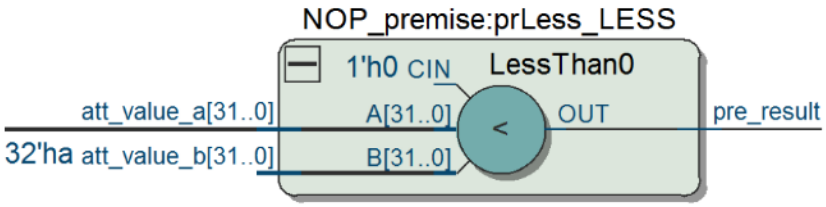

Figura 27 - Exemplo de *Premise*.

Fonte: Autoria própria.

Assim como os componentes anteriores, a *Premise* também foi implementada como um componente VHDL. Neste âmbito é necessário configurar o componente VHDL que descreve a *Premise* com a operação lógico-relacional que deve ser executada. Esta configuração é realizada através de um parâmetro do tipo "*generics*". O número de bits dos dados utilizados na operação pela *Premise* também deve ser informado. Um parâmetro do tipo "*generics*" é também utilizado para este fim.

O código em VHDL da *Premise* foi salvo como componente VHDL em um arquivo chamado “NOP_ premise.vhd”. O conteúdo deste arquivo é apresentado no Apêndice D.

3.1.1.4 Outros componentes do PON-HD 1.0

Como já foi mencionado, levando em conta o modelo estrutural do PON, alguns componentes não possuem correspondentes físicos no *hardware*. Outros componentes

executam operações relativamente simples e não necessitam de circuitos complexos para sua implementação.

Como exemplo têm-se as *Conditions*, as quais exercem a função de concatenar as *Premises*. Uma *Condition* é responsável por executar a operação lógica E ou OU entre os valores das *Premises* que a compõe. Este componente é facilmente implementado em *hardware* através da utilização de portas lógicas E ou OU. Assim, não foi necessário criar um componente VHDL para esta função. Um exemplo de *Condition* é representado pela porta E da Figura 28, a qual concatena os sinais das *Premises* "prEnd" e "prLess".

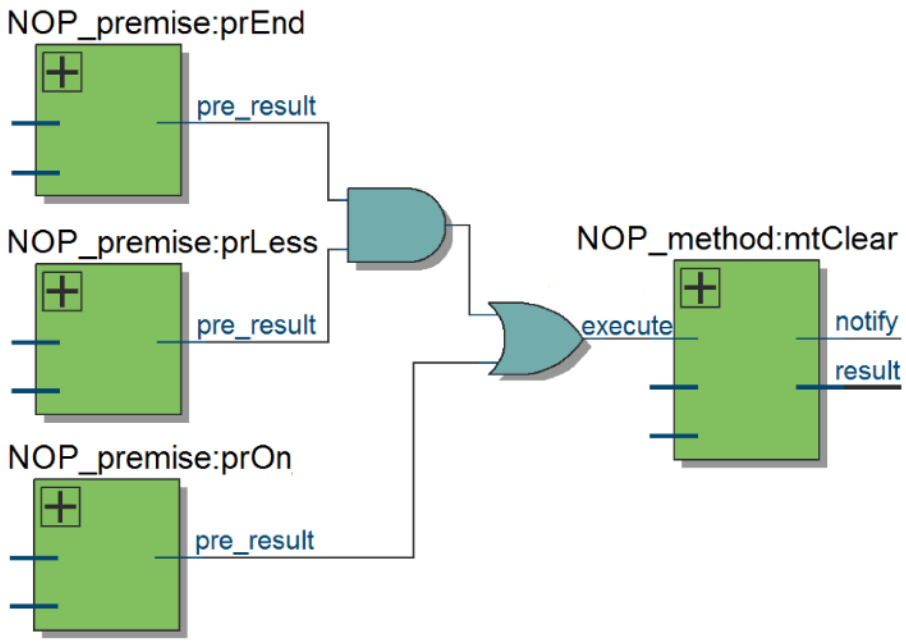

Figura 28 - Exemplo de *Condition* e *Action*.

Fonte: Autoria própria.

Uma situação parecida acontece com as *Actions* e as *Instigations*, que são responsáveis por transportar o sinal de ativação das *Conditions* para os *Methods*, seguindo as ligações estabelecidas no projeto. Para compor esta relação de ativação simples, uma porta OU é suficiente.

A Figura 28 apresenta um exemplo deste processo implementado no *hardware* utilizando o *framework* PON-HD 1.0. No exemplo desta figura o *Method* "mtClear" é ativado quando a *Premise* "prOn" é verdadeira ou quando as *Premises* "prEnd" e "prLess" são verdadeiras.

No PON-HD Prototipal a implementação da *Action* foi completamente diferente. Como pode ser observado na Figura 18, um circuito composto por dois registradores é utilizado para este fim. Esta implementação também gera atrasos, uma vez que insere mais um ciclo de *Clock* no circuito de ativação dos *Methods*.

Estas são enfim as implementações dos componentes do PON em *hardware*. A seguir será discutido a utilização destes componentes como uma ferramenta de desenvolvimento de *hardware* em alto nível.

3.1.2 O PON-HD 1.0 como ferramenta de desenvolvimento em alto nível

As características do PON permitem desenvolver as aplicações de forma mais simples e intuitiva, pois o conhecimento embutido em uma aplicação em PON-HD 1.0 é organizado na forma de fatos e de regras. Sendo assim, pode-se afirmar que o PON-HD 1.0 fornece ao desenvolvedor uma abordagem de alto nível para resolver o problema do desenvolvimento de *hardware*. Nesta abordagem o desenvolvedor se preocupa apenas com o fluxo de notificações e os elementos do PON-HD 1.0 e não precisa se preocupar tanto com a complexa sintaxe das linguagens de descrição de *hardware*.

Outra importante característica da utilização do PON-HD 1.0 no desenvolvimento de circuitos em *Hardware* digital é a redução dos erros de programação. Como os elementos do PON-HD 1.0 são predefinidos e testados, a probabilidade de erros diminui. Característica esta comum as ferramentas de síntese em alto nível (MEEUS *et al*., 2012).

Como os elementos do PON-HD 1.0 são coesos e desacoplados, é possível afirmar que estes elementos são dissociados uns dos outros, sendo interligados apenas pelas notificações. Esta estrutura de projeto permite que o desenvolvedor, através do PON-HD 1.0, se preocupe apenas com um elemento de cada vez, facilitando o desenvolvimento de uma determinada solução.

A utilização direta dos componentes VHDL do *framework* PON-HD 1.0 permite criar uma solução em *hardware* segundo os preceitos do PON. Contudo, ainda exige que o desenvolvedor tenha algum conhecimento de VHDL. Para aumentar ainda mais o nível de abstração, o desenvolvedor pode descrever sua aplicação usando a linguagem LingPON-HD 1.0, que com a ajuda do compilador PON-HD 1.0 produz todo o código VHDL necessário. Como vantagem adicional, a verificação da lógica de funcionamento do código em LingPON-

HD 1.0 pode ser feita em alto nível com a utilização do simulador PON-HD 1.0. Estas ferramentas são descritas em mais detalhes na seção 3.1.3.

3.1.3 Ferramentas de desenvolvimento PON-HD 1.0

Com o intuito de facilitar a utilização dos componentes VHDL desenvolvidos para o *framework* PON-HD 1.0 foi desenvolvido um conjunto de ferramentas de *software*. Estas ferramentas permitem a utilização do PON no desenvolvimento de *hardware* de forma vantajosa em relação ao projeto baseado puramente em VHDL. Este conjunto de ferramentas é composto pela linguagem de programação LingPON-HD 1.0, pelo compilador PON-HD 1.0 e pelo simulador PON-HD 1.0.

Com estas ferramentas, desenvolvedores que não estejam muito familiarizados com linguagens de descrição de *hardware* podem escrever suas aplicações em uma linguagem mais simples e amigável, pois se trata de uma linguagem baseada em regras de alto nível.

A Figura 29 apresenta um diagrama com o processo de compilação e depuração de aplicações criadas com o ferramental PON-HD 1.0.

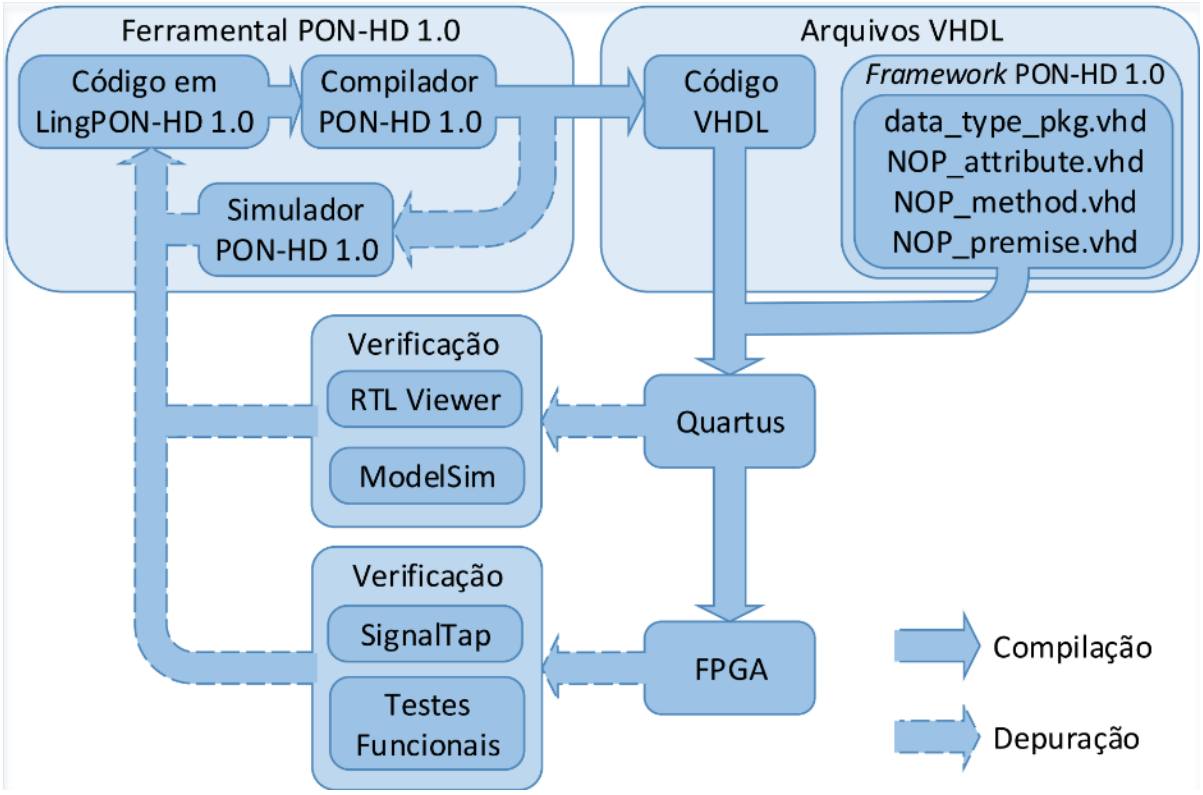

Figura 29 - Compilação e depuração de aplicações em PON-HD.

Fonte: Autoria própria.

O compilador gera automaticamente todos os arquivos VHDL necessários. Ainda, o simulador permite que estes desenvolvedores verifiquem se o comportamento do programa está de acordo com o esperado. Entretanto, é ainda necessário que o desenvolvedor tenha um conhecimento mínimo das ferramentas de síntese para que possa compilar os arquivos VHDL gerados e configurar a FPGA de forma apropriada.

A linguagem LingPON-HD 1.0, o compilador PON-HD 1.0 e o simulador PON-HD 1.0 são descritos a seguir.

3.1.3.1 LingPON-HD 1.0

A linguagem LingPON-HD 1.0 é uma derivação da linguagem LingPON 1.0, já descrita anteriormente. A linguagem LingPON 1.0 foi desenvolvida para criar aplicações de *software* em PON para computadores usuais (i. e. von Neumann), não apresentado todas as características necessárias a descrição de *hardware* digital. Neste contexto, foi desenvolvida pelo autor deste trabalho uma derivação desta linguagem com as características necessárias a descrição de *hardware* digital, a LingPON-HD 1.0.

Esta linguagem, chamada de LingPON-HD 1.0, acrescenta à linguagem LingPON 1.0 alguns elementos necessários e geração de código para *hardware* digital. A compatibilidade entre as duas variantes da linguagem PON é desta forma mantida, bastando apenas acrescentar alguns comandos a um programa em LingPON 1.0 para torna-lo compatível com a LingPON-HD 1.0.

Estes comandos, que necessitam ser adicionados, dizem respeito ao número de bits de cada um dos *Attributes*, as conexões de entrada e saída do circuito e ao nome da entidade gerada no código VHDL.

O LingPON 1.0 também apresenta estruturas utilizadas exclusivamente para a criação de sub-rotinas de *software*, que não são utilizadas na descrição do *hardware*. A Linguagem LingPON-HD 1.0 permite que os *Methods* sejam compostos apenas por comandos simples, que executam operações lógicas e aritméticas.

A Figura 30 apresenta um programa em LingPON-HD 1.0. Ao bem da verdade, este é o programa que permitiu gerar o circuito apresentado anteriormente na Figura 22.

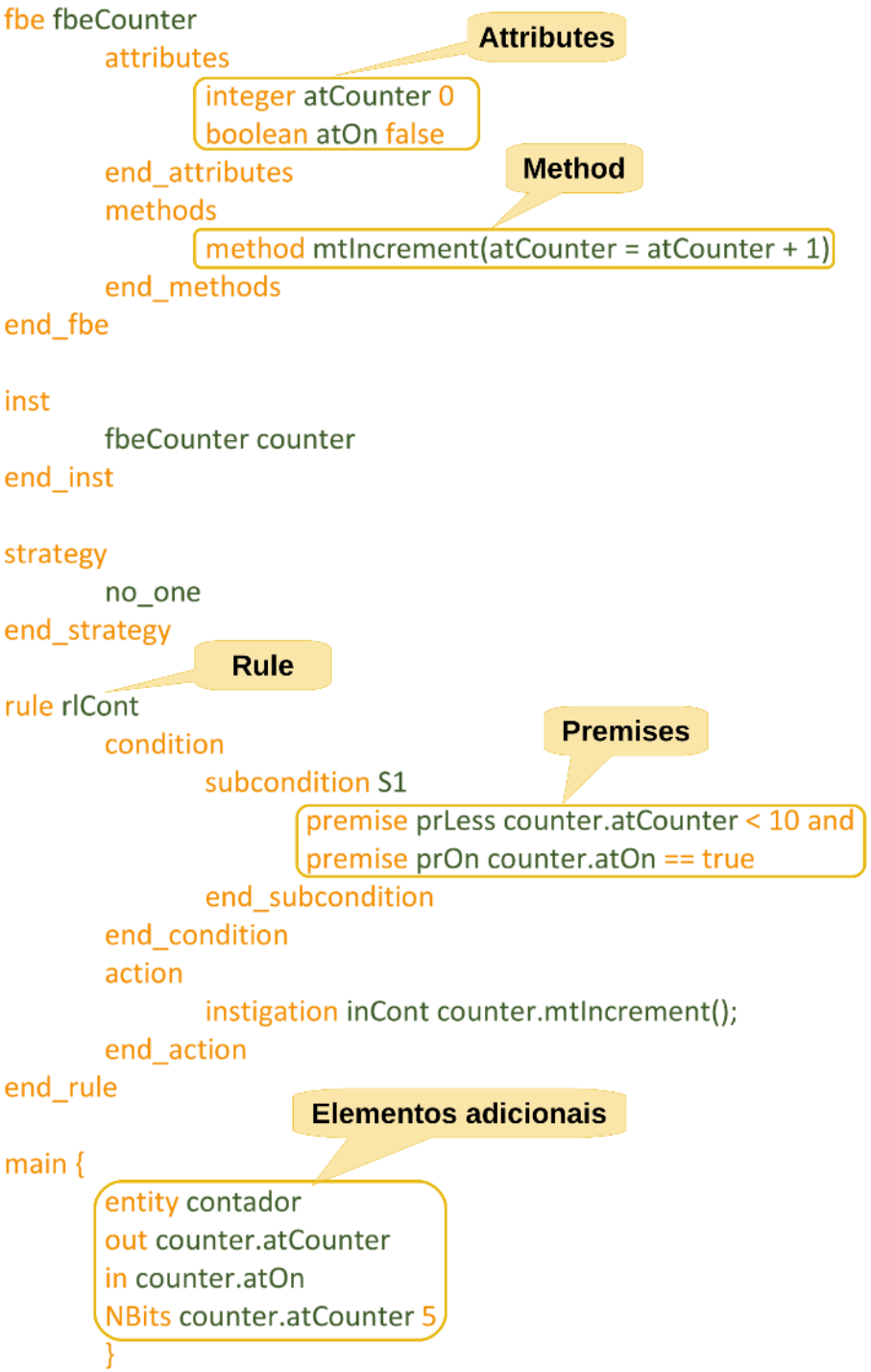

Figura 30 - Exemplo de programa em LingPON-HD 1.0.

Fonte: Autoria própria.

Este exemplo, apesar de simples, apresenta todos os principais elementos do *framework* PON-HD 1.0 na linguagem LingPON-HD 1.0. É possível identificar no código as entidades do PON, como *Attributes*, *Premises*, *Rules* e *Methods*. Também é possível observar as operações realizadas por estes elementos e os valores envolvidos, como por exemplo a

Premise “prLess”, que verifica se o *Attribute* “counter.atCounter” é menor que a constante 10. É ainda possível observar as interconexões entre os elementos, como por exemplo, na *Rule* “rlCont”, que executa o *Method* “counter.mtIncrement” quando as *Premises* “prOn” e “prLess” forem verdadeiras.

3.1.3.2 Compilador PON-HD 1.0

Para que a linguagem LingPON-HD 1.0 possa ser utilizada na síntese de *hardware* digital é necessário um compilador que transcreva os programas nela escritos em um código que possa ser utilizado na geração do *hardware*. Neste sentido foi desenvolvida uma derivação do compilador LingPON 1.0, que converte programas escritos em LingPON-HD 1.0 para VHDL a luz do PON-HD 1.0.

Este arquivo VHDL, junto com os componentes do PON-HD 1.0, pode então ser compilado pelas ferramentas fornecidas pelos fabricantes de *hardware*. Na verdade, o arquivo VHDL gerado pela compilação do programa em LingPON-HD 1.0 utiliza instâncias dos componentes do *framework* PON-HD 1.0. Cada componente instanciado é parametrizado através de seus “generics” para executar a função desejada, com o número de bits requerido. As interconexões entre os componentes são também realizadas neste arquivo VHDL. Para fazer estas interconexões, são utilizados sinais (“*signal*” em VHDL) do tipo “STD_LOGIC” ou “STD_LOGIC_VECTOR”, conforme a necessidade.

O compilador LingPON-HD 1.0 não foi desenvolvido a partir do zero. Toda a parte de análise léxica e semântica foi aproveitada do compilador LingPON 1.0 (RONSZCKA *et al*, 2017; FERREIRA, 2015; SANTOS *et al*, 2017). Do compilador LingPON 1.0 foi aproveitada a parte que lê o arquivo fonte de entrada e armazena os “*tokens*” em estruturas de dados. A partir destas estruturas de dados o compilador LingPON-HD 1.0 gera todas as estruturas em linguagem VHDL, de forma a implementar em *hardware* digital o que foi descrito em LingPON-HD 1.0. O compilador também foi adequado para permitir a descrição dos elementos necessários a geração, como descrito na seção anterior (3.1.3.1).

Outra tarefa realizada pelo compilador é a geração de um arquivo contendo as estruturas de dados necessárias ao funcionamento do simulador PON-HD 1.0, que será descrito na seção seguinte.

3.1.3.3 Simulador PON-HD 1.0

Como em uma aplicação desenvolvida em PON-HD 1.0 todos os elementos são executados de forma paralela, não é fácil prever como estes elementos irão interagir entre si, bem como com as entradas e saídas do circuito. Para que se possa utilizar os simuladores fornecidos com as ferramentas tradicionais de síntese de *hardware*, é necessário realizar a compilação do código VHDL, processo normalmente bastante demorado. Além disso, estas simulações são realizadas em nível de sinais digitais, dificultando a identificação das falhas a nível de programa em LingPON-HD 1.0. O mesmo acontece quando os circuitos são testados diretamente no *hardware*.

Neste contexto, bem como com o objetivo de facilitar a depuração de programas escritos em LingPON-HD 1.0, foi desenvolvido pelo autor deste trabalho o simulador PON-HD 1.0. Este simulador foi desenvolvido em linguagem C++, com o auxílio do ambiente de desenvolvimento "Qt C++". O mesmo pode ser executado nos sistemas operacionais Windows e Linux. A Figura 31 apresenta a interface do simulador PON-HD 1.0, com a descrição de seus principais comandos.

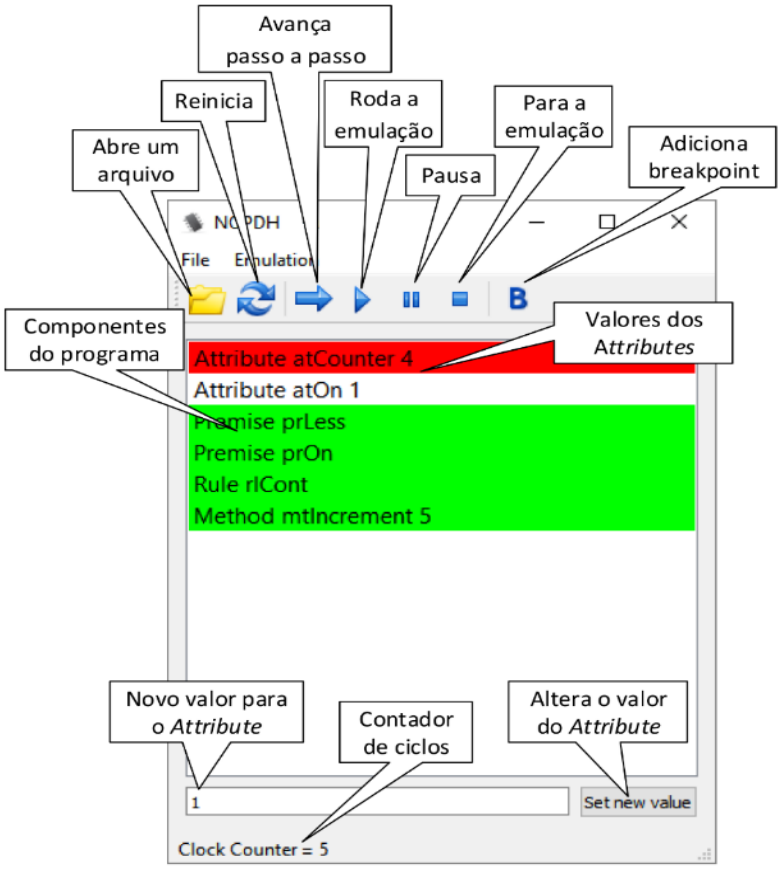

Figura 31 - Interface do simulador PON-HD.

Fonte: Autoria própria.

O simulador PON-HD 1.0 permite visualizar o comportamento de cada um dos elementos do programa em LingPON-HD 1.0, bem como interagir com os mesmos através da alteração dos valores armazenados nos *Attributes*.

Para o funcionamento do simulador PON-HD 1.0 é necessário que um arquivo de entrada forneça um conjunto de estruturas de dados com todos os elementos do programa e suas interconexões. Este arquivo é gerado pelo compilador PON-HD 1.0, durante a compilação do programa.

A principal função do simulador é executar o programa passo a passo, ou até que um ponto de parada (*breakpoint*) seja encontrado, de forma que o estado das *Premises, Rules e Methods* e os valores dos *Attributes* possam ser verificados. Como pode ser observado na Figura 31, o estado destes elementos é representado por um código de cores. O estado das *Premises* é indicado pela cor verde quando a mesma é inferida como verdadeira e pela cor vermelha quando é inferida como falsa, as *Rules* e os *Methods* são marcados com a cor verder quando ativados e os *Attributes*, são marcados com a cor vermelha toda vês que seu valor é alterado.

Para o controle da simulação são utilizados os comandos de "*Restart*", "*Play*", "*Pause*", "*Stop*" e "*Step*" que respectivamente servem para reiniciar, iniciar, pausar, parar e executar um passo (um ciclo de *clock*) da simulação. É também possível adicionar pontos de parada, que param a simulação em momentos determinados, como quando um *Attribute* atinge determinado valor ou uma *Premise* se torna verdadeira.

A interação do usuário com o programa é realizada pela alteração dos valores dos *Attributes*. Assim, é possível simular os sinais de entrada ou testar condições específicas.

O simulador PON-HD 1.0 se mostrou uma ferramenta importante, pois permite testar o comportamento do programa sem a necessidade de compilar o código VHDL.

3.2 Considerações sobre o capítulo

Este capítulo apresentou as ferramentas que foram desenvolvidas para permitir que o ferramental PON-HD 1.0 fosse utilizado como uma ferramenta de desenvolvimento em alto nível. Todo este conjunto forma o cerne deste trabalho de doutorado.

É importante salientar que para o desenvolvimento do ferramental PON-HD 1.0 foi exigido primeiramente um esforço no entendimento de vários temas relacionados, como, paradigmas de programação, PON, PON-HD Prototipal, dispositivos de lógica reconfigurável,

ferramentas de síntese e VHDL. Durante o desenvolvimento deste ferramental foram ainda exigidos estudos em Engenharia de Requisitos, Engenharia de Sistemas, Engenharia de *Software*, Circuitos Digitais entre outros.

Uma discussão dos aspectos positivos e negativos da implementação do PON em *hardware*, bem como das dificuldades e limitações encontradas durante a elaboração desta implementação serão apresentadas junto as conclusões.

O capítulo seguinte apresenta um conjunto de experimentos realizados para verificar o funcionamento e as características do ferramental PON-HD 1.0.

4 Experimentos e testes realizados

Neste capítulo serão apresentados os experimentos realizados para verificar as funcionalidades da implementação do PON em *hardware* digital versão 1.0 (PON-HD 1.0).

A Tabela 5 apresenta resumidamente os experimentos realizados, enfatizando as ferramentas utilizadas em seu desenvolvimento (*Framework* PON-HD 1.0, LingPON-HD 1.0, simulador PON-HD 1.0 e Vivado HLS) e verificação (SignalTap II), bem como seus principais objetivos.

Tabela 5 - Resumo dos experimentos com PON-HD 1.0, seus objetivos e ferramentas utilizadas.

	***Framework* PON-HD1.0**	**LingPON-HD 1.0**	**Simulador PON-HD 1.0**	**SignalTap II**	**Comparação com Vivado HLS**
Experimento 1 "Contador"	Usado via compilador	Usado	Usado	Usado	Não
	Objetivo: Exemplo didático e validação do conjunto de ferramentas do PON-HD 1.0.				
Experimento 2 "Driver para display de 7 segmentos"	Usado via compilador	Usado	Usado	Não	Sim
	Objetivo: Comparativo de performance entre os circuitos gerados pelo PON-HD 1.0 e pela ferramenta Vivado HLS, para um circuito combinacional.				
Experimento 3 "Contador Decimal"	Usado via compilador	Usado	Usado	Usado	Não
	Objetivo: Validação do Simulador PON-HD 1.0, verificação da conectividade do circuito e validação do VHDL gerado no âmbito do PON-HD 1.0.				
Experimento 4 "Calculador de Média"	Usado via compilador	Usado	Usado	Não	Sim
	Objetivo: Comparativo de performance entre os circuitos gerados pelo PON-HD 1.0 e pela ferramenta vivado HLS, para um circuito primariamente sequencial.				
Experimento 5 "Ordenador de dados"	Usado	Parcialmente*	Não usado	Usado	Não
	Objetivo: Verificação da performance do circuito, bem como da escalabilidade do PON-HD 1.0.				
Experimento 6 "Controlador de Esteira"	Usado via compilador	Usado	Não usado	Usado	Não
	Objetivo: Verificação da facilidade de utilização e validação do ferramental PON-HD 1.0 por parte de elementos externos ao grupo de pesquisa do PON.				
Experimento 7 "Controlador de robô"	Usado via compilador	Usado	Usado	Usado	Não
	Objetivo: Resolução de conflitos, estabilidade, facilidade de utilização, validação do simulador PON-HD 1.0 e revalidação do PON-HD 1.0.				
Experimento 8	Usado	Usado	Usado	Não	Sim

“Controlador PID”	via compilador				
	Objetivo: Comparativo de performance entre os circuitos gerados pelo PON-HD 1.0 e pela ferramenta vivado HLS, para um circuito mais complexo de cunho mais prático.				

* O compilador LingPON-HD 1.0 foi utilizado apenas para um número fixo de elementos na ordenação, o restante foi implementado diretamente em VHDL, como explicado mais à frente.
Fonte: Autoria própria.

A seguir serão descritos cada um dos experimentos já listados na Tabela 5, bem como os resultados com eles alcançados e as conclusões obtidas a partir destes resultados.

4.1 Experimento 1: Contador

O experimento do contador é apresentado por motivos didáticos. Sua concepção simples facilita o entendimento dos mecanismos do ferramental PON-HD 1.0.

4.1.1 Descrição do Experimento 1 (Contador)

Este experimento foi concebido como um exemplo didático, tendo também o objetivo de verificar o funcionamento do conjunto de ferramentas como um todo. Em sua simplicidade, foi possível verificar se o programa em LingPON-HD 1.0 é corretamente compilado e o código VHDL é corretamente gerado. O funcionamento do simulador PON-HD também foi verificado com este experimento. Na Figura 32 é possível visualizar um diagrama do circuito implementado neste experimento.

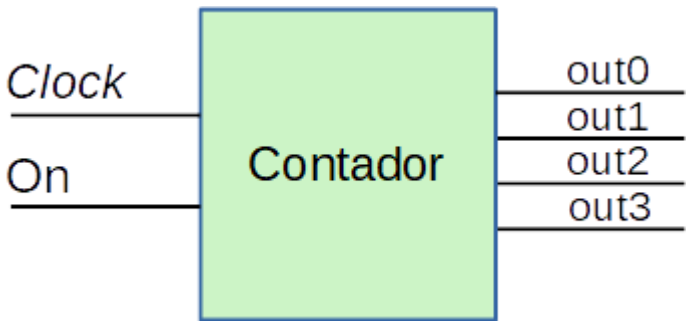

Figura 32 - Diagrama do contador
Fonte: Autoria própria.

O experimento do contador já foi apresentado nos capítulos anteriores, como exemplo, durante a apresentação do PON-HD 1.0. Neste sentido a Figura 22 apresenta o diagrama de blocos para o circuito gerado após a compilação do código VHDL e a Figura 30 apresenta o código em LingPON-HD 1.0 para este experimento. Estas figuras são aqui abaixo replicadas, Figura 33 e Figura 34, para fins de facilidade de compreensão.

Na Figura 33 é possível observar o diagrama RTL do circuito deste experimento descrito em lingPON-HD, cada bloco corresponde a um componente do PON-HD 1.0.

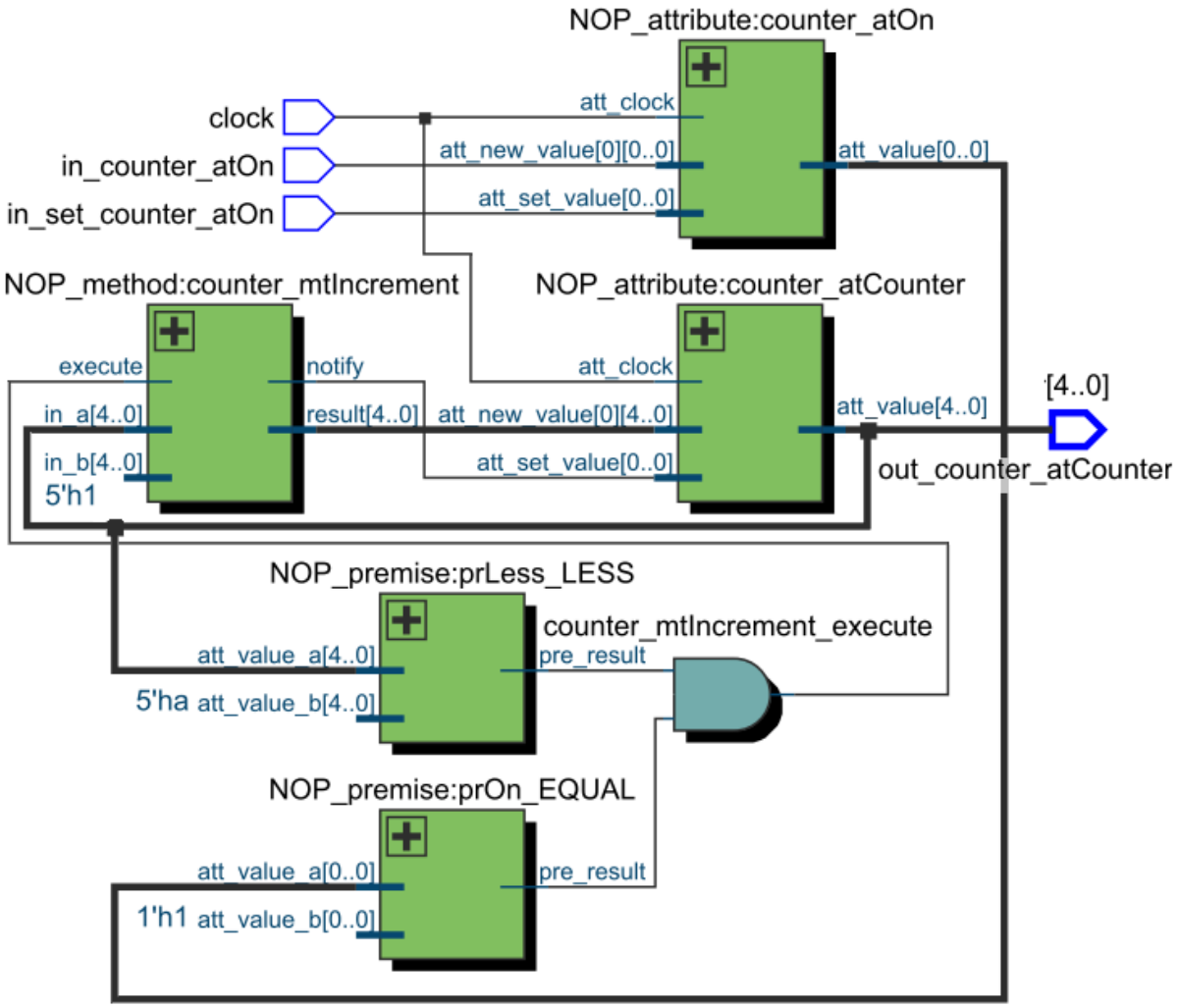

Figura 33 - Diagrama de blocos do contador.

Fonte: Autoria própria.

Observando o circuito da Figura 33 nota-se que foi implementado um sinal de entrada que não estava previsto no diagrama da Figura 32. Isto acontece devido as características do PON-HD 1.0, onde é necessário um sinal para indicar o armazenamento dos dados de entrada no *Attribute.*

A Figura 34, por sua vez, apresenta o código fonte em LingPON-HD para este experimento.

```
fbe fbeCounter
        attributes
                integer atCounter 0
                boolean atOn false
        end_attributes
        methods
                method mtIncrement(atCounter = atCounter + 1)
        end_methods
end_fbe

inst
        fbeCounter counter
end_inst

strategy
        no_one
end_strategy

rule rlCont
        condition
                subcondition S1
                        premise prLess counter.atCounter < 10 and
                        premise prOn counter.atOn == true
                end_subcondition
        end_condition
        action
                instigation inCont counter.mtIncrement();
        end_action
end_rule

main {
        entity contador
        out counter.atCounter
        in counter.atOn
        NBits counter.atCounter 5
        }
```

Figura 34 - Programa do contador.

Fonte: Autoria própria.

A Figura 35 apresenta um diagrama que ilustra o fluxo do experimento. É possível observar as várias etapas de desenvolvimento e de verificação. Os blocos correspondentes as ferramentas do PON-HD 1.0 e que foram desenvolvidos neste trabalho de doutorado são destacados.

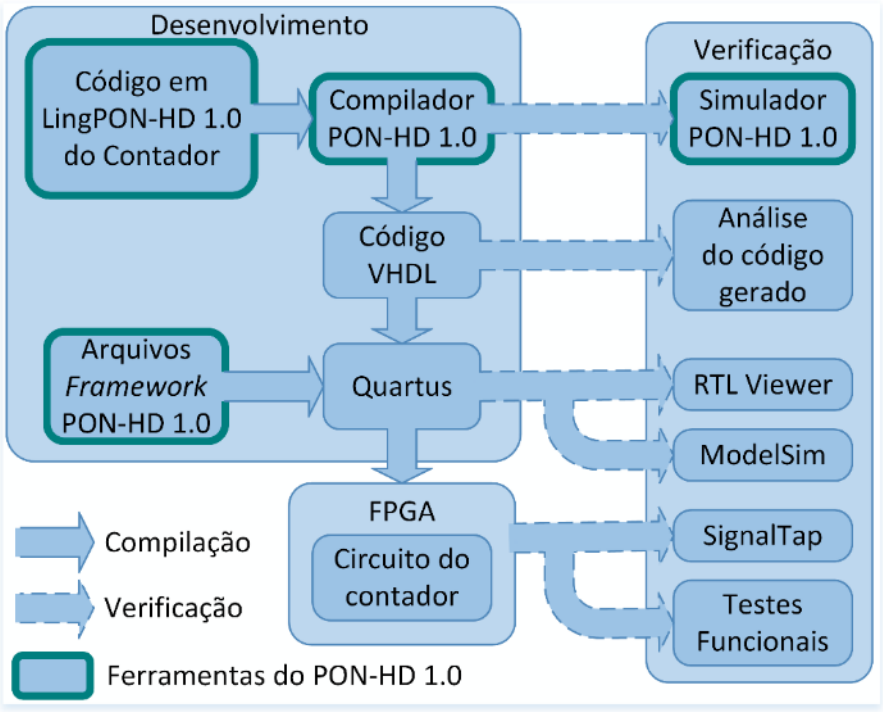

Figura 35 - Diagrama de fluxo do experimento 1.

Fonte: Autoria própria.

Como apresentado na Figura 36, o circuito deste experimento possui uma entrada de *clock* com o nome "*clock*", uma entrada que habilita ou desabilita a contagem chamada "*in_counter_atOn*" e uma entrada que habilita o registrador e armazena o valor de "*in_ counter_atOn*" no *Attribute* "*counter_atOn*" chamada "*in_set_ counter_atOn*". Por sua vez, a única saída deste circuito é o valor da contagem que fica armazenado no *Attribute* "*counter_atCounter*" e é chamada "*out_ counter_atCounter*".

A nomenclatura utilizada nos elementos do PON é gerada automaticamente pelo compilador em função do nome dos elementos e do nome da FBE a que eles pertencem. É por esta razão que os nomes são tão longos.

O circuito possui duas *Premises*, uma responsável por verificar se o *Attribute* "*counter_atOn*" é verdadeiro chamada "*prOn*" e outra responsável por verificar se o *Attribute* "*counter_atCounter*" é menor que dez, chamada "*prLess*".

O funcionamento do circuito é o seguinte: quando a valor do *Attribute* "*counter_atOn*" é 1 (verdadeiro) e o valor do *Attribute* "*counter_atCounter*" é menor que dez, as *Premises* "*prOn*" e "*prLess*" se tornam verdadeiras e notificam o *Method* "*counter_mtIncrement*". O *Method* "*counter_mtIncrement*", por sua vez, incrementa o valor do *Attribute* "*counter_atCounter*", isso na borda de subida do *clock*. A contagem é interrompida assim que umas das condições se tornar falsa.

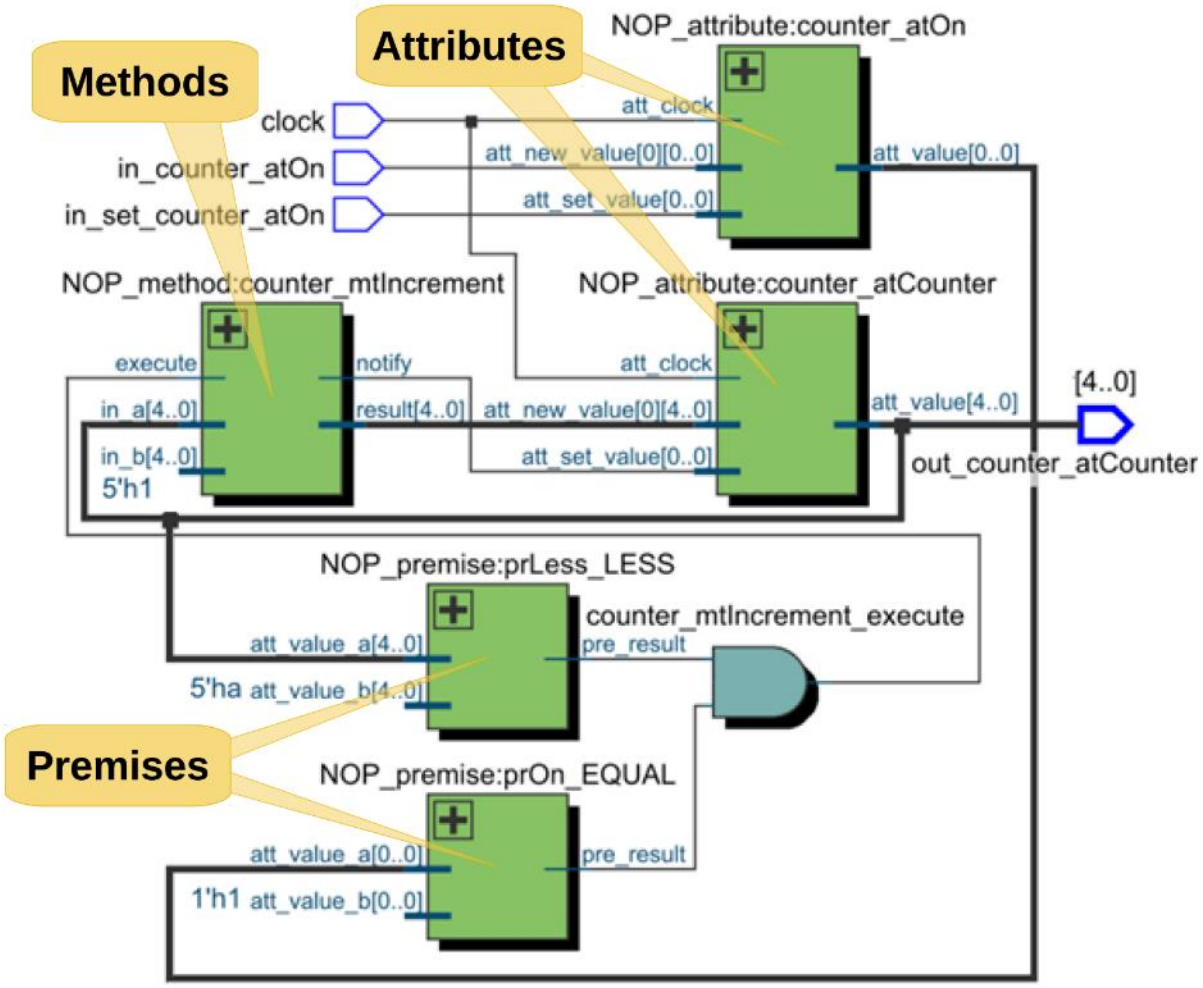

Figura 36 - Circuito do contador com seus principais elementos em destaque.

Fonte: Autoria própria.

O circuito deste experimento foi implementado em uma FPGA modelo "*Cyclone IV*" da fabricante Altera e foi compilado com o *software* "*Quartus Prime*", também da fabricante Altera. Outrossim, o funcionamento do circuito foi verificado inicialmente através de simulações com o simulador "*ModelSim*". Posteriormente o circuito foi testado diretamente na FPGA e a leitura dos resultados foi realizada com a ferramenta "*SignalTap II*". A Figura 37 apresenta os sinais de entrada e saída do experimento.

Em tempo, os diagramas apresentados na Figura 33 e na Figura 36 foram obtidos após a compilação do código VHDL com o auxílio da ferramenta "RTL Viewer". O código completo em VHDL gerado da compilação do código em LingPON-HD 1.0 deste experimento pode ser visualizado no Apêndice E.

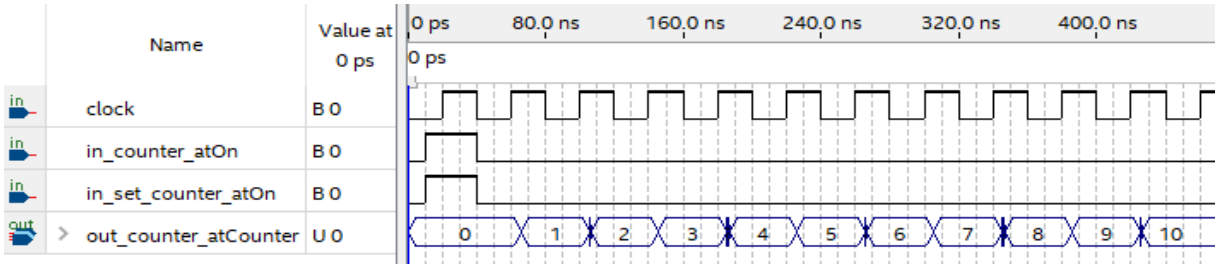

Figura 37 - Resultado do experimento do contador.

Fonte: Autoria própria.

4.1.2 Resultados do Experimento 1 (Contador)

Com este experimento, apesar de simples, foi possível verificar que o compilador interpretou corretamente o programa em LingPON-HD 1.0 e gerou corretamente o código VHDL correspondente.

O funcionamento do simulador PON-HD 1.0 também foi verificado com este experimento, pois observa-se que o comportamento do circuito implementado no *hardware* é adequadamente replicado no simulador. Ainda após a compilação do código VHDL, é assaz fácil verificar se o circuito gerado corresponde ao que foi inicialmente programado e simulado.

A Figura 36 apresenta o diagrama do circuito gerado da compilação do código VHDL. Neste diagrama fica clara a correspondência entre os componentes declarados no programa em LingPON-HD 1.0, apresentado na Figura 34, e os componentes gerados no *hardware.* A Tabela 6 apresenta esta correspondência, mostrando os elementos ativos instanciados no programa em LingPON-HD 1.0 e seu correspondente implementado no *hardware* pelo compilador. É possível notar que o circuito corresponde fielmente ao que foi programado.

Tabela 6 - Correspondência entre elementos do programa e do *hardware*.

Tipo do elemento	Elemento instanciado em LingPON-HD 1.0	Elemento implementado no *hardware*
Attribute	atCounter	NOP_attribute:counter_atCounter
Attribute	atOn	NOP_attribute: counter_atOn
Premise	prLess	NOP_premise:prLess_LESS
Premise	prOn	NOP_premise:prOn_EQUAL
Rule	rlCont	Implementado através da porta “AND”
Method	mtIncrement	NOP_method: counter_mtIncrement

Fonte: Autoria própria.

Analisando o comportamento do circuito no *hardware* foi possível verificar que os componentes do *Framework* PON-HD 1.0 desempenharam seu papel corretamente, demostrando assim que o ferramental PON-HD 1.0 é funcional, ao menos para este experimento. Esta analise se deu verificando os sinais elétricos no interior da FPGA com a ferramenta "*SignalTap II*".

Também foi possível verificar com este experimento que o simulador PON-HD 1.0 funciona corretamente, simulando o comportamento do programa em LingPON-HD 1.0 permitindo interagir visualmente com os componentes do mesmo. Esta verificação se deu através da utilização do simulador nas mais diversas situações permitidas pelo *design* deste circuito.

Levando em consideração que o ferramental PON-HD 1.0 permite através da LingPON-HD 1.0 expressividade através de regras em alto nível na descrição do circuito do experimento observa-se que o ferramental PON-HD 1.0 é relativamente fácil de utilizar, isso devido exatamente a utilização destas regras em alto nível.

A Tabela 7 apresenta as características da implementação do experimento no *hardware.*

Tabela 7 - Características de implementação do experimento 1.

	LEs	FFs	Ciclos de *clock*	Máxima Frequência
PON-HD 1.0	8	6	1 ciclos de *clock*	250 MHz (Limitado pela FPGA)
VHDL	6	4	1 ciclos de *clock*	250 MHz (Limitado pela FPGA)

Fonte: Autoria própria.

Dentre as informações da Tabela 7 pode-se destacar que foram utilizados 6 Flip-Flops para armazenar as informações dos *Attributes,* 1 para a entrada "*atOn*" e 5 para o *Attribute* "*atCounter*". Este *Attribute* utilizou 5 bits por armazenar os dados em complemento de 2, necessitando assim de um bit adicional para o sinal.

A título de comparação foi adicionado a tabela um conjunto de valores referente a implementação do mesmo circuito diretamente em VHDL. Esta implementação em VHDL utilizou menos recursos da FPGA por não registrar o sinal de ativação do contador e não utilizar complemento de 2 no sinal de saída. A frequência de operação dos circuitos foi limitada pela máxima frequência de operação da FPGA, e não pela topologia do circuito.

4.1.3 Conclusões do Experimento 1 (Contador)

O primeiro experimento é um contador, o qual serviu para exemplificar como funciona o ferramental PON-HD 1.0 e como seus componentes são implementados no *hardware*. Este experimento serviu também para verificar como os componentes do *framework* PON-HD 1.0 desempenharam seu papel no *hardware*, demostrando assim que o PON-HD 1.0 é funcional. O correto funcionamento do simulador PON-HD 1.0 pode também ser verificado durante este experimento. Assim pode-se concluir que o Ferramental PON-HD 1.0 é funcional para a descrição de circuitos em lógica reconfigurável.

4.2 Experimento 2: *Driver* para display de 7 segmentos

Neste experimento serão realizadas comparações entre o PON-HD 1.0 e a ferramenta de síntese em alto nível desenvolvida pela Xilinx chamada Vivado HLS. Para este experimento foi proposto um circuito relativamente simples e de operação sequencial.

4.2.1 Descrição do Experimento 2 (Driver para display de 7 segmentos)

O circuito proposto para este experimento é um *driver* para *display* de 7 segmentos. Sua função é receber um número codificado em um sinal de 4 bits e fornecer como saída os 7 sinais necessários para apresentar este número em um *display* de 7 segmentos. O Objetivo deste experimento é realizar um comparativo de performance entre os circuitos gerados pelo PON-HD 1.0 e pela ferramenta Vivado HLS, para um circuito combinacional.

A Figura 38 apresenta um diagrama de blocos que demostra o funcionamento do circuito proposto para o experimento.

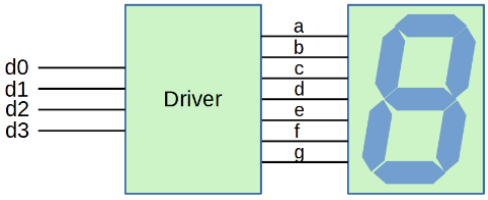

Figura 38 - Driver para *display* de 7 segmentos.

Fonte: Autoria própria.

O circuito escolhido, por sua simplicidade, possibilita observar como cada ferramenta realiza sua implementação no *hardware*. Neste experimento pretende-se observar características de performance de cada uma das implementações, em PON-HD 1.0 e em Vivado HLS. A quantidade de recursos utilizados na FPGA por cada uma das implementações também é uma característica importante que se pretende verificar neste experimento.

A Figura 39 apresenta um diagrama que ilustra o fluxo do experimento. Neste diagrama é possível observar as várias etapas de desenvolvimento e de verificação realizadas neste experimento.

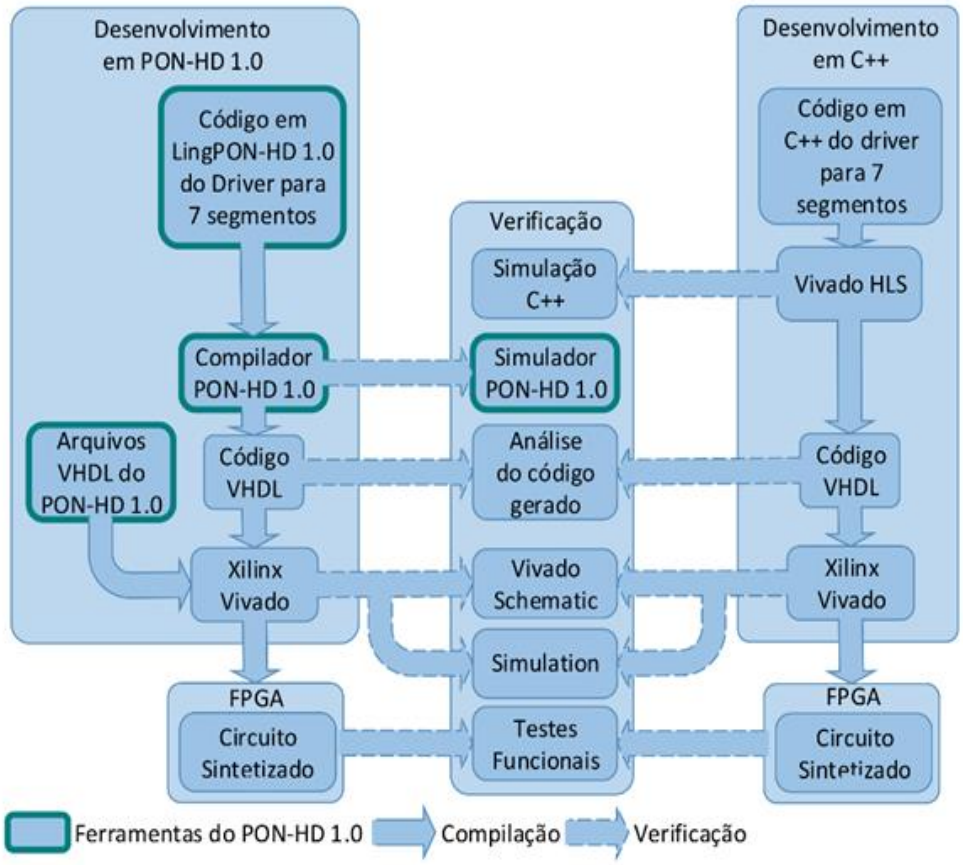

Figura 39 - Diagrama de fluxo do experimento 2.

Fonte: Autoria própria.

Houve uma grande preocupação em tornar as duas implementações, em PON-HD 1.0 e em Vivado HLS, o mais semelhante possível, através de implementações semelhantes da lógica de funcionamento, de forma a tornar as comparações mais justas.

4.2.1.1 Desenvolvimento do *driver* em PON-HD 1.0

Para a implementação do *driver* para *display* de 7 segmentos em PON-HD 1.0 inicialmente desenvolveu-se o código em LingPON-HD 1.0. Neste código as entradas e saídas foram mapeadas em dois *Attributes*. Um conjunto de *Rules* verifica o valor das entradas e o estado de uma estrada específica de *reset* através de *Premises*. Para cada número apresentado na entrada uma *Rule* específica é ativada. Esta *Rule,* quando ativada instiga um *Method*, este *Method* por sua vez atribui o valor apropriado ao *Attribute* de saída.

A Figura 40 apresenta um trecho do código em LingPON-HD 1.0 da implementação deste *driver* em PON-HD 1.0.

O código completo do programa em LingPON-HD 1.0 pode ser encontrado no Apêndice F. Este código é composto por 3 *Attributes*, o sinal de *reset*, o valor numérico de entrada e o conjunto de sinais de saída, 12 *Premises*, 11 *Rules*, 10 *Methods*.

Este código foi então compilado com o compilador PON-HD 1.0 resultando em um arquivo VHDL. Também foi utilizado nesta etapa o simulador PON-HD 1.0 para verificar se o programa implementado em LingPON-HD 1.0 funcionou corretamente.

O código VHDL gerado foi inspecionado e, estando de acordo com o esperado, prosseguiu-se para o processo de síntese do circuito. O código VHDL completo gerado automaticamente pelo compilador PON-HD 1.0 pode ser observado no Apêndice G.

Para facilitar as comparações com o código gerado pelo software Vivado HLS da Xilinx optou-se por utilizar o conjunto de ferramentas Vivado 2018.1 para a síntese do código VHDL gerado pelo PON-HD 1.0.

É importante esclarecer que apesar da similaridade nos nomes são duas ferramentas diferentes deste mesmo fabricante. A ferramenta “Vivado HLS” permite a descrição do *hardware* em alto nível, utilizando linguagens derivadas do C e do C++, e algumas outras tarefas relacionadas, enquanto que a ferramenta “Vivado” realiza a síntese do código VHDL, sua simulação entre outras coisas.

A FPGA escolhida para as simulações é do modelo XA7A12TCPG238-2L da família Artix-7 fabricada pela Xilinx.

Para verificar o funcionamento do circuito foram realizados alguns testes. Inicialmente verificou-se o esquemático (RTL) do circuito gerado pela ferramenta de síntese. Subsequentemente uma análise mais profunda do funcionamento do circuito foi realizada através de simulações e verificações funcionais.

```
fbe fbeDriverDisplay
        attributes
                integer entrada 64
                integer saida 0
                boolean reset false
        end_attributes
        methods
                method mtVal0(saida = 64)
                method mtVal1(saida = 121)
                method mtVal2(saida = 36)
                method mtVal3(saida = 48)
                method mtVal4(saida = 25)
                method mtVal5(saida = 18)
                method mtVal6(saida = 2)
                method mtVal7(saida = 120)
                method mtVal8(saida = 0)
                method mtVal9(saida = 16)
        end_methods
end_fbe

inst
        fbeDriverDisplay DD
end_inst

strategy
        no_one
end_strategy

rule rlReset
        condition
                subcondition SubReset
                        premise prReset DD.reset == true
                end_subcondition
        end_condition
        action
                instigation inReset DD.mtVal0();
        end_action
end_rule
```

Figura 40 - Código em LingPON-HD 1.0 para o experimento 2.

Fonte: Autoria própria.

A Figura 41 apresenta o bloco de circuito gerado pela síntese do código VHDL para o *driver* de *display* de 7 segmentos, resultante do programa em LingPON-HD 1.0.

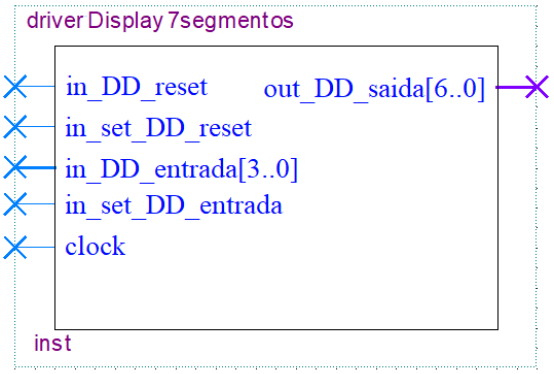

Figura 41 - Bloco gerado pela síntese do código do experimento 2.

Fonte: Autoria própria.

Nesta figura é possível observar a entrada de *clock*, a entrada de *reset* e seu sinal de atribuição (set) e a entrada de dados com seus 4 bits e seu respectivo sinal de atribuição. A saída do circuito também pode ser observada, como seus 7 bits. Estes sinais de atribuição não estavam previstos no diagrama da Figura 38, mas são implementados automaticamente pelo PON-HD, de forma a gerenciar o armazenamento de dados nos *Attributes*.

Para realizar as comparações propostas o mesmo circuito foi implementado na ferramenta Vivado HLS. A seção a seguir apresenta esta implementação.

4.2.1.2 Desenvolvimento do *driver* em Vivado HLS

Para a implementação do *driver* para *display* de 7 segmentos no Vivado HLS utilizou-se a versão 2018.1 desta ferramenta. A maior vantagem do Vivado HLS é permitir a descrição do circuito utilizando linguagens de mais alto nível. Para este experimento optou-se por utilizar a linguagem C++.

A escolha do Vivado HLS como ferramenta de comparação com o PON-HD 1.0 se deu inicialmente por sugestão dos membros da banca de qualificação que antecedeu a realização deste trabalho de pesquisa. A escolha é pertinente por se tratar de uma ferramenta robusta e de qualidade reconhecida, além disso, esta ferramenta apresenta semelhanças com o PON-HD 1.0 como, por exemplo, a geração de código VHDL a partir de uma linguagem de alto nível.

A comparação do PON-HD 1.0 com a ferramenta Vivado HLS permite verificar características importantes relacionadas ao desenvolvimento em alto nível. A ferramenta Vivado HLS é bastante completa, permitindo a descrição e o teste dos algoritmos em alto nível através de linguagens como C e C++ (NANE *et al*., 2016). Outra característica importante do

Vivado HLS é que esta ferramenta pode ser aplicada a qualquer tipo de problema, e não apenas a tipos específicos de aplicações, como acontece com outras ferramentas de síntese em alto nível (NANE *et al*., 2016).

Devido a estas características optou-se para este experimento implementar os mesmos algoritmos em PON-HD 1.0 e em Vivado HLS de modo a verificar como cada uma destas ferramentas se comporta em relação a utilização de recursos da FPGA e em relação a performance.

Um trecho do código em C++ para o circuito do *driver* para *display* de 7 segmentos bem como a interface do Vivado HLS são apresentados na Figura 42.

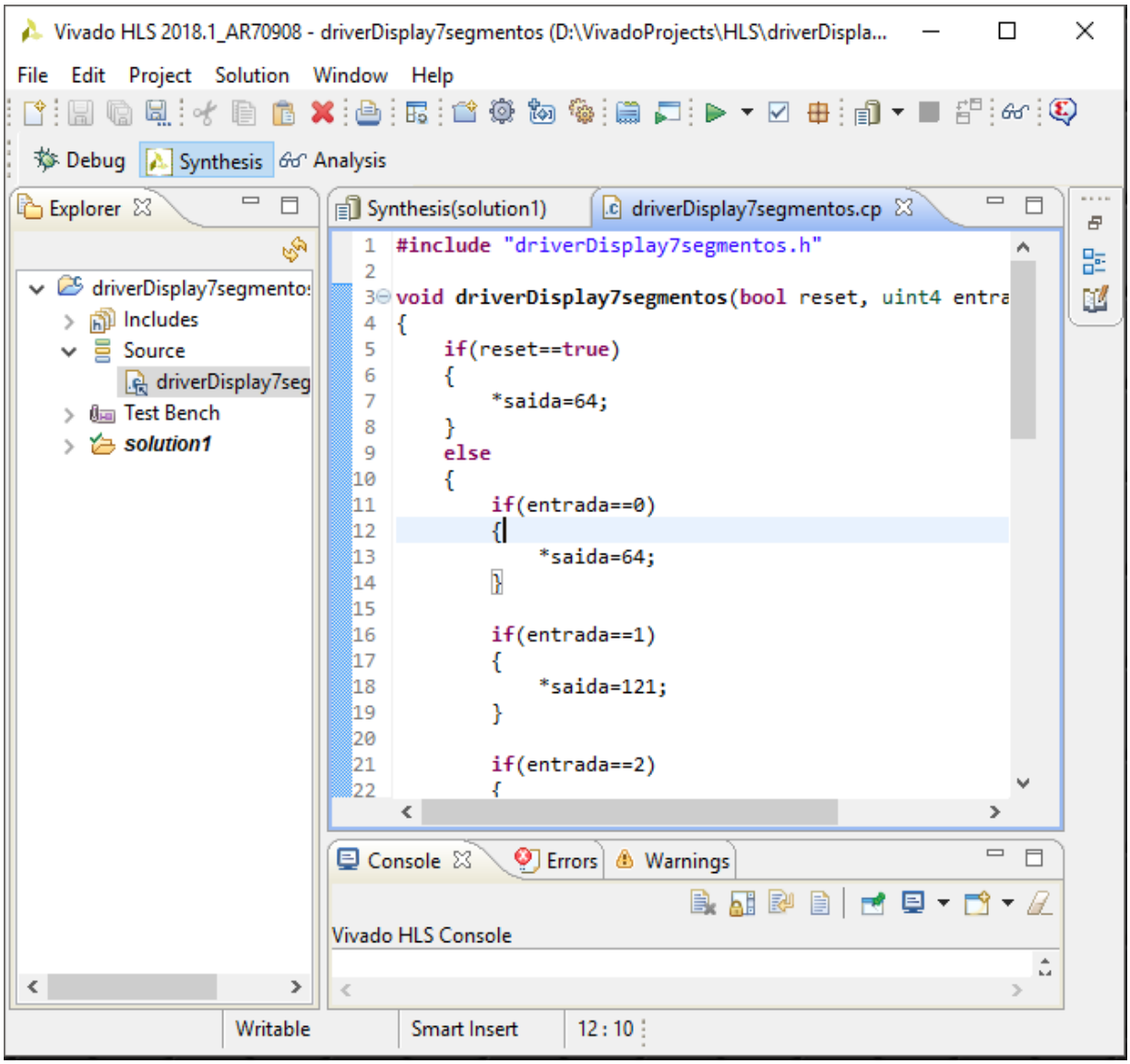

Figura 42 - Trecho do código em C++ do experimento 2.

Fonte: Autoria própria.

O código C++ completo para este experimento pode ser visualizado no Apêndice H.

A ferramenta Vivado HLS permite que se realize testes e simulações diretamente no código C++. Assim as primeiras simulações e testes de funcionamento foram realizadas ainda no código em C++, e a verificação do funcionamento do algoritmo do *driver* para *display* de 7 segmentos se deu antes mesmo da geração do código VHDL.

Uma vez verificado o funcionamento do código C++ do experimento a geração do código VHDL se deu através da própria ferramenta Vivado HLS.

Desta etapa em diante o procedimento é muito semelhante ao executado para o código VHDL gerado com o PON-HD 1.0. O código VHDL gerado pelo Vivado HLS foi inspecionado e estando de acordo com o esperado prosseguiu-se para o processo de síntese do circuito. O código VHDL completo pode ser observado no Apêndice I.

O código VHDL gerado pelo Vivado HLS foi então sintetizado pela ferramenta Vivado 2018.1. A FPGA escolhida para a síntese e para as simulações é, assim como no caso anterior, do modelo XA7A12TCPG238-2L da família Artix-7 fabricada pela Xilinx.

Para verificar o funcionamento do circuito foram realizados alguns testes. Inicialmente verificou-se o esquemático (RTL) do circuito gerado pela ferramenta de síntese. Subsequentemente uma análise mais profunda do funcionamento do circuito foi realizada através de simulações e verificações funcionais.

A Figura 43 apresenta o bloco de circuito gerado pela síntese do código VHDL para o *driver* de *display* de 7 segmentos desta implementação.

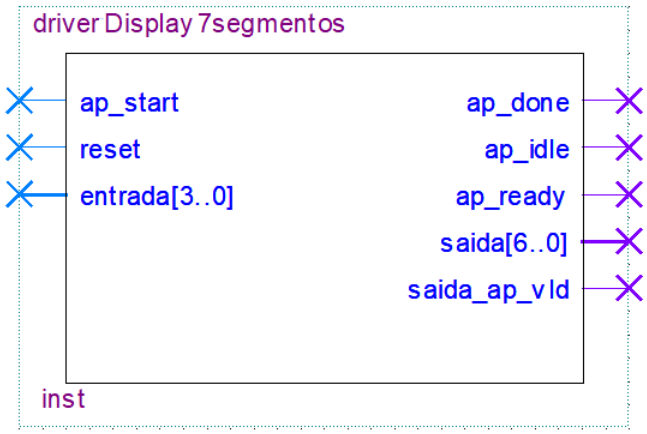

Figura 43 - Bloco gerado pela síntese do código C++ do experimento 2.

Fonte: Autoria própria.

Na figura, é possível observar os sinais de entrada de dados com seus 4 bits e de *reset.* Também é possível observar o sinal de saída com seus 7 bits. A ferramenta Vivado HLS acrescenta automaticamente ao circuito um conjunto de entradas e saídas, as quais são identificadas na figura pelas letras "ap_". Estes sinais servem para interligar este circuito com outros circuitos do projeto.

É importante destacar que na implementação do *driver* para *display* de 7 segmentos a ferramenta Vivado HLS, diferentemente do PON-HD 1.0, não utilizou "*clock*". Isso demonstra que o circuito gerado é puramente combinacional e não sequencial.

4.2.2 Resultados do Experimento 2 (Driver para display de 7 segmentos)

A composição deste experimento, apesar de simples, permitiu comparar a forma de gerar os circuitos empregada pelo PON-HD 1.0 e pelo Vivado HLS. Ambas as implementações executaram a tarefa de decodificar os dados para *displays* de 7 segmentos de forma correta. As estratégias utilizadas, no entanto, foram bastante distintas.

No PON-HD 1.0, devido a estrutura do próprio paradigma, as entradas e saídas são naturalmente registradas e armazenadas nos *Attributes*. O Vivado HLS por sua vez não apresenta esta característica e, portanto, gerou um circuito completamente combinacional.

A Tabela 8 apresenta as principais características de performance e utilização de recursos da FPGA apresentado por cada uma das implementações.

Tabela 8 - Características de implementação do experimento 2.

	LUTs	FFs	Ciclos de *clock*	Máxima Frequência
PON-HD 1.0	4	12	2 ciclos de *clock*	95,67 MHz
Vivado HLS	5	0	Não aplicável	Não aplicável
VHDL	4	0	Não aplicável	Não aplicável

Fonte: Autoria própria.

Observando esta tabela, é possível notar que a nomenclatura utilizada nos elementos de *hardware* difere do que foi utilizado no experimento anterior. Isso se deve a mudança de ferramenta de síntese utilizada. No experimento anterior foi utilizado o Quartus II, enquanto que neste experimento foi utilizado o Vidado. Cada fabricante trata de uma forma diferente a nomenclatura dos elementos de *hardware*, mas é possível estabelecer uma relação. Na Tabela

8, os chamados LUTs são as *LookUp Tables*, e estão relacionados aos elementos lógicos. Os FFs são os *Flip Flops* e são equivalentes aos registradores. Com relação a latência apresentada na tabela, representa o número de ciclos de *clock* necessários para que uma mudança na entrada reflita em uma mudança na saída.

Isso posto, é possível observar que a implementação em PON-HD 1.0 utilizou apenas 4 LUTs, enquanto a implementação em Vivado HLS utilizou 5. Contudo, a implementação em Vivado HLS não demandou registradores, enquanto que o PON-HD 1.0 necessitou de 12 registradores. Esta diferença é facilmente explicada se observarmos a estrutura do PON-HD 1.0. No PON-HD 1.0, as entradas e saídas são armazenadas nos *Atributes*. Assim temos 4 bits na entrada de dados, 1 bit na entrada de *reset* e 7 bits na saída, totalizando os 12 registradores apresentados na tabela.

Outra característica do PON-HD 1.0, que fica saliente nos dados da Tabela 8, é a latência de 2 ciclos de *clock*. Um ciclo é necessário para armazenar os dados de entrada nos *Attributes* de entrada e outro ciclo é necessário para armazenar os dados de saída nos *Attributes* de saída. Observa-se com este experimento que o PON-HD 1.0 apresenta performance inferior ao Vivado HLS para circuitos muito simples, que podem ser representados por circuitos puramente combinacionais.

A implementação em Vivado HLS não apresenta uma máxima frequência de operação por não possuir um sinal de *clock*. Desta forma este tipo de informação não se aplica.

É importante salientar que neste experimento os dados não necessitavam ficar armazenados no *driver*. Se a aplicação em questão necessitasse armazenar os dados, os resultados seriam diferentes. Neste caso, a implementação em Vivado HLS também necessitaria de registradores e consequentemente de um sinal de *clock*.

A título de comparação foram também incluídos na Tabela 8 dados referentes a uma implementação do experimento diretamente em VHDL. Nesta implementação foram necessários apenas 4 LUTs, e o circuito gerado é completamente combinacional.

4.2.3 Conclusões do Experimento 2 (Driver para display de 7 segmentos)

O segundo experimento, por sua vez, comparou circuitos gerados com o PON-HD 1.0 com circuitos gerados com a ferramenta de síntese em alto nível Vivado HLS. Para tanto, foi desenvolvido um driver para display de 7 segmentos. Este algoritmo permitiu comparar ambas as ferramentas na implementação de um circuito combinacional, situação a princípio

desfavorável para o PON-HD 1.0. Pode-se concluir com este experimento que o PON-HD 1.0 apresenta algumas limitações de performance, principalmente para circuitos predominantemente combinacionais, isto devido a própria estrutura do PON. Porém, de forma geral, a utilização do ferramental PON-HD 1.0 não gera degradação de performance tão significativa a ponto de comprometer sua utilização como ferramenta de síntese em alto nível.

4.3 Experimento 3: Contador Decimal

O terceiro experimento trata de um contador decimal. Este experimento é um pouco mais complexo que os anteriores, dado que é composto de vários contadores e decodificadores.

4.3.1 Descrição do Experimento 3 (Contador Decimal)

Este experimento foi pensado com os objetivos de validar o Simulador PON-HD 1.0, verificar a conectividade entre o circuito descrito em LingPON-HD e seus periféricos e validar o VHDL gerado no âmbito do PON-HD 1.0. O circuito escolhido para esta tarefa foi um contador decimal de três dígitos.

Para implementar um contador decimal de três dígitos em PON-HD 1.0, um conjunto de contadores e decodificadores foi arranjado de forma a compô-lo. O diagrama de blocos da Figura 44 apresenta os principais elementos deste circuito.

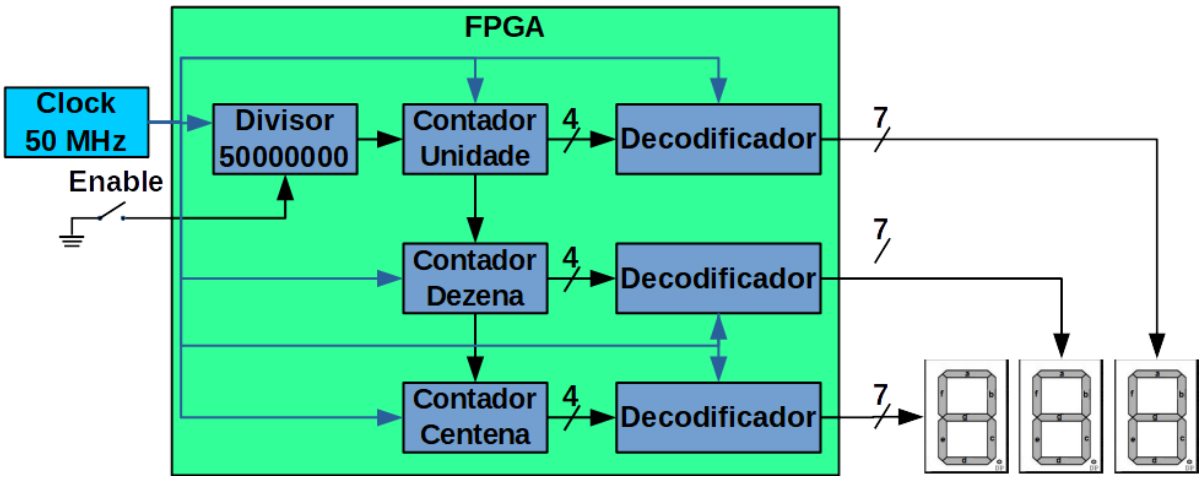

Figura 44 - Diagrama de blocos do contador decimal.

Fonte: Autoria própria.

O funcionamento é simples: a base da contagem é um sinal de *clock* de 50 MHz que é dividido por 50000000 para obter um sinal de 1 Hz. Este sinal é aplicado a um contador que é responsável por contar as unidades. Quando o contador de unidades atinge o valor 10 ele retorna a zero e incrementa o contador de dezenas. O mesmo acontece com relação aos contadores de dezenas e centenas.

A saída destes três contadores passa por decodificadores e é aplicada a *displays* de sete segmentos, no qual a contagem pode ser visualizada. Por fim, a contagem pode ser ativada ou desativada através de uma entrada de "*Enable*" que habilita ou desabilita o divisor de *clock*.

A Figura 45 apresenta um diagrama que ilustra o fluxo do experimento. Assim como no diagrama do experimento anterior, neste diagrama é possível observar as várias etapas de desenvolvimento e de verificação.

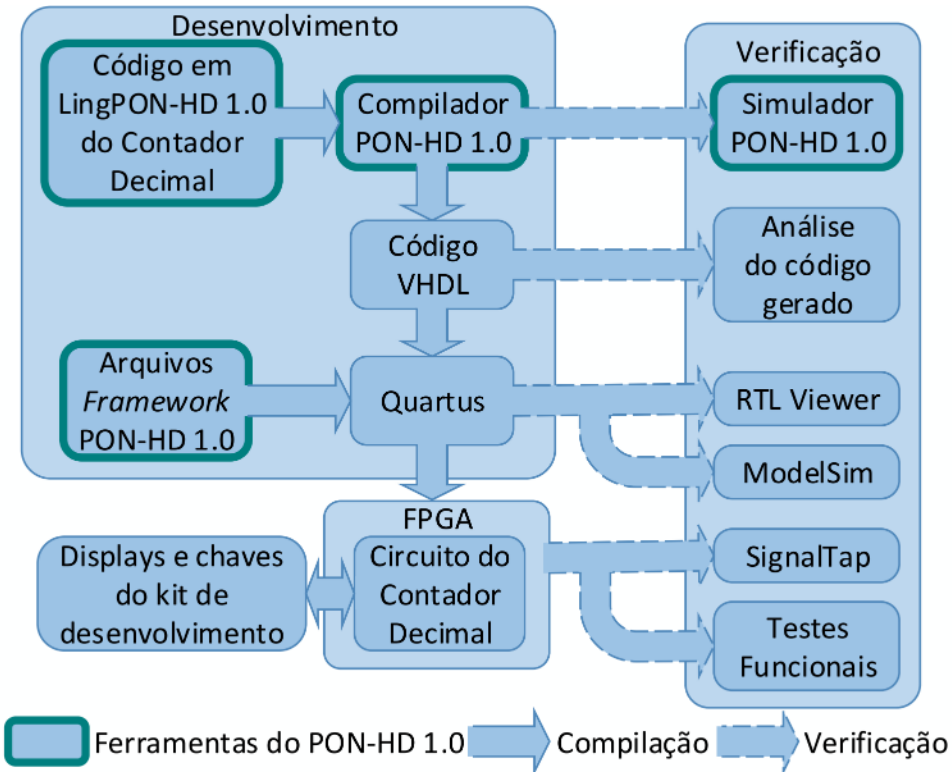

Figura 45 - Diagrama de fluxo do experimento 2.

Fonte: Autoria própria.

Este experimento foi realizado para validar o simulador PON-HD 1.0 e verificar de forma mais profunda o código VHDL gerado, isto através de uma análise detalhada de cada componente instanciado no código VHDL. O experimento serviu também para verificar como os circuitos desenvolvidos com o PON-HD 1.0 se comportam em relação as entradas e saídas.

As conexões do circuito com o *hardware* necessário para o experimento, como *displays*, circuito de *clock* e chave de *enable* são um fator importante e que deve ser testado com cuidado.

O experimento foi implementado em LingPON-HD 1.0 e necessitou de 8 *Attributes*, 35 *Premises*, 38 *Rules*, 38 *Methods*. A Figura 46 apresenta um trecho do programa em LingPON-HD 1.0 deste experimento.

```
fbe fbeContadorDecimal
        attributes
                integer atContClock 0
                integer atUnidade 0
                integer atDezena 0
                integer atCentena 0
                boolean atLigado false
        end_attributes
        methods
                method mtZera(atContClock = 0)
                method mtIncrementaCont(atContClock = atContClock + 1)
                method mtIncUnid(atUnidade = atUnidade + 1)
                method mtIncDezena(atDezena = atDezena + 1)
                method mtIncCentena(atCentena = atCentena + 1)
                method mtZeraUnidade(atUnidade = 0)
                method mtZeraDezena(atDezena = 0)
                method mtZeraCentena(atCentena = 0)
        end_methods
end_fbe
```

Figura 46 - Trecho do programa do contador decimal.

Fonte: Autoria própria.

Neste trecho pode-se observar a declaração da FBE “fbeContadorDecimal”. O código completo em LingPON-HD 1.0 deste experimento pode ser visualizado no Apêndice J. Após compilado, o código foi verificado com o simulador PON-HD 1.0. Uma vez demostrado por simulação o seu funcionamento, o mesmo foi sintetizado e gravado em uma FPGA. O código completo em VHDL gerado da compilação do código em LingPON-HD 1.0 deste experimento pode ser visualizado no Apêndice K.

O *hardware* escolhido para o experimento foi um kit de desenvolvimento DE1 fabricada pela Terasic. Este Kit é apropriado pois possui todos os elementos necessários, como sinal de *clock*, *displays* e chaves. A FPGA disponível neste kit é uma *Cyclone II* fabricada pela Altera. A compilação do código VHDL foi realizada com o *software* Quartus II, também fabricado pela Altera.

Para verificar o funcionamento do circuito foram realizados alguns testes. Inicialmente verificou-se o resultado da contagem nos *displays*, conforme o circuito recebia o comando de ativação. Subsequentemente uma análise mais profunda do funcionamento do circuito foi realizada através de simulações com o *software* de simulação "*ModelSim*" e, posteriormente, foram analisados os sinais no interior da FPGA com a ferramenta "*SignalTap II*". A Figura 47 apresenta alguns sinais adquiridos durante o experimento do contador decimal.

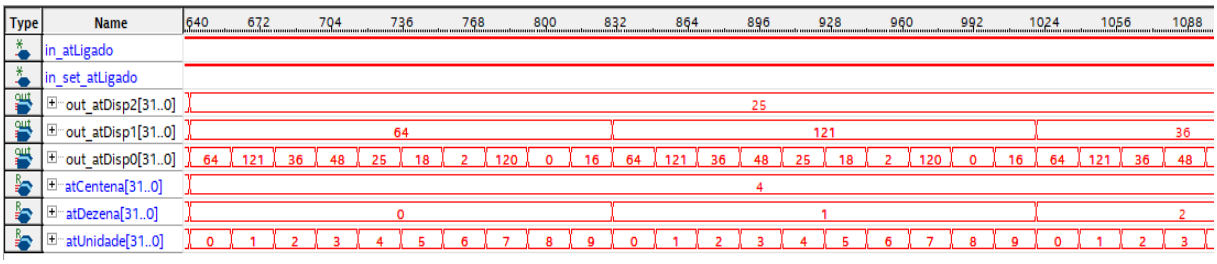

Figura 47 - Sinais adquiridos durante o experimento do contador decimal.

Fonte: Autoria própria.

Também foi realizada uma análise minuciosa do circuito gerado após a compilação do código VHDL observando o diagrama do circuito com a ajuda da ferramenta "RTL Viewer". A análise do diagrama RTL (*Register Transfer Level*) do circuito é particularmente importante pois permite verificar se o programa em LingPON-HD 1.0 foi corretamente interpretado e implementado pelas ferramentas do PON-HD 1.0. Esta verificação é realizada comparando se cada elemento descrito no programa em LingPON-HD 1.0 tem seu correspondente corretamente implementado no diagrama RTL. O diagrama não foi reproduzido aqui devido a seu tamanho realmente grande.

4.3.2 Resultados do Experimento 3 (Contador decimal)

Este experimento, por se tratar de um arranjo de vários contadores e decodificadores, como apresentado no diagrama de blocos da Figura 44, permitiu verificar o correto funcionamento dos circuitos sintetizados com o PON-HD 1.0. O funcionamento do circuito foi exatamente como esperado demostrando estabilidade mesmo após longos períodos de operação. O circuito ficou em operação por seis horas ininterruptas sem mostrar variações em seu funcionamento.

Este experimento serviu também para demostrar o correto funcionamento das ferramentas do PON-HD 1.0, principalmente o compilador e o simulador. O comportamento do circuito sintetizado com o PON-HD 1.0 em relação a suas entradas e saídas também foi verificado neste experimento. Várias conexões de *hardware* foram necessárias para interconectar o circuito com os *displays*, com o circuito de *clock* e com a chave de *enable*. Ainda assim os resultados demostraram que os circuitos sintetizados com o PON-HD 1.0 são facilmente conectados a outros componentes de *hardware*.

Uma análise mais profunda, verificando o funcionamento de cada um dos contadores e decodificadores do circuito, foi realizada através de simulações com o *software* de simulação "*ModelSim*". O mesmo procedimento foi realizado no interior da FPGA com a ferramenta "*SignalTap II*". Estas analises demostraram que o circuito executou corretamente o que foi especificado no programa em LingPON-HD 1.0, validando assim o código VHDL gerado, bem como o seu processo de geração.

A Tabela 9 apresenta as características de implementação do circuito deste experimento. A título de comparação o mesmo algoritmo foi implementado também diretamente em VHDL. A Tabela 9 também apresenta as características desta implementação.

Tabela 9 - Características de implementação do experimento 3.

	LUTs	FFs	Máxima Frequência
PON-HD 1.0	42	64	95,30 MHz
VHDL	23	14	98.91 MHz

Fonte: Autoria própria.

Como esperado a implementação realizada diretamente em VHDL consumiu menos recursos da FPGA, pois devido as características do PON-HD e sua necessidade de registrar todos os dados nos *Attibutes* é natural a utilização de mais componentes de *hardware*. Com relação a máxima velocidade de *clock* não ouve diferenças significativas, com a implementação em PON-HD sendo um pouco mais lenta.

Um estudo detalhado do circuito gerado após a compilação do código VHDL através da ferramenta "RTL Viewer" demostrou que os componentes do PON foram corretamente implementados no *hardware*, demonstrando o correto funcionamento do compilador.

4.3.3 Conclusões do Experimento 3 (Contador)

O terceiro experimento é um circuito contador decimal de três dígitos. Neste experimento é explorada a conexão dos circuitos gerados com o PON-HD 1.0 com outros elementos de *hardware*, como *displays* de sete segmentos e chaves. As características do experimento permitiram ainda verificar o correto funcionamento dos circuitos sintetizados com o *framework* PON-HD 1.0 e o correto funcionamento das ferramentas do PON-HD 1.0, principalmente o compilador e o simulador. Pode-se concluir com exte experimento que o ferramental PON-HD 1.0 se mostra funcional para desenvolvimento de *hardware* digital em alto nível. Conclui-se também que o simulador permite testar de forma rápida e fácil aplicações descritas em LingPON-HD 1.0 sem a necessidade sintetizar os respetivos circuitos. O código VHDL gerado é perfeitamente funcional, executando corretamente o algoritmo implementado em LingPON-HD 1.0. Outra conclusão importante deste experimento é que os circuitos desenvolvidos se conectam com facilidade aos outros circuitos que compõe o *hardware*.

4.4 Experimento 4: Calculador de Média

Este experimento foi pensado para ser um circuito sequencial, necessitando de várias etapas para executar o algoritmo proposto. Desta forma, o experimento realiza comparações entre o PON-HD 1.0 e a ferramenta Vivado HLS, de uma forma substancialmente diferente dos outros experimentos. São realizadas comparações de performance e de utilização pelo circuito de recursos na FPGA.

4.4.1 Descrição do Experimento 4 (Calculador de Média)

O Objetivo deste experimento é realizar um comparativo de performance entre os circuitos gerados pelo PON-HD 1.0 e pela ferramenta vivado HLS, para um circuito sequencial.

O experimento realiza o cálculo da média de um conjunto de 10 valores inteiros de 16 bits, localizados em uma memória externa ao circuito. O diagrama de blocos da Figura 48 apresenta a topologia do circuito proposto para o experimento.

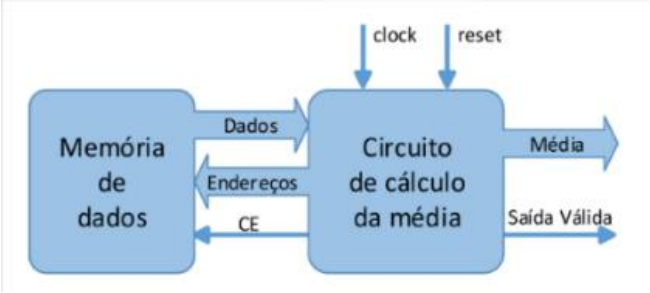

Figura 48 - Diagrama de blocos do experimento 4.

Fonte: Autoria própria.

Observando o diagrama, nota-se que o circuito proposto é responsável por gerar os sinais de endereço e de CE (*Chip Enable*) para uma memória externa, de onde será realizada a leitura dos valores numéricos (dados). Estes dados são compostos por 10 valores inteiros de 16 bits, os quais devem ser somados. O resultado desta soma deve ser dividido por 10, de forma a calcular a média destes valores. Finalmente, o resultado deve ser apresentado pelo circuito em sua saída. Adicionalmente, uma saída chamada "Saída Válida" deve ser ativada para indicar que o valor na saída do circuito é válido. O circuito ainda possui uma entrada de *clock* e uma entrada de *reset*, que servem para controlar seu funcionamento.

A Figura 49 apresenta um diagrama que ilustra o fluxo do experimento.

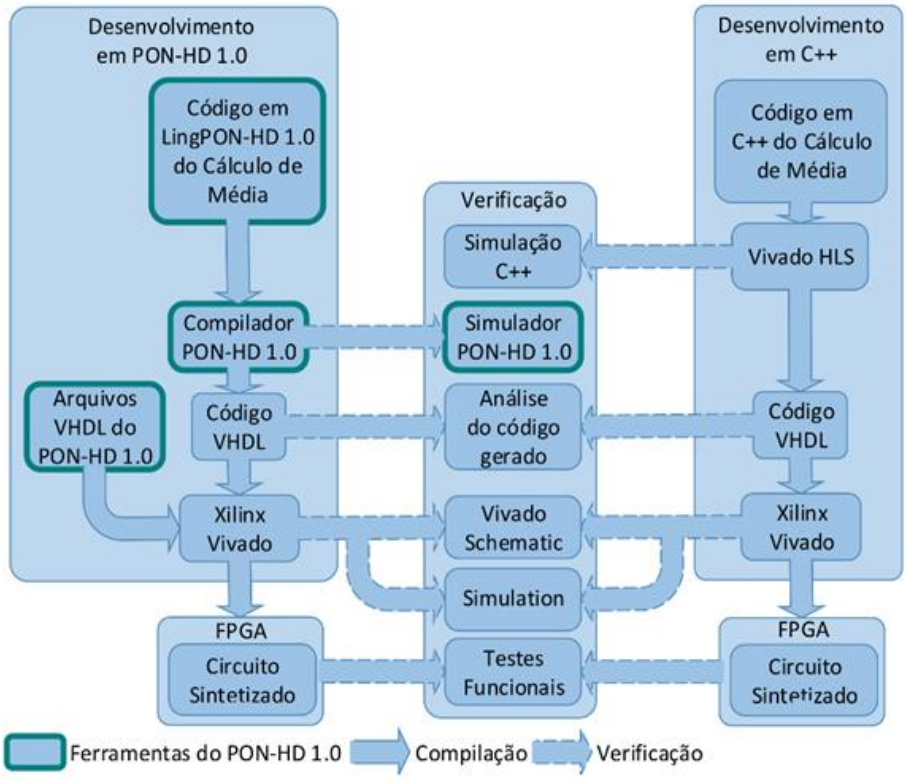

Figura 49 - Diagrama de fluxo do experimento 4.

Fonte: Autoria própria.

Neste diagrama, é possível observar as várias etapas de desenvolvimento e de verificação realizadas neste experimento.

4.4.1.1 Desenvolvimento do circuito de cálculo de média em PON-HD 1.0

O primeiro passo para o desenvolvimento do circuito de cálculo de média em PON-HD 1.0 foi o desenvolvimento do código em LingPON-HD 1.0. A Figura 50 apresenta um trecho do código em LingPON-HD 1.0 do circuito de cálculo de média.

```
fbe fbeCalculaMedia
        attributes
                integer atDataIn 0 // entrada de dados
                integer atTemp 0 // armazena temporariamente o resultado
                integer atRes 0 // saída que apresenta a média
                integer atAddr 0 // endereço para a leitura dos dados de entrada
                integer atStep 0 // controla as etapas de funcionamento do circuito
                boolean atCE false // sinal de leitura dos dados de entrada
                boolean atResVal false // saída que indica que o resultado é válido
                boolean atReset false // entrada de reset
        end_attributes
        methods
                method mtResetRes(atRes = 0)
                method mtResetStep(atStep = 0)
                method mtIncStep(atStep = atStep + 1)
                method mtResVal_0(atResVal = false)
                method mtResVal_1(atResVal = true)
                method mtAddr_0(atAddr = 0)
                method mtAddrInc(atAddr = atAddr + 1)
                method mtCE_0(atCE = false)
                method mtCE_1(atCE = true)
                method mtAtualRes(atRes = atTemp / 10) // atualiza Resultado
                method mtTempDataIn(atTemp = atDataIn)
                method mtAcumulaTemp(atTemp = atTemp + atDataIn)
        end_methods
end_fbe
⋮
rule rlReset
        condition
                subcondition SReset
                        premise prReset CM.atReset == true
                end_subcondition
        end_condition
        action
                instigation inResetRes CM.mtResetRes();
                instigation inResetAddr CM.mtAddr_0();
                instigation inResetStep CM.mtResetStep();
                instigation inResetResetCE CM.mtCE_0();
                instigation inResetResVal CM.mtResVal_0();
        end_action
end_rule
```

Figura 50 - Código em LingPON-HD 1.0 para o experimento 4.

Fonte: Autoria própria.

O código completo do programa em LingPON-HD 1.0 pode ser encontrado no Apêndice L. O código elaborado para este experimento é composto por 8 *Attributes*, 21 *Premises*, 21 *Rules*, 12 *Methods*.

O método escolhido para implementar o algoritmo de cálculo da média em LingPON-HD 1.0 foi o seguinte. Um *Attribute* chamado "atStep" foi criado para controlar as várias etapas de execução do algoritmo. A cada etapa este atribute é comparado pelas *Premises* de forma que apenas um *Method* é ativado por vez. Em cada *Method* ativado a etapa relacionada é executada. São exemplos de etapas executadas pelos *Methods*: gerar o endereço e ativar a memória, ler um novo valor da memória, somar os valores, ativar a próxima etapa e assim por diante.

Seguindo o sistema de prioridades e resolução de conflitos implementado no PON-HD 1.0 o sinal de *reset* tem prioridade sobre as demais etapas do processo, ativando um *Method* que por sua vez leva todos os *Attributes* a seu estado inicial.

Uma vez concluído o desenvolvimento do código em LingPON-HD 1.0, o mesmo foi então compilado com o compilador PON-HD 1.0 resultando em um arquivo VHDL. Também foi utilizado nesta etapa o simulador PON-HD 1.0 para verificar se o programa implementado em LingPON-HD 1.0 funcionou corretamente.

O código VHDL gerado foi inspecionado e, estando de acordo com o esperado, prosseguiu-se para o processo de síntese do circuito. O código VHDL completo gerado pelo compilador PON-HD 1.0 pode ser observado no Apêndice M.

Utilizou-se a ferramenta Vivado 2018.1 para a síntese do código VHDL gerado pelo PON-HD 1.0. A FPGA escolhida para as simulações é do modelo XA7A12TCPG238-2L da família Artix-7 fabricada pela Xilinx.

Para verificar o funcionamento do circuito foram realizados alguns testes. Inicialmente, verificou-se o esquemático (RTL) do circuito gerado pela ferramenta de síntese. Na sequência foi realizada uma análise mais profunda do funcionamento do circuito através de simulações. Na sequência, vários conjuntos de dados de entrada foram fornecidos ao circuito gerado e seu correto funcionamento foi verificado. A Figura 51 apresenta o bloco de circuito gerado pela síntese do código VHDL para o cálculo de média, resultante do programa em LingPON-HD 1.0.

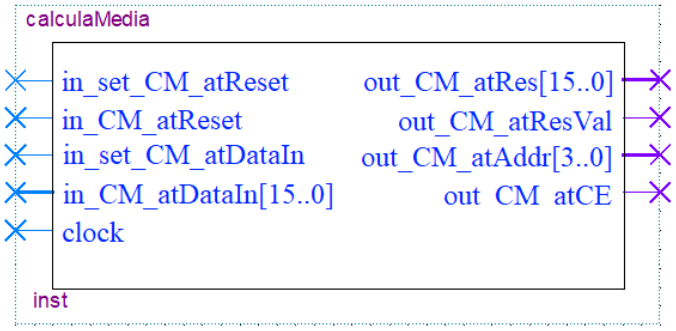

Figura 51 - Bloco gerado pela síntese do código do experimento 4.

Fonte: Autoria própria.

Nesta figura é possível observar a entrada de *clock*, a entrada de *reset*, o seu respectivo sinal de atribuição (set), a entrada de dados de 16 bits e o seu respectivo sinal de atribuição. Também é possível observar a saída de endereços de 4 bits, a saída que apresenta o resultado de 16 bits, a saída de ativação da memória e a saída que indica um resultado válido. Estes sinais de atribuição não estavam previstos no diagrama da Figura 48, mas são implementados automaticamente pelo PON-HD, de forma a gerenciar o armazenamento de dados nos *Attributes*.

Para realizar as comparações propostas, o mesmo circuito foi implementado na ferramenta Vivado HLS. A seção a seguir apresenta esta implementação.

4.4.1.2 Desenvolvimento do circuito de cálculo de média em Vivado HLS

A ferramenta utilizada foi o Vivado HLS versão 2018.1. O algoritmo de cálculo de média foi implementado nesta ferramenta em C++. O código em C++ para este experimento é bastante simples, pois todos os sinais de interface são gerados automaticamente pela ferramenta. A Figura 52 apresenta o código em C++ para o cálculo de média.

As primeiras simulações e testes de funcionamento foram realizadas ainda no código em C++, e a verificação do funcionamento do algoritmo de cálculo de média se deu antes mesmo da geração do código VHDL.

Uma vez verificado o funcionamento do código C++ do experimento, a geração do código VHDL se deu através da própria ferramenta Vivado HLS.

Assim como aconteceu para o PON-HD 1.0, o código VHDL gerado pelo Vivado HLS foi inspecionado e, estando de acordo com o esperado, prosseguiu-se para o processo de síntese do circuito. O código VHDL completo pode ser observado no Apêndice N.

```
1  #include "calculaMedia.h"
2
3  void calculaMedia(int16 v[10], int16 *res)
4  {
5      uint4 cont;
6      *res=0;
7      for(cont=0; cont<10; cont++)
8      {
9          *res=*res+v[cont];
10     }
11     *res=*res/10;
12 }
13
```

Figura 52 - Código em C++ do experimento 4.

Fonte: Autoria própria.

O código VHDL gerado pelo Vivado HLS foi então sintetizado pela ferramenta Vivado 2018.1. A FPGA escolhida para a síntese e para as simulações é, assim como no caso anterior, do modelo XA7A12TCPG238-2L da família Artix-7 fabricada pela Xilinx.

Para verificar o funcionamento do circuito foram realizados alguns testes. Inicialmente verificou-se o esquemático (RTL) do circuito gerado pela ferramenta de síntese. Na sequência, vários conjuntos de dados de entrada foram fornecidos ao circuito, de forma a verificar seu correto funcionamento.

A Figura 53 apresenta o bloco de circuito gerado pela síntese do código VHDL para o cálculo de média desta implementação.

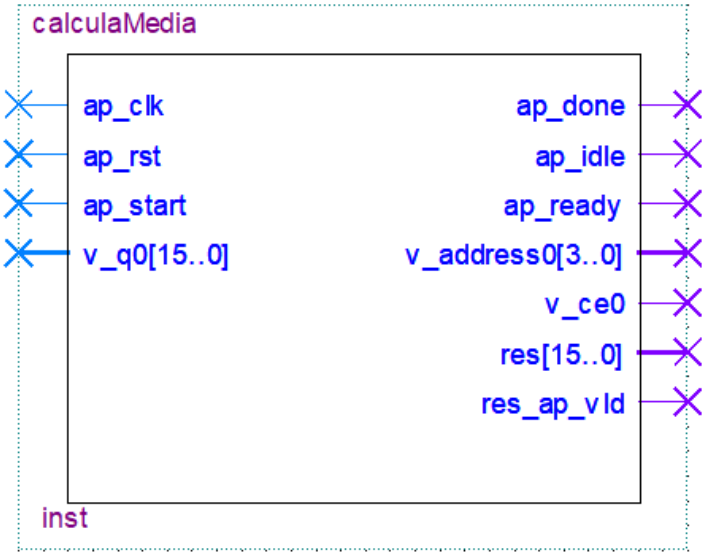

Figura 53 - Bloco gerado pela síntese do código C++ do experimento 4.

Fonte: Autoria própria.

Na Figura 53, é possível observar o sinal de entrada de dados de16 bits, o sinal de *reset*, o sinal de *clock* e um sinal de *start*, todos gerados automaticamente pelo Vivado HLS. Também é possível observar o sinal de saída de 16 bits, o sinal de endereçamento da memória de 4 bits, o sinal de ativação da memória CE, a sinal de saída válida, além dos sinais de *done*, *idle*, *ready*, também gerados automaticamente pela ferramenta em questão.

Assim como nos demais experimentos, houve uma grande preocupação em tornar as duas implementações o mais semelhante possível, de forma a tornar as comparações mais precisas. Isto se deu através da implementação de algoritmos com comportamento semelhante.

4.4.2 Resultados do Experimento 4 (Calculador de Média)

Ambas as implementações deste experimento utilizam sinal de *clock*, ou seja, são circuitos sequenciais. Desta forma, é possível realizar uma comparação entre os circuitos gerados pelo PON-HD 1.0 e pelo Vivado HLS para uma aplicação sequencial.

Ambas as implementações executaram a tarefa de calcular a média de um conjunto de 10 valores inteiros de 16 bits de forma correta. As estratégias utilizadas, no entanto, foram ligeiramente diferentes. Estas diferenças podem ser observadas nos dados apresentados na Tabela 10. Esta tabela demostra as principais características de performance e utilização de recursos da FPGA apresentado por cada uma das implementações.

Tabela 10 - Características de implementação do experimento 4.

	LUT	FF	DSP	Frequência Máxima	ciclos de *clock*
PON-HD	230	58	0	35.71 MHz	20 ciclos de *clock*
Vivado HLS	88	27	1	38.46 MHz	21 ciclos de *clock*
VHDL	121	43	0	53.38 MHz	20 ciclos de *clock*

Fonte: Autoria própria.

Adotou-se na tabela o padrão de nomenclatura da Xilinx. Os dados que compõem a tabela foram obtidos com a ferramenta Vivado, após a síntese do circuito.

Na Tabela 10 é possível observar que a implementação em PON-HD 1.0 utilizou 230 LUTs, enquanto a implementação em Vivado HLS utilizou apenas 88. Esta diferença se dá, entre outras coisas, pela utilização por parte da implementação em Vivado HLS de um bloco

DSP da FPGA, o que reduz significativamente o uso de LUTs necessário para a realização das operações matemáticas.

Com relação a utilização de registradores (FFs), a implementação em Vivado HLS utiliza 27 registradores, enquanto que o PON-HD 1.0 utilizou 58 registradores. Assim, como no experimento anterior, esta diferença é explicada pela estrutura do PON-HD 1.0, onde os resultados intermediários são armazenados nos *Attributes*, e consequentemente nos registradores, além da necessidade de utilizar-se mais *hardware* nas operações matemáticas, uma vez que não é utilizado um bloco DSP.

Após uma análise mais aprofundada do esquemático (RTL) do circuito gerado após a síntese do código VHDL da implementação em PON-HD 1.0 foi possível observar que o método responsável pela operação aritmética de divisão utilizou 148 LUTs. Fazendo uma análise semelhante no esquemático da implementação realizada com o Vivado HLS observa-se que a mesma operação é realizada por um bloco “DSP48” e alguns circuitos combinacionais.

Levando então em consideração as LUTs utilizadas pela implementação em PON-HD 1.0 para realizar a operação de divisão, o consumo de LUTs pelo restante do circuito é bastante semelhante ao consumo de LUTs da implementação realizada em Vivado HLS.

Como as duas implementações utilizam sinal de *clock*, é possível estimar a máxima frequência de operação de cada uma delas. Neste critério, a implementação em Vivado HLS leva uma pequena vantagem, com uma diferença de 38.46MHz para 35.71MHz.

Já com relação a latência, ou seja, com relação ao número de ciclos de *clock* necessários para concluir as operações, a implementação em PON-HD 1.0 leva uma pequena vantagem, de 20 ciclos, contra 21 ciclos de clock requeridos pela síntese do Vivado HLS.

Esta característica explica, em parte, a frequência mais baixa de operação, pois a implementação em PON-HD 1.0 tem que realizar mais operações por ciclo de *clock*. A utilização do bloco DSP também favorece a implementação em Vivado HLS em relação a máxima frequência de operação, pois o bloco DSP é especializando em cálculos matemáticos, sendo mais rápido nesta tarefa do que um circuito genérico implementado para este fim.

Um resultado interessante aparece considerando-se a máxima frequência de operação e a latência ao mesmo tempo. Neste caso, uma implementação tem maior velocidade de *clock* e necessita de mais ciclos para executar as operações, enquanto que a outra possui menor velocidade, mas necessita de menos ciclos para executar as operações. Assim, para a implementação em PON-HD 1.0 temos 20 ciclos de 28 ns (35.71MHz) o que resulta em um tempo total de execução de 560ns. Para a implementação em Vivado HLS, por sua vez, temos 21 ciclos de 26 ns (38.46MHz) o que resulta em um tempo total de execução de 546ns.

Comparativamente, a implementação em Vivado HLS é apenas 14 ns mais rápida que a implementação em PON-HD 1.0 na conclusão das operações, isso utilizando-se um bloco DSP de *hardware* dedicado nas operações matemáticas.

A título de comparação foi também realizada uma implementação deste algoritmo diretamente em VHDL. A Tabela 10 apresenta os resultados encontrados para esta implementação. Observando estes dados nota-se que a implementação realizada diretamente em VHDL é mais rápida que as implementações em PON-HD e Vivado HLS. Também é possível observar que não é utilizado nenhum bloco DSP nesta implementação.

4.4.3 Conclusões do Experimento 4 (Calculador de Média)

O algoritmo proposto permitiu comparar o ferramental PON-HD 1.0 com o Vivado HLS na implementação de um circuito majoritariamente sequencial. É interessante visualizar como as ferramentas mesmo utilizando estratégias de implementação bastante diferentes obtiveram uma performance bastante semelhante. Assim, pode-se concluir que a performance dos circuitos implementados com o ferramental PON-HD 1.0 pode ser considerada polo menos satisfatória. Com relação a frequência de *clock*, a latência e a utilização de recursos, quando comparadas as implementações em PON-HD 1.0 e em outras ferramentas, não são observadas discrepâncias tão significativas a ponto de inviabilizar a utilização do PON-HD 1.0 como ferramenta de desenvolvimento de circuitos em *hardware* digital, ao menos para este experimento.

4.5 Experimento 5: *Ordenador de Dados*

Com o objetivo de verificar a estabilidade e a escalabilidade dos circuitos sintetizados utilizando o *Framework* PON-HD 1.0 e comparar estas características com circuitos sintetizados em VHDL de forma tradicional, um experimento mais complexo foi elaborado no tocante a ordenação de dados. Este experimento, devido a sua complexidade e tamanho, permitiu também avaliar a performance do PON-HD 1.0.

4.5.1 Descrição do Experimento 5 (Ordenador de Dados)

O circuito proposto para este experimento é um ordenador de dados. A Figura 54 apresenta a topologia proposta para este circuito.

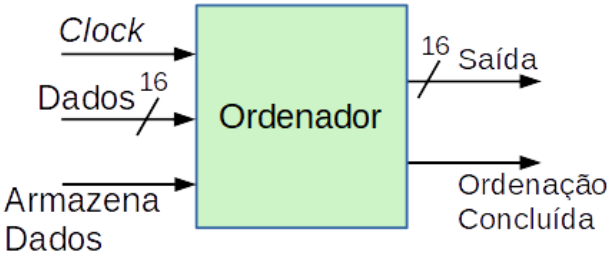

Figura 54 - Diagrama do ordenador de dados

Fonte: Autoria própria.

Distintamente dos experimentos anteriores, este experimento não foi completamente implementado em LingPON-HD 1.0, ou seja, os componentes do *Framework* PON-HD 1.0 foram utilizados em um programa VHDL, composto manualmente. Isto se deu porque para este experimento era necessário o uso de vetores, sendo que a Linguagem LingPON-HD 1.0, assim como a Linguagem LingPON de forma geral, ainda não permite a utilização de dados na forma de vetores. A linguagem LingPON-HD 1.0 foi utilizada para implementar a ordenação em um conjunto fixo de 16 elementos e sua implementação funcionou corretamente. Porém a implementação de um número maior de elementos se tornou pouco prática e desta forma uma abordagem alternativa foi adotada.

Assim sendo, para contornar esta limitação e tornar o código flexível em relação ao número de elementos no conjunto de dados da ordenação, o *Framework* PON-HD 1.0 foi utilizado sem seu compilador. Esta abordagem foi adotada pois assim é possível a utilização de instruções do tipo "*Generate*" que, no VHDL, permitem construir códigos parametrizáveis.

É importante enfatizar que a metodologia adotada serviu apenas para flexibilizar a implementação do algoritmo de ordenação no *hardware*. Esta metodologia também mostra que o *framework* PON-HD 1.0 é perfeitamente funcional e sua utilização sem o auxílio da LingPON-HD 1.0 não somente é possível, como é simples.

Como será apresentado a seguir, o experimento realiza a ordenação de um conjunto de dados. A utilização do "*Generate*" permitiu realizar experimentos com diferentes tamanhos de conjuntos de dados, apenas alterando alguns parâmetros. A possibilidade de alterar o tamanho

do conjunto de dados permitiu principalmente verificar a escalabilidade dos circuitos compostos com o PON-HD 1.0.

A Figura 55 apresenta um diagrama que ilustra o fluxo do experimento. Em comparação com os diagramas dos experimentos anteriores é possível notar a ausência dos blocos referentes ao programa em LingPON-HD 1.0, ao compilador PON-HD 1.0 e ao simulador PON-HD 1.0.

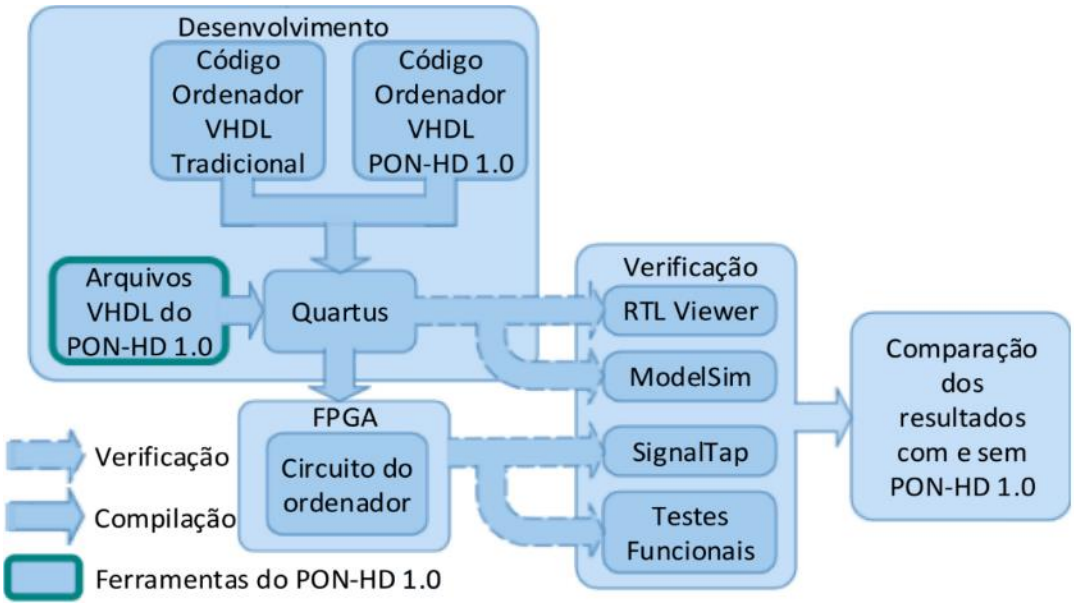

Figura 55 - Diagrama de fluxo do experimento 5.

Fonte: Autoria própria.

O experimento implementa um algoritmo paralelo de ordenação de dados chamado "*Odd Even Sort*" (MOZAFFARI, 2010; HEMATIAN *et al*., 2013). O algoritmo possui uma complexidade temporal $O(N^2)$ onde N é o número de elementos a ordenar (HEMATIAN *et al*., 2013).

A Figura 56 apresenta um diagrama de blocos que mostra como os componentes do PON-HD 1.0 foram arranjados para executar a ordenação de um conjunto de dados de quatro elementos. Cada elemento é um número inteiro de 16 bits, armazenado em um *Attribute*. Neste algoritmo, o objetivo é organizar os dados em ordem crescente. Para isso, todos os pares de *Attributes* com índices (ímpar, par) de elementos adjacentes no conjunto de dados são comparados e, se um par de *Attributes* está na ordem errada (se o valor do primeiro *Attribute* é maior do que o valor do segundo *Attribute*), os valores dos *Attributes* são trocados. No passo seguinte, os pares de *Attributes* com índices (par, ímpar) são tratados da mesma forma. As etapas (ímpar, par) e (par, ímpar) são repetidas de forma a ordenar todo o conjunto de dados.

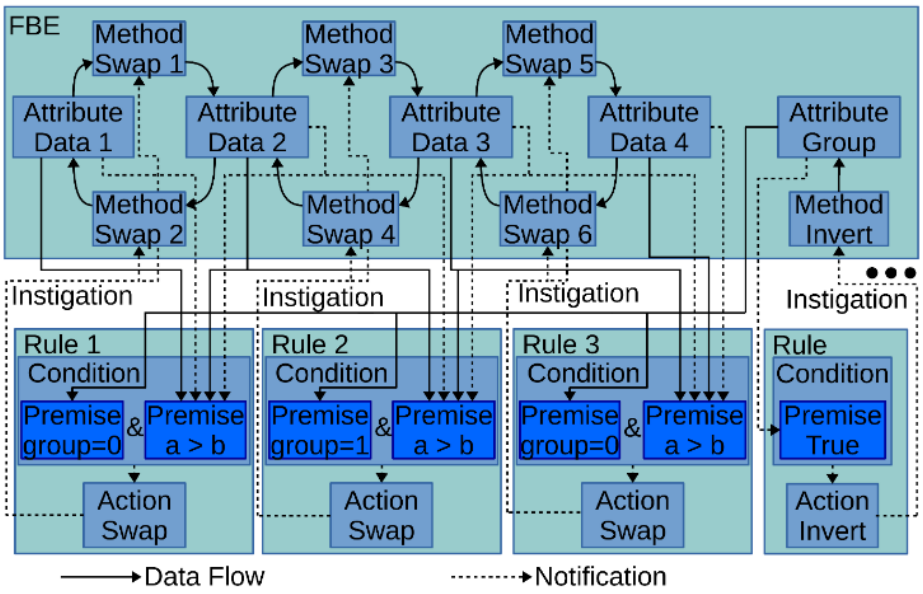

Figura 56 - Diagrama de blocos do ordenador PON.

Fonte: Autoria própria.

Ainda na Figura 56, é possível observar o princípio de funcionamento do circuito. Um *Attribute* é utilizado para determinar qual o grupo de elemento está sendo comparado, os pares ou os ímpares. A alternância entre o grupo dos elementos pares e o grupo dos elementos ímpares é realizada por um conjunto de *Premises* e *Methods*. Para cada par de *Attributes* contendo dados existe uma *Condition* composta por duas *Premises*.

A primeira *Premise* verifica se o valor do primeiro elemento do par de *Attributes* é maior que o valor do segundo (i.e., se os valores do par de *Attributes* está fora da ordem desejada). Por sua vez, a segunda *Premise* verifica se o grupo que está sendo comparado é o grupo desejado (i.e. grupo de *Attributes* par ou grupo de *Attributes* ímpar).

Se as duas *Premises* forem verdadeiras, deve então haver a troca dos valores dos elementos do par de *Attributes.* Para isso, os *Methods* responsáveis pela troca dos valores são executados. O ciclo de testes entre pares e ímpares vai se repetindo até que todos os dados estejam na posição correta e mais nenhuma troca seja necessária. O código VHDL completo construído utilizando o *Framework* PON-HD 1.0 para este experimento pode ser encontrado no Apêndice O.

Um algoritmo com o mesmo comportamento foi implementado em VHDL, sem a utilização dos componentes do *Framework* PON-HD 1.0. Esta implementação serviu como parâmetro nos testes de performance e das características de funcionamento. Foram tomados vários cuidados para tornar as comparações as mais fidedignas possível, principalmente em

relação ao comportamento do algoritmo. Apesar das diferenças de implementação a lógica de ordenação utilizado nos dois códigos é rigorosamente a mesma, apresentado comportamento semelhante em ambas as versões. O código completo construído diretamente em VHDL para este experimento pode ser encontrado no Apêndice P.

Na implementação do algoritmo de ordenação diretamente em VHDL houve uma preocupação em utilizar boas práticas de codificação, de forma a não prejudicar as comparações. O projeto de cada componente do código foi realizado com base em Pedroni (2010). As experiências anteriores do desenvolvedor (e proponente desta tese de doutorado) em VHDL, durante seu mestrado e posteriormente em sua vida profissional como engenheiro eletricista contribuíram para a qualidade do código desenvolvido.

Foram realizados experimentos com conjuntos de dados variando de quatro a três mil elementos. O limite de três mil elementos foi definido pela capacidade do *hardware* utilizado, neste caso, o número de elementos lógicos disponíveis na FPGA.

O *hardware* utilizado no experimento é uma placa DE2-115 fabricada pela TERASIC e utiliza uma FPGA Cyclone IV modelo EP4CE115, com 114480 elementos lógicos. Esta informação é relevante, pois isso foi o fator limitante no tamanho dos conjuntos de dados utilizados nos experimentos.

Foram utilizados nos experimentos conjuntos de dados aleatórios e conjuntos de dados inversamente ordenados. Para tornar as comparações fidedignas, os mesmos conjuntos de dados foram utilizados em ambas as implementações (com e sem o *Framework* PON-HD 1.0). Estes conjuntos de dados aleatórios foram previamente gerados por uma aplicação desenvolvida especificamente para este fim em linguagem C++ e armazenados nos arquivos.

A aplicação em *framework* PON-HD 1.0 foi constituída, conforme o tamanho do conjunto de dados, pelos seguintes componentes: de 5 a 3001 *Attributes,* de 4 a 3000 *Rules,* de 7 a 5999 *Premises* e de 7 a 5999 *Methods,* tudo isso limitado pelo tamanho da FPGA disponível, como já mencionado.

Durante os testes do experimento, vários conjuntos de dados foram aplicados aos dois circuitos. Inicialmente, verificou-se a correta execução do algoritmo nos dois códigos criados. Uma vez garantida a correta execução dos mesmos, foram levantados dados de performance, como máxima frequência de operação e número de elementos lógicos utilizados por cada um dos circuitos obtidos. Para tanto, foram realizadas simulações com o *software* de simulação

"*ModelSim*". Também foi realizada a análise dos sinais no interior da FPGA com o auxílio da ferramenta "*SignalTap II*".[5]

Este experimento também foi utilizado para verificar o comportamento do compilador LingPON-HD 1.0 e do simulador PON-HD 1.0, mas agora sem usar vetores. Como mencionado anteriormente, para tanto foi escrito e simulado um programa em LingPON-HD 1.0 que implementou o mesmo algoritmo de ordenação, só que agora para um número fixo de 16 elementos. O circuito sintetizado a partir deste programa é equivalente ao circuito sintetizado para 16 elementos descrito diretamente o VHDL. Assim foi possível observar, em mais um experimento, que o compilador gera os códigos VHDL de forma correta. A Figura 57 apresenta um trecho do programa em LingPON-HD 1.0 deste ordenador de dados. Neste trecho é possível observar que foram declarados 16 *Attributes* para alocar os dados a serem ordenados. Estes *Attributes* foram inicializados com dados inversamente ordenados. O código completo em LingPON-HD 1.0 para este experimento pode ser encontrado no Apêndice Q.

```
// ordenador de dados para 16 elementos inversamente ordenados
fbe fbeOrdenador
        attributes
                integer atData1 15
                integer atData2 14
                integer atData3 13
                integer atData4 12
                integer atData5 11
                integer atData6 10
                integer atData7 9
                integer atData8 8
                integer atData9 7
                integer atData10 6
                integer atData11 5
                integer atData12 4
                integer atData13 3
                integer atData14 2
                integer atData15 1
                integer atData16 0
                boolean finished false
                boolean atOdd false
        end_attributes
```

Figura 57 - Trecho do programa em LingPON-HD 1.0 do ordenador.

Fonte: Autoria própria.

[5] Um artigo apresentando este e alguns outros algoritmos de ordenação implementados com o PON-HD 1.0 foi apresentado no 12° Congresso Brasileiro de Inteligência Computacional em Curitiba (KERSCHBAUMER *et al.*, 2015). Um outro artigo descrevendo o PON-HD 1.0 e apresentando os resultados deste experimento foi publicado no Journal of Circuits, Systems and Computers (KERSCHBAUMER *et al.*, 2018a).

A necessidade de se fixar o número de elementos advém, como já mencionado, de uma limitação do LingPON e consequentemente do LingPON-HD 1.0, que é a falta de suporte a vetores parametrizáveis. Existe um esforço do grupo de pesquisa que trabalha com o PON para sanar tal deficiência.

4.5.2 Resultados do Experimento 5 (Ordenador de Dados)

Com este experimento foi possível verificar o comportamento dos componentes do *Framework* PON-HD 1.0 e do circuito com eles construído em *hardware*. Como já foi mencionado, via de regra, neste experimento não foi utilizado o compilador. Assim, via de regra, os componentes do *Framework* PON-HD 1.0 foram instanciados manualmente em um arquivo VHDL. Com isto, foi possível verificar que mesmo sem a utilização do compilador a utilização dos componentes predefinidos do *Framework* PON-HD 1.0 não foi tão difícil, pois basta apenas conectá-los através de sinais. Para ilustrar o funcionamento deste circuito a Figura 58 apresenta o comportamento de um conjunto de dados de 16 elementos durante a ordenação.

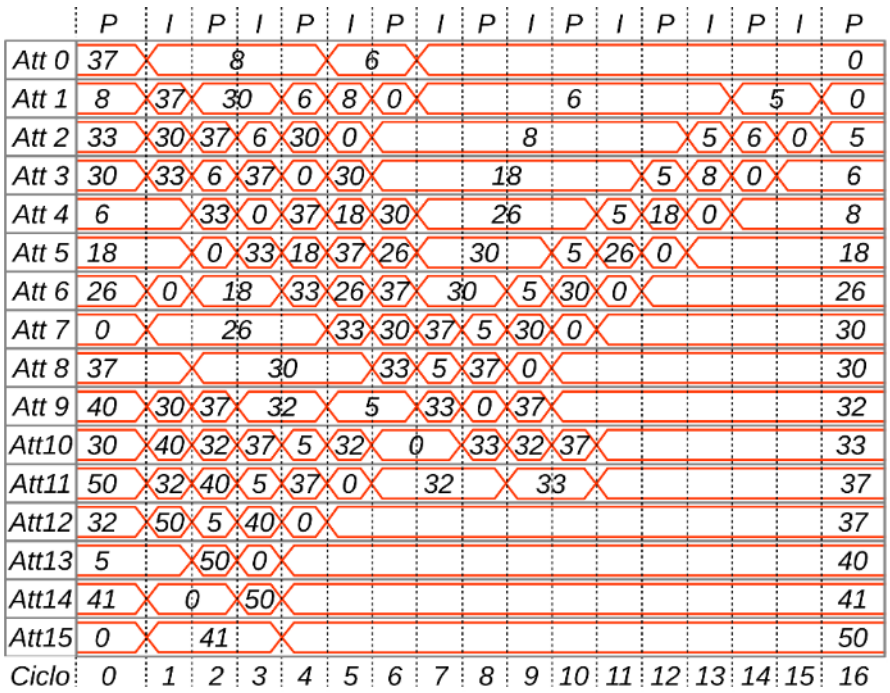

Figura 58 - Comportamento dos dados durante a ordenação.

Fonte: Autoria própria.

O gráfico apresentado na Figura 58 foi obtido com a ferramenta "*SignalTap II*" e demonstra o comportamento real do circuito durante a ordenação dos dados. Foram adicionadas na primeira linha do gráfico as letras "P" e "I", as quais indicam os ciclos pares e ímpares da ordenação, respectivamente. Pode-se destacar, como exemplo, o ciclo 13, onde observa-se a comparação de elementos adjacentes com índices ímpar-par.

Neste ciclo, dentre outras comparações, o conteúdo do *Attribute* 1 é comparado com o conteúdo do *Attribute* 2, como os valores destes *Attributes* são respectivamente 6 e 5 uma troca é necessária. Na borda de subida do próximo ciclo de *clock* a troca é realizada. O mesmo comportamento pode ser observado nos *Attributes* 3 e 4. Pode-se observar também o ciclo 14, que é um ciclo par (no qual se compara elementos adjacentes com índices par-ímpar), nele os *Attributes* 2 e 3 são comparados e então trocados.

Foi possível com este experimento verificar o comportamento do *framework* PON-HD 1.0 em circuitos de diferentes tamanhos, testando assim sua escalabilidade em função do uso de recursos de *hardware*, uma vez que foram utilizados conjuntos de dados de até três mil elementos, como já mencionado.

Um resultado importante em relação a escalabilidade do circuito é apresentado na Figura 59. Nesta figura é apresentada a máxima frequência de operação do circuito ordenador de dados para diferentes tamanhos de conjuntos de dados e em duas diferentes implementações, PON-HD 1.0 e VHDL tradicional.

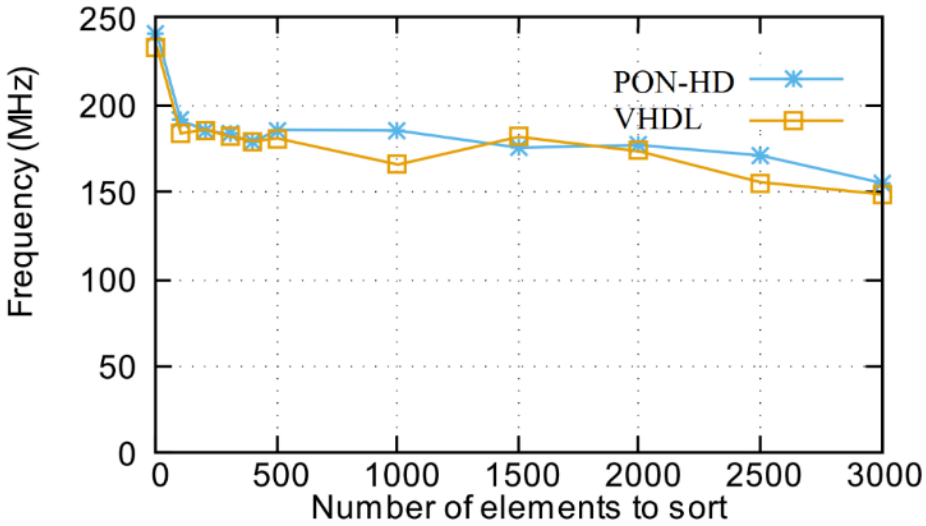

Figura 59 - Frequência de operação dos circuitos ordenadores.

Fonte: Autoria própria.

É possível observar que para este experimento a frequência de operação do circuito construído com o *framework* PON-HD 1.0 é igual e em alguns momentos até superior a frequência de operação do circuito sintetizado diretamente em VHDL. Este resultado demonstra que, ao menos para esta aplicação, o PON-HD 1.0 têm performance semelhante a mesma aplicação desenvolvida diretamente em VHDL.

Os dados de máxima frequência de operação para os vários conjuntos de dados foram obtidos diretamente do *software* "Quartus II" e são uma estimativa calculada pelo próprio fabricante do *hardware*.

As implementações em PON-HD 1.0 e VHDL receberam os mesmos conjuntos de dados de teste e os experimentos foram realizados nas mesmas condições. A Tabela 11 apresenta os dados utilizados para construir o gráfico da Figura 59.

Tabela 11 - Resultados obtidos nos experimentos em função do número de elementos a ordenar.

	Implementação PON-HD 1.0			**Implementação VHDL**		
Elementos a ordenar	**Frequência (MHz)**	**Elementos Lógicos**	**FFs**	**Frequência (MHz)**	**Elementos Lógicos**	**FFs**
4	204.42	118	65	198.89	110	65
100	191.17	3303	1601	183.65	3321	1601
200	185.74	6637	3201	186.08	6666	3201
300	183.08	9972	4801	181.92	10012	4801
400	178.79	13303	6401	179.28	13356	6401
500	185.77	16636	8001	180.86	16701	8001
1000	185.39	33303	16001	165.81	33424	16001
1500	175.53	49970	24001	181.79	50148	24001
2000	176.96	66636	32001	173.37	66872	32001
2500	171.06	83319	40001	155.69	83607	40001
3000	155.40	99978	48001	148.74	100323	48001

Fonte: Autoria própria.

Outro parâmetro analisado neste experimento foi o número de elementos lógicos utilizados na FPGA por cada uma das implementações do algoritmo de ordenação de dados, em função do número de elementos a ser ordenado. Os resultados desta análise são apresentados na Figura 60. Os dados utilizados para construir este gráfico também estão disponíveis na Tabela 11.

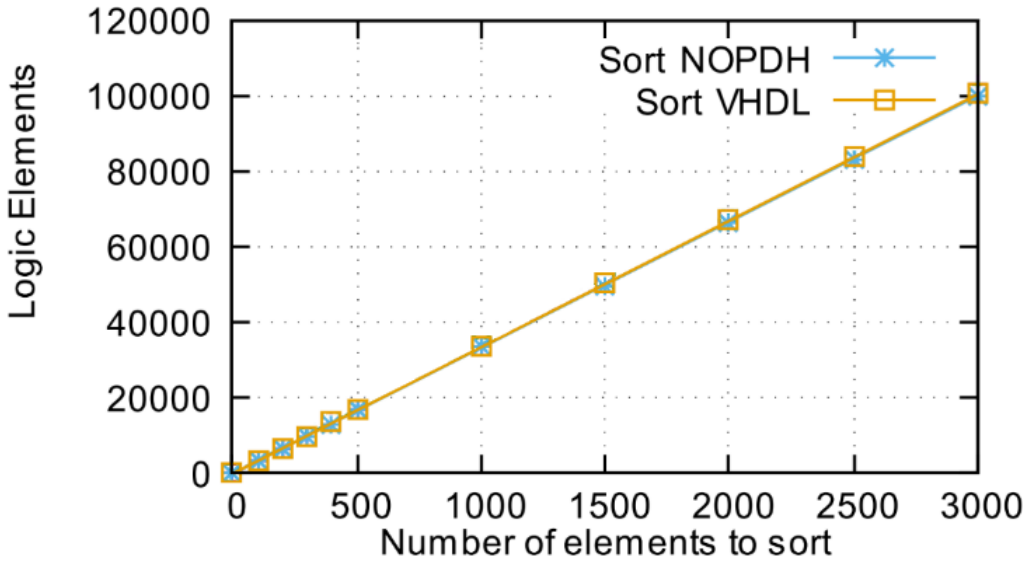

Figura 60 - Elementos lógicos utilizados pelo ordenador.

Fonte: Autoria própria.

Analisando os resultados apresentados no gráfico, é possível observar que a quantidade de elementos lógicos utilizados por ambas as implementações é muito parecida. A diferença entre as implementações em PON-HD 1.0 e em VHDL é menor que 1%. Assim, pode-se concluir que, neste experimento, o circuito sintetizado com o PON-HD 1.0 utiliza uma quantidade semelhante de elementos lógicos ao mesmo circuito sintetizado diretamente em VHDL. Este resultado é importante, pois como foi explicado anteriormente, uma das deficiências das ferramentas de síntese em alto nível tradicionais é o tamanho dos circuitos gerados, tipicamente maiores que os equivalentes gerados diretamente em VHDL. Esta questão é trazida à luz pelos seguintes trabalhos: Meeus (2012) e Nane (2016).

É importante ressaltar que os números são semelhantes, porém não iguais, o que indica que os circuitos gerados não são os mesmos. Uma análise dos circuitos com a ferramenta “RTL Viewer” confirmou que os circuitos gerados são diferentes, mas apresentam o mesmo comportamento, ou seja, executam o algoritmo de ordenação da mesma forma. Adicionalmente, como a lógica de ordenação implementada em PON-HD 1.0 e em VHDL foi a mesma, a forma de funcionamento do ordenador é a mesma e assim o tempo de execução (número de ciclos de *clock*) foi o mesmo em ambos os casos.

Com este experimento, obteve-se indícios, de que a utilização do PON-HD 1.0 na síntese de circuitos digitais possivelmente não traz prejuízos com relação a frequência de operação do circuito, ao número de elementos lógicos utilizados e ao tempo de execução dos algoritmos, se comparado a mesma aplicação realizada em VHDL. Isso é claro, vai depender

da natureza da aplicação, ou seja, este resultado não pode ser extrapolado para todas as aplicações em PON-HD 1.0.

Ainda, é pertinente lembrar que estes resultados são dependentes da habilidade do desenvolvedor em descrever os algoritmos tanto em *framework* PON-HD 1.0 quanto em VHDL. No *framework* PON-HD 1.0, apesar de estruturas padronizadas, ainda assim a lógica aplicada pode influenciar no número de elementos padronizados sintetizados. Na implementação puramente em VHDL, tanto a lógica quanto a estruturação dependem do desenvolvedor, sendo esta mais suscetível a capacidade técnica deste. Entretanto, no âmbito do experimento dado, um grande esforço foi realizado para tornar ambas as implementações as mais otimizadas possível nesse quadro dado.

Isto considerado, o *framework* PON-HD 1.0 se mostrou estável durante o experimento pois foi capaz de lidar com circuitos de expressivo tamanho e complexidade sem apresentar falhas ou inconsistências.

4.5.3 Conclusões do Experimento 5 (Ordenador de Dados)

Neste experimento, conjuntos de dados são ordenados de forma paralela. São utilizados conjuntos de dados de tamanhos diferentes, permitindo a análise da escalabilidade das aplicações desenvolvidas com o *framework* PON-HD 1.0. Este ordenador também foi desenvolvido diretamente em VHDL, usando desenvolvimento usual, permitindo assim comparações entre os dois circuitos gerados, em termos de utilização de recursos e velocidade de *clock*.

Analisando os resultados deste experimento conclui-se que a utilização do *framework* PON-HD 1.0 possivelmente não traz grandes prejuízos com relação a frequência de operação do circuito, ao número de elementos lógicos utilizados e ao tempo de execução dos algoritmos, se comparado a mesma aplicação realizada em VHDL usual. Isso para o experimento em questão. O circuito do experimento apresentou estabilidade, mesmo trabalhando com conjuntos de dados de expressivo tamanho.

4.6 Experimento 6: Controlador de Esteira

Uma dupla de discentes[6], do curso de Bacharelado em Engenharia de Computação, da UTFPR – Campus Curitiba – Sede Central – DAINF/DAELN, interessou-se no ferramental PON-HD 1.0. Isto se deu no âmbito da disciplina "LRH0087 - Lógica Reconfigurável Por *Hardware*", ministrada pelo Professor Carlos Raimundo Erig Lima, após uma apresentação sobre o ferramental PON-HD 1.0 feita pelo doutorando autor deste documento de pesquisa. O interesse de deu em desenvolver seu trabalho final desta disciplina utilizando o ferramental PON-HD 1.0.

4.6.1 Descrição do Experimento 6 (Controlador de Esteira)

Aproveitando o interesse apresentado por essa dupla de discentes, surgiu sinergicamente a oportunidade de testar o ferramental PON-HD 1.0, principalmente com relação a facilidade de utilização, a luz da avaliação de outrem. Neste sentido, por se tratar de um grupo de discentes não pertencentes ao grupo de pesquisa do PON, sua experiência na utilização do PON-HD 1.0 trouxe informações relevantes ao desenvolvimento deste ferramental. Assim, o objetivo deste experimento foi verificar a facilidade de utilização e validar o ferramental PON-HD 1.0, isto por parte de elementos externos ao grupo de pesquisa do PON.

O problema proposto pelo grupo para ser resolvido com a utilização de lógica reconfigurável é o projeto de um controlador para uma esteira que transporta caixas de papelão com produtos do setor de embalagem para o setor de carga. Esta esteira transporta três tamanhos de caixas que devem ser alocadas em paletes. Cada palete tem sua capacidade limitada a um arranjo de caixas de diferentes tamanhos. O controlador faz o acionamento da esteira e monitora o tamanho das caixas através de sensores. O controlador também apresenta em um conjunto de *displays* o número de caixas de cada tamanho que passaram pela esteira.

Existe ainda um semáforo no final da esteira que deve ser controlado pelo circuito de forma a indicar aos operadores se ainda existe espaço para a caixa presente na esteira no palete.

A Figura 61 apresenta um diagrama para o circuito proposto neste experimento.

---

[6] Aluno: 1047680 - Andre Augusto Kaviatkovski Curso: 212 - Eng De Computação, Aluno: 1338773 - Gabriel Rodrigues Garcia Curso: 212 - Eng De Computação.

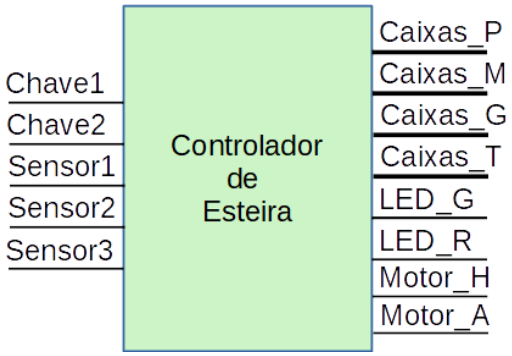

Figura 61 - Diagrama do controlador de esteira.

Fonte: Autoria própria.

Um conjunto de condições descrito no projeto define o funcionamento deste semáforo. O relatório descrevendo na totalidade o projeto desenvolvido pode ser visualizado no Anexo B. Ainda o código escrito em LingPON-HD 1.0 para este projeto pode ser visualizado na integra no Anexo C. É necessário esclarecer que este experimento foi realizado em uma versão anterior da linguagem LingPON-HD, e assim o código do anexo difere em parte dos códigos já apresentados para esta linguagem.

4.6.2 Resultados do Experimento 6 (Controlador de Esteira)

A experiência vivida pelos discentes do grupo durante a utilização do ferramental PON-HD 1.0 no desenvolvimento de seu projeto, por sua vez, é descrita em um documento redigido por eles chamado “Avaliação PON-HD”. O documento pode ser visualizado na integra no Anexo D. Neste documento são descritos, entre outras coisas, alguns pontos fortes do PON-HD 1.0 que segundo a experiência destes discentes e em suma são: (a) linguagem de alto nível; (b) a linguagem é mais acessível a um programador do que VHDL puro; (c) proximidade com o raciocínio humano; (d) se bem projetado antes, a implementação é bastante simples e direta; e (e) desempenho interessante uma vez que explora implicitamente o paralelismo permitido pela FPGA.

Foram também destacados alguns pontos negativos relacionados a recursos do PON-HD 1.0. Seguem dois pontos negativos: (a) a falta de um operador “OR” entre *Métodos* (já resolvido nas versões mais recentes da LingPON); e (b) a limitada documentação que explique a utilização do mesmo.

Observando os relatos dos discentes é possível obter os primeiros indícios de outrem que o ferramental PON-HD 1.0 está atingindo seus objetivos, uma vez que a solução proposta

pelo PON-HD 1.0 permitiu o desenvolvimento de circuitos digitais de forma mais fácil, por meio da descrição através de regras em alto nível. Ainda, é pertinente destacar que, dentre os pontos positivos enumerados pelos discentes, a linguagem LingPON-HD 1.0 foi de fato considerada de alto nível, acessível e mais próxima do raciocínio humano.

Tais características vêm de encontro ao objetivo de garantir que o ferramental PON-HD 1.0 permita expressão em alto nível, permitindo que desenvolvedores com pouca experiência em síntese de *hardware* desenvolvam circuitos digitais.

O algoritmo proposto utilizou 15 *Attributes*, 22 *Premises*, 36 *Rules*, 26 *Methods*. O circuito obtido da síntese deste algoritmo utilizou 175 elementos lógicos e 141 registradores, operando em 178,18 MHz.

O correto funcionamento do circuito gerado com o PON-HD 1.0 no experimento dado, demonstrado pelo correto funcionamento do projeto realizado pela equipe, conforme relatório apresentado no Anexo B, ajudou a validar o ferramental do PON-HD 1.0 como um todo.

4.6.3 Conclusões do Experimento 6 (Controlador de Esteira)

Este experimento possibilitou avaliar a facilidade de utilização e o funcionamento do ferramental PON-HD 1.0 como um todo. Características ressaltadas pelos discentes que realizaram o experimento, tais como, linguagem LingPON-HD 1.0 de alto nível, acessível e mais próxima do raciocínio humano ajudam a tornar o ferramental PON-HD 1.0 de fácil utilização, permitindo que desenvolvedores com pouca experiência em desenvolvimento de *hardware* desenvolvam circuitos digitais. É importante salientar que são apenas dois alunos envolvidos no experimento e assim são ainda necessários mais experimentos neste sentido para verificar estas condições com mais confiabilidade. Dito isso, conclui-se que o experimento apresenta consideráveis indícios que sinalizam que o ferramental PON-HD 1.0 facilita o desenvolvimento de circuitos em *hardware* digital.

A linguagem LingPON-HD 1.0 tem grande importância neste sentido pois propicia expressividade por meio de regras em alto nível permitindo a utilização de entidades coesas e desacopladas na descrição dos circuitos.

4.7 Experimento 7: Controlador de Robô

Neste experimento é desenvolvido um controlador de um robô, para um robô hexápode. O experimento foi concebido com o objetivo de verificar as principais características da ferramenta PON-HD 1.0. As principais características do PON-HD 1.0 analisadas são: a resolução de conflitos, a estabilidade, a facilidade de utilização (expressividade através de regras em alto nível) e a validação do simulador.

4.7.1 Descrição do Experimento 7 (Controlador de Robô)

O controlador de robô foi idealizado para se comunicar através de uma porta serial com o robô a ser controlado. A Figura 62 apresenta o diagrama do circuito proposto para o controlador de robô hexápode.

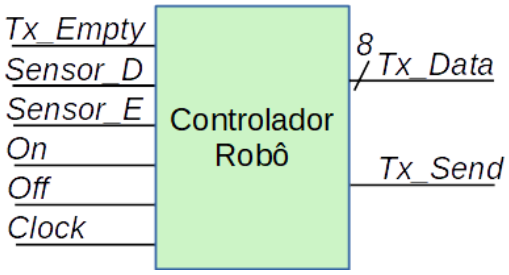

Figura 62 - Diagrama do controlador de robô.

Fonte: Autoria própria.

A Figura 63 apresenta um diagrama que ilustra o fluxo do experimento. Neste diagrama é possível observar as várias etapas de desenvolvimento e de verificação realizadas no experimento sobre o controlador de um robô hexápode simulado.

Conforme a Figura 63, o robô hexápode é simulado em um computador e a ligação entre o computador e o *hardware* do experimento se dá através de um canal de radiofrequência. A Figura 64 apresenta o robô utilizado no experimento.

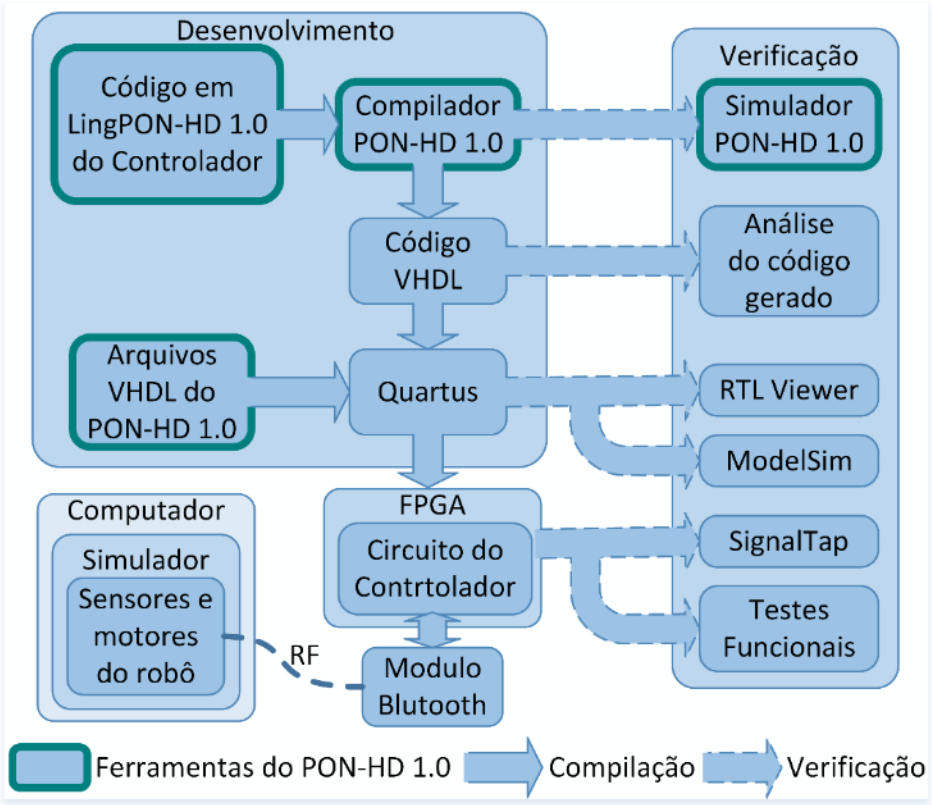

Figura 63 - Diagrama de fluxo do experimento 4.

Fonte: Autoria própria.

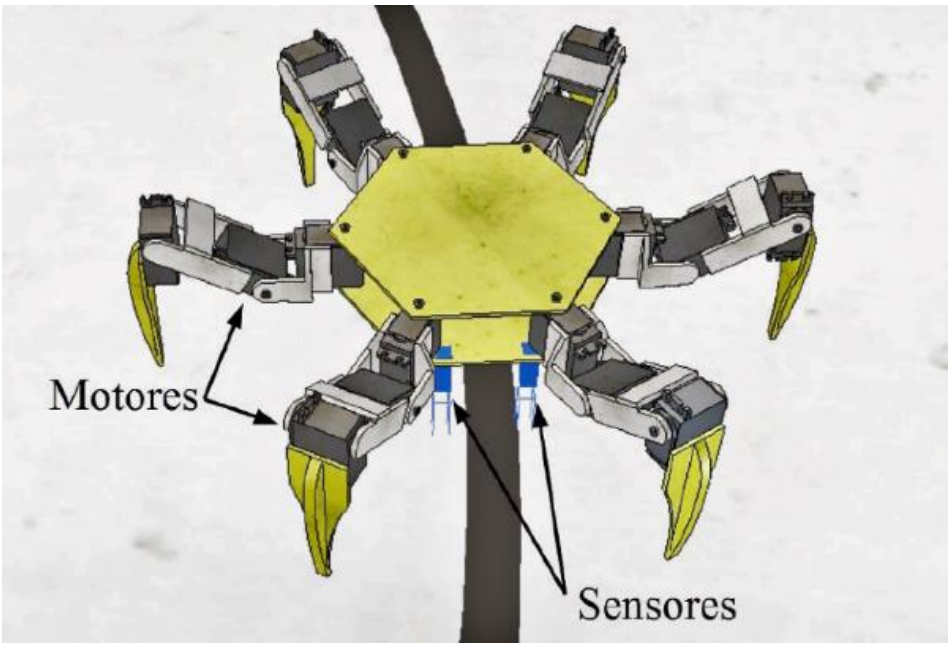

Figura 64 – Representação do robô hexápode.

Fonte: Autoria própria.

O robô é composto por seis pernas, cada uma das quais possui três servo-motores. Esta configuração exige o controle simultâneo de dezoito motores. Além disso, o robô possui ainda dois sensores óticos, que identificam a cor da superfície sob o robô. Este robô não foi construído fisicamente. O mesmo foi de fato simulado com o auxílio de um computador, por meio de um simulador de robôs chamado V-Rep (ROHMER, 2013).

Para executar a tarefa de controlar este robô o controlador calcula sucessivamente as posições angulares de cada um dos dezoito motores de forma a fazer o robô caminhar acompanhando a linha.

O robô é capaz de determinar sua posição em relação à linha utilizando sensores ópticos. As entradas do controlador são os dois sinais que representam a cor da superfície sob cada um dos sensores. A saída é a posição angular de cada um dos dezoito motores.

O *hardware* utilizado no experimento é uma placa de desenvolvimento “DE-0 Nano” que utiliza uma FPGA Cyclone IV da Altera. A mesma pode ser observada na Figura 65. Nesta figura, também é possível observar o módulo Bluetooth modelo “HC-05” responsável pela comunicação entre a FPGA e o computador que executou a simulação.

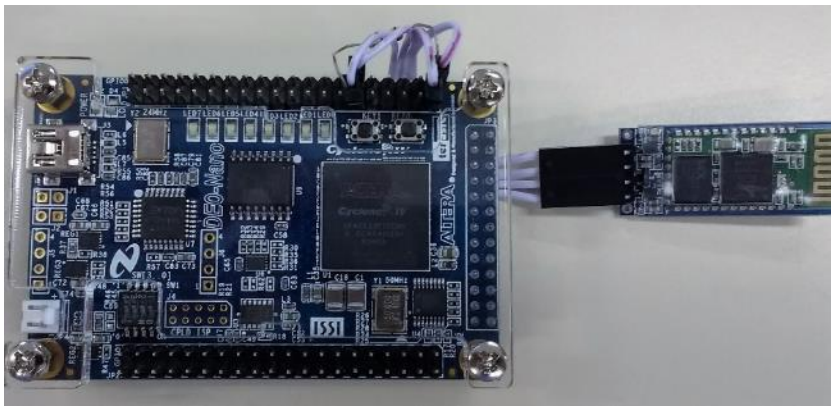

Figura 65 - *Hardware* utilizado no experimento do robô.
Fonte: Autoria própria.

Neste sentido, a conexão entre a FPGA com o controlador e o computador onde o robô foi simulado se deu por meio de um *link* de comunicação de dados Bluetooth, conforme apresentado na Figura 66.

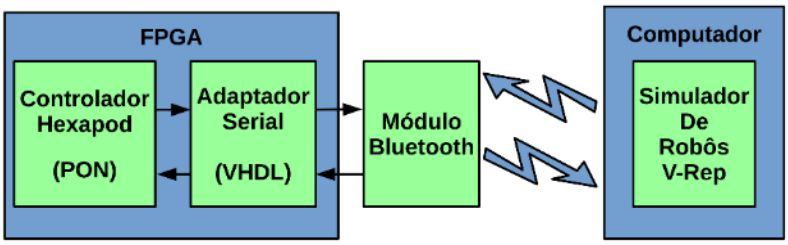

Figura 66 - Conexão entre a FPGA e o computador.

Fonte: Autoria própria.

Toda a temporização e sincronização entre os elementos da simulação foi de responsabilidade da aplicação desenvolvida em LingPON-HD 1.0.

A estrutura adotada para o experimento permitiu que o controlador implementado no interior da FPGA controlasse o robô simulado como se estivesse efetivamente controlando um robô físico. Os sinais dos sensores do robô foram transmitidos para o controlador, que por sua vez enviou para o robô a posição angular que cada um dos dezoito motores deveria adotar.

O circuito controlador sintetizado na FPGA foi implementado via LingPON-HD 1.0 e para isso foram utilizados 29 *Attributes*, 50 *Premises*, 127 *Rules* e 103 *Methods*. Esse tem a função de controlar os motores de forma que o robô possa caminhar seguindo o percurso definido por uma linha preta sobre uma superfície branca.

A Figura 67 apresenta um trecho do programa em LingPON-HD 1.0. No trecho apresentado é realizado parte do processamento do sinal dos sensores que detectam a linha. Nele, a decisão sobre se o robô deve virar à esquerda é tomada com base no sinal destes sensores. O programa completo de controle do robô hexápode, em LingPON-HD 1.0, pode ser encontrado no Apêndice R.

```
rule RlViraEsquerda
        condition
                subcondition SubViEs
                        premise prViEsSd controlador.sensorDir == false and
                        premise prViEsSe controlador.sensorEsq == true
                end_subcondition
        end_condition
        action
                instigation inViraEsquerda controlador.mtViraEsquerda();
        end_action
end_rule
```

Figura 67 - Trecho do programa de controle do robô.

Fonte: Autoria própria.

Para permitir o sincronismo dos motores e assim realizar os movimentos do robô foi implementada uma máquina de estados síncrona em LingPON-HD 1.0. Para isso, um *Attribute* foi utilizado para contar e sincronizar as etapas dos movimentos de cada uma das pernas. A Figura 68 apresenta um diagrama que representa esta máquina de estados.

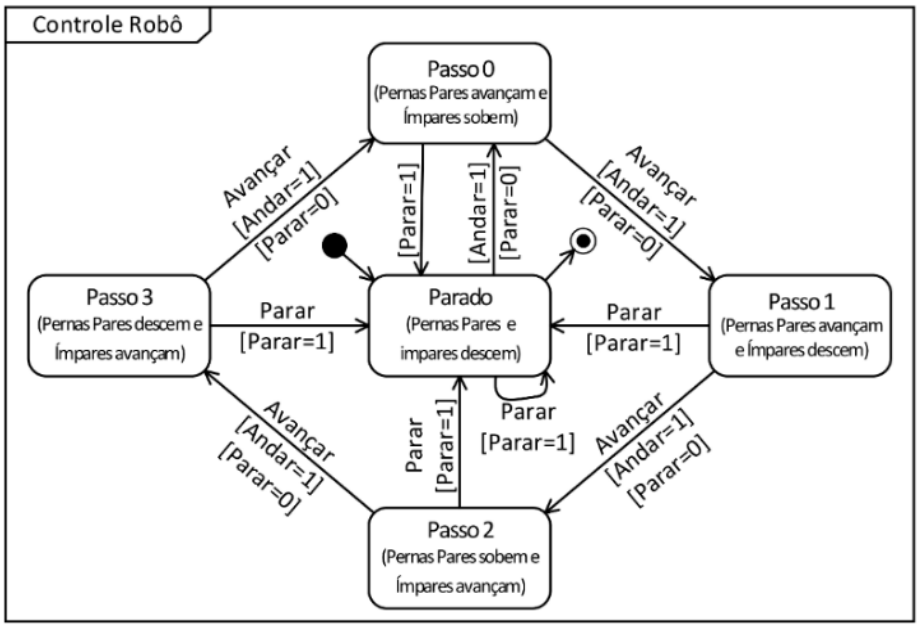

Figura 68 - Máquina de estados do controle do robô hexápode.

Fonte: Autoria própria.

A direção que o robô deve tomar (ir em frente, à esquerda ou à direita) é determinada com base nos sensores. A mudança de direção é feita manipulando a amplitude dos passos realizados pelas pernas em cada um dos lados do robô.

A utilização de máquinas de estado no projeto de aplicações em LingPON-HD 1.0 é de grande ajuda. Sua utilização facilita a tarefa de sincronização entre os vários elementos do programa, uma vez que devido ao paralelismo de execução do PON-HD 1.0 no *hardware* tudo acontece ao mesmo tempo.

Foram realizados vários testes, com diferentes velocidades de movimentação do robô, diferentes configurações de movimentos na realização dos passos e diferentes percursos de teste. Foi utilizado ainda um módulo escrito em VHDL para realizar a comunicação com o módulo Bluetooth. Assim foi possível verificar a integração entre um elemento escrito em PON-HD 1.0 e um elemento descrito em VHDL de forma tradicional.

Além dos experimentos de funcionamento do circuito no controle do robô foram também realizadas análises dos sinais internos do circuito, com o objetivo de verificar a correta

construção do *hardware* por parte das ferramentas do PON-HD 1.0. Estas análises foram realizadas com as ferramentas "*ModelSim*", "*SignalTap II*" e "*RTL Viewer*".

É importante destacar que na análise do diagrama RTL do circuito foi possível observar que, apesar da saída do circuito ser serial, o controle dos dezoito motores utilizados aconteceu de forma paralela. Infelizmente estes diagramas possuem grandes dimensões e seus componentes não estão ordenados de forma didática, o que exige a utilização de ferramentas de *software* para a navegação no mesmo e impede sua inclusão neste documento.

4.7.2 Resultados do Experimento 7 (Controlador de Robô)

Este experimento demostrou que o controlador funcionou corretamente, controlando o robô de forma adequada, permitindo ao mesmo andar e fazer curvas seguindo o caminho determinado pela linha. A Figura 69 apresenta uma captura de tela do simulador de robôs V-Rep simulando o robô hexápode, sendo controlado pelo circuito implementado com o ferramental PON-HD 1.0.

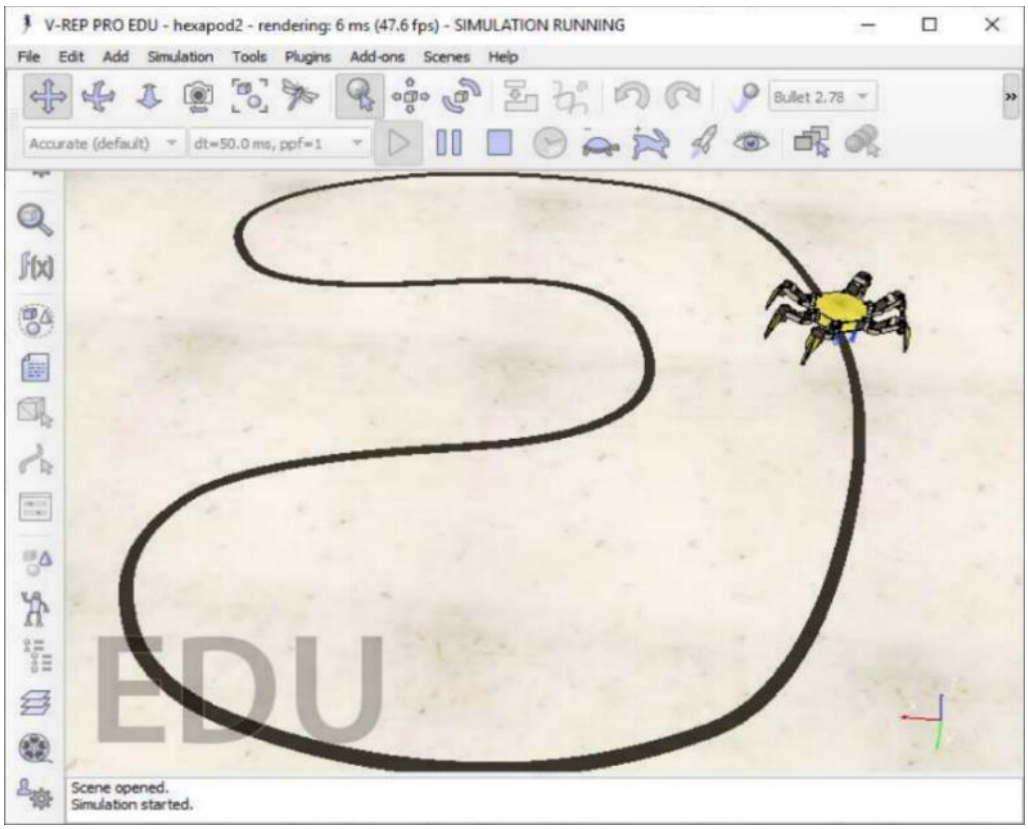

Figura 69 - Simulação do robô hexápode.

Fonte: Autoria própria.

Este experimento permitiu observar uma característica importante do PON-HD 1.0, a resolução de conflitos. Como todas as *Premises*, *Rules*, e *Methods* são processadas paralelamente e toda a sincronização do movimento dos dezoito motores acontece segundo as regras estabelecidas no programa, existem situações em que mais de um *Methods* tentam gravar no mesmo *Attribute* ao mesmo tempo. O mecanismo de resolução de conflitos implementado no PON-HD 1.0 tratou corretamente estas situações evitando os conflitos e garantindo o correto funcionamento do circuito.

A complexidade do programa desenvolvido em LingPON-HD 1.0 para este experimento permitiu uma melhor verificação do funcionamento do simulador PON-HD 1.0, o qual se mostrou funcional e fiel ao comportamento esperado para o circuito.

Apesar da complexidade do circuito e da necessidade de sincronismo entre os dezoito motores o experimento apresentou indícios de que a expressividade propiciada pela programação através de regras em alto nível permite a descrição do comportamento do circuito em LingPON-HD 1.0 de forma simples e descomplicada. O desacoplamento entre os elementos do *framework* PON-HD 1.0 e o paralelismo na execução das regras facilitaram este processo.

O controlador deste experimento foi intencionalmente desenvolvido na forma de um componente, que por sua vez foi utilizado junto a outro componente desenvolvido em VHDL (controlador de comunicação serial) para compor o circuito final de controle do robô. Esta topologia foi apresentada na Figura 66. Desta forma foi possível verificar a possibilidade de desenvolver partes de uma aplicação em PON-HD 1.0 enquanto outras partes desta mesma aplicação podem ser desenvolvidas em outras linguagens de descrição de *hardware*. O resultado foi positivo pois como os componentes do *Framework* PON-HD 1.0 estão em VHDL conecta-los a outros componentes do circuito é naturalmente fácil. Isto acontece porque as ferramentas de síntese permitem naturalmente a junção de módulos VHDL na composição do circuito.

Durante o experimento o circuito foi testado exaustivamente, com a simulação do robô operando durante horas sem apresentar problemas, demostrando assim a estabilidade do circuito sintetizado com o ferramental PON-HD 1.0.

4.7.3 Conclusões do Experimento 7 (Controlador de Robô)

Este experimento, bastante complexo, permitiu verificar principalmente a facilidade de utilização do ferramental PON-HD 1.0 na síntese de *hardware* digital, a estabilidade dos

circuitos com ele gerados, a resolução de conflitos e o funcionamento do simulador. O paralelismo implícito na execução dos elementos do PON-HD 1.0 também foi observado com este experimento. Assim, com uma análise mais profunda do circuito gerado com o ferramental PON-HD 1.0, conclui-se que o comportamento do circuito reflete a estrutura de notificações característica do PON. Conclui-se também que o mecanismo de resolução de conflitos é efetivamente funcional. O experimento também permitiu concluir que o simulador funciona corretamente. O tamanho e a complexidade do circuito gerado com o ferramental PON-HD 1.0 neste experimento permitem concluir que o PON-HD 1.0 é estável, independentemente do tamanho da aplicação. Este experimento também forneceu indícios que sinalizam que o ferramental PON-HD 1.0 permite a descrição em alto nível de aplicações destinadas ao desenvolvimento em lógica reconfigurável, tornando mais fácil a utilização desta plataforma.

4.8 Experimento 8: Controlador PID

O oitavo e último experimento é a implementação de um controlador PID. Maiores informações sobre este tipo de controladores podem ser obtidas em KOCUR *et al.* (2014). Este circuito foi escolhido por ser mais complexo que os anteriores e por não possuir predominância de componentes sequenciais ou combinacionais, permitindo maiores comparações entre as ferramentas PON-HD 1.0 e Vivado HLS.

4.8.1 Descrição do Experimento 8 (Controlador PID)

Este experimento consiste da implementação de um controlador PID. Este controlador foi implementado em PON-HD 1.0 e em Vivado HLS de forma a permitir comparações entre as duas ferramentas. O objetivo deste experimento é realizar um comparativo de performance entre os circuitos gerados pelo PON-HD 1.0 e pela ferramenta vivado HLS, para um circuito mais complexo de cunho mais prático. Assim, como nos experimentos anteriores, houve uma grande preocupação em tornar as duas implementações o mais semelhante possível, de forma a tornar as comparações mais precisas.

A Figura 70 apresenta um diagrama que ilustra o fluxo de execução do experimento 8.

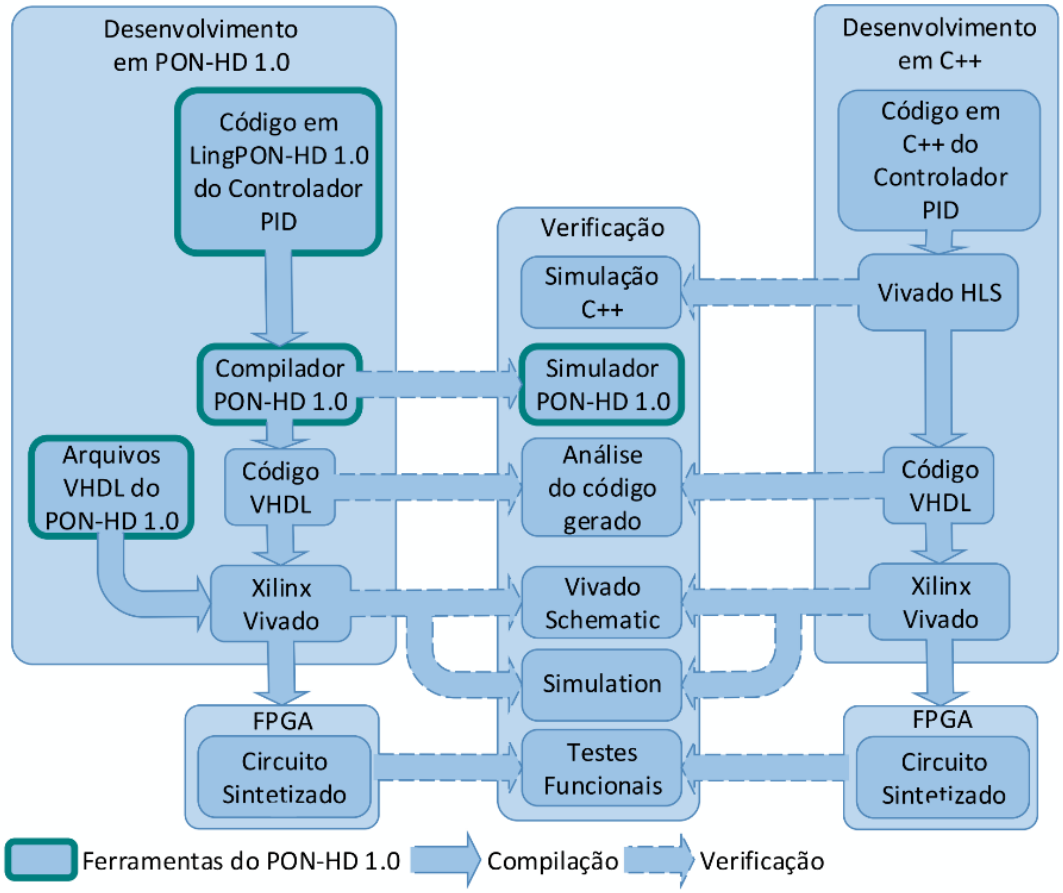

Figura 70 - Diagrama de fluxo do experimento 8.

Fonte: Autoria própria.

Neste diagrama é possível observar as várias etapas de realização do experimento.

O diagrama de blocos da Figura 71 apresenta a topologia do controlador proposto para este experimento.

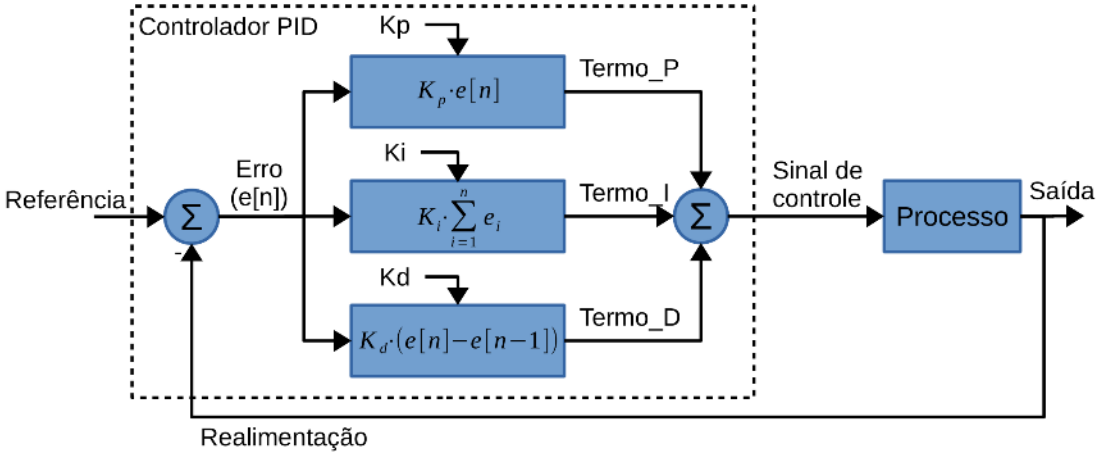

Figura 71 - Diagrama de blocos do experimento 8.

Fonte: Autoria própria.

Observando o diagrama, nota-se que o circuito tem como entradas um sinal de referência e um sinal de realimentação, além de três entradas de parametrização do controlador que são os valores de Kp, Ki e Kd. A saída do circuito é o sinal de controle aplicado ao processo.

A primeira etapa do funcionamento do controlador é o cálculo do erro. Este erro é calculado subtraindo-se a realimentação da referência. Na sequência são calculados os termos Proporcional (P), Integral (I) e Derivativo (D). O termo proporcional é calculado multiplicando-se o erro atual pelo valor de Kp. O termo integral é calculado multiplicando-se o erro acumulado (somatório dos erros anteriores) pelo valor de Ki. E finalmente o termo derivativo é calculado multiplicando-se a diferença entre o erro atual e o erro anterior pelo valor de Kd. O sinal de controle (saída do circuito controlador PID) é o somatório dos três termos.

Outra tarefa que o circuito deve desempenhar e que não é visível no diagrama é evitar o estouro das variáveis, ou seja as operações não podem produzir resultados que ultrapassem a capacidade das variáveis utilizadas.

4.8.1.1 Desenvolvimento do controlador PID em PON-HD 1.0

Iniciou-se o desenvolvimento do controlador PID em PON-HD 1.0 pelo desenvolvimento do código em LingPON-HD 1.0. A Figura 72 apresenta um trecho do código em LingPON-HD 1.0 do controlador PID em PON-HD 1.0.

O código completo do programa em LingPON-HD 1.0 pode ser encontrado no Apêndice S O código elaborado para este experimento é composto por 18 *Attributes*, 6 *Premises*, 5 *Rules*, 28 *Methods*.

No algoritmo escrito em LingPON-HD 1.0 optou-se por separar as operações necessárias em 6 etapas. Na primeira etapa, é realizada a leitura das entradas e o armazenamento dos dados nos *Attributes*. Na segunda etapa, são calculados os erros relacionados aos termos Integral e Derivativo. Na terceira etapa, são calculados os termos Proporcional e Derivativo e são verificados os limites do acumulador de erro. Na quarta etapa, é calculado o termo Integral e realizada a soma dos termos Proporcional e Derivativo. Na quinta etapa, é adicionado a soma anterior o valor do termo Integral. Na sexta e última etapa, são verificados os limites do valor de saída e a saída do controlador é atualizada.

```
fbe f_controladorPID
  attributes
    boolean atReset 0
    boolean atNovoDado 0
    integer atErroIntegral 0
    integer atErroAnterior 0
    integer atErroD 0
    integer atErro
    integer atTermo_P 0
    integer atTermo_I 0
    integer atTermo_D 0
    integer atReferencia 0
    integer atRealimentacao 0
    integer atSinalControle 0
    integer atKp 0
    integer atKi 0
    integer atKd 0
    integer atSaidaTmp 0
    integer atEtapa 0
  end_attributes
  methods
    method mtResetIntegral(atErroIntegral = 0)
    method mtResetEtapa(atEtapa = 0)
    method mtResetSaida(atSinalControle = 0)
    method mtResetErroAnterior(atErroAnterior = 0)
    method mtNovoDado(atEtapa = 1)
    method mtCalculaErro(atErro = atReferencia -
atRealimentacao)
    method mtEtapa2(atEtapa = 2)
    method mtCalculaErroI(atErroIntegral = atErroIntegral
+ atErro)
    method mtCalculaErroD(atErroD = atErro -
atErroAnterior)
    method mtEtapa3(atEtapa = 3)
    method mtCalculaP(atTermo_P = atKp * atErro)
    method mtCalculaD(atTermo_D = atKd * atErroD)
    method mtMaiorMax(atErroIntegral = 32000)
    method mtMenorMin(atErroIntegral = -32000)
    method mtResetTemp(atTemp = 0)
    method mtResetSaidaTmp(atSaidaTmp = 0)
    method mtMaiorMax(atErroIntTmp = 10000)
    method mtMenorMin(atErroIntTmp = -10000)
    method mtSaidaMaiorMax(atSinalControle = 10000)
    method mtSaidaMenorMin(atSinalControle = -10000)
    method mtCalculaErro(atErro = atReferencia -
atRealimentacao)
    method mtGravaErroP(atErroP = atErro)
    method mtErroIntTmp(atErroIntTmp = atErroI + atErro)
    method mtErroD(atErroD = atErro - atErroAnterior)
    method mtArmazenaErro(atErroAnterior = atErro)
    method mtCalculaP(atTermo_P = atKp * atErroP)
    method mtCalculaErroI(atErroI = atErroIntTmp)
    method mtCalculaD(atTermo_D = atKd * atErroD)
    method mtCalculaI(atTermo_I = atKi * atErroI)
    method mtTmp(atTemp = atTermo_P + atTermo_D)
    method mtSaidaTmp(atSaidaTmp = atTemp +
atTermo_I)
    method mtSaida(atSinalControle = atSaidaTmp)
  end_methods
end_fbe

inst
  f_controladorPID cPID
end_inst

strategy
  no_one
end_strategy

// verifica a condição de reset
rule RlReset
  condition
    subcondition SubReset
      premise prReset cPID.atReset == true
    end_subcondition
  end_condition
  action
    instigation inResetSaida cPID.mtResetSaida();
    instigation inResetErroAnterior
cPID.mtResetErroAnterior();
    instigation inResetErro cPID.mtResetErro();
    instigation inResetErroP cPID.mtResetErroP();
    instigation inResetErroIntTmp
cPID.mtResetErroIntTmp();
    instigation inResetErroD cPID.mtResetErroD();
    instigation inResetErroI cPID.mtResetErroI();
    instigation inResetTermo_P cPID.mtResetTermo_P();
    instigation inResetTermo_I cPID.mtResetTermo_I();
    instigation inResetTermo_D cPID.mtResetTermo_D();
    instigation inResetTemp cPID.mtResetTemp();
    instigation inResetSaidaTmp cPID.mtResetSaidaTmp();
  end_action
end_rule
```

Figura 72 - Código em LingPON-HD 1.0 para o experimento 8.

Fonte: Autoria própria.

É importante salientar que, apesar de serem necessárias 6 etapas para realizar os cálculos, todas estas etapas acontecem paralelamente, ou seja, a cada ciclo de *clock* um novo conjunto de entradas é lido e um novo sinal de controle é apresentado na saída, diferentemente da implementação em Vivado HLS. O funcionamento é na forma de "*pipeline*" e assim a cada ciclo de *clock* tem-se um novo resultado apresentado na saída. Nesta topologia, uma alteração na entrada vai provocar uma alteração na saída após 6 ciclos de *clock*.

Assim, como nos experimentos anteriores, uma vez concluído o desenvolvimento do código em LingPON-HD 1.0, o mesmo foi compilado com o compilador PON-HD 1.0,

resultando em um arquivo VHDL. Também foi utilizado nesta etapa o simulador PON-HD 1.0, permitindo verificar se o programa implementado em LingPON-HD 1.0 funcionou corretamente.

O código VHDL gerado foi inspecionado e, estando de acordo com o esperado, prosseguiu-se para o processo de síntese do circuito. O código VHDL completo gerado pelo compilador PON-HD 1.0 pode ser observado no Apêndice T.

Assim, como nos experimentos anteriores, utilizou-se a ferramenta Vivado 2018.1 para a síntese do código VHDL gerado pelo PON-HD 1.0. A FPGA escolhida para as simulações é do modelo XA7A12TCPG238-2L da família Artix-7 fabricada pela Xilinx.

Para verificar o funcionamento do circuito foram realizados alguns testes. Inicialmente, verificou-se o esquemático do circuito gerado pela ferramenta de síntese. Na sequência, foi realizada uma análise mais profunda do funcionamento do circuito através de simulações.

A Figura 73 apresenta o bloco de circuito gerado pela síntese do código VHDL para o controlador PID, resultante do programa em LingPON-HD 1.0.

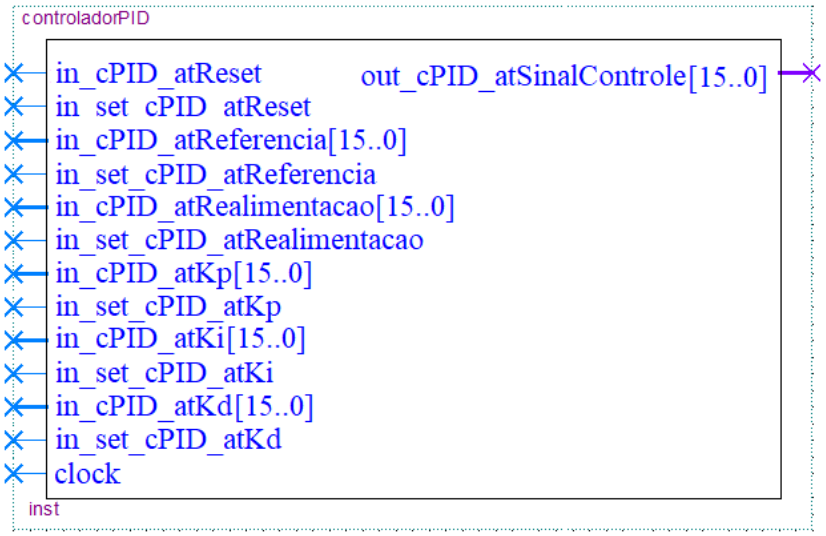

Figura 73 - Bloco gerado pela síntese do código do experimento 8.

Fonte: Autoria própria.

Nesta figura, é possível observar a entrada de *clock*, a entrada de *reset*, a entrada de referência, a entrada de realimentação e as entradas dos três parâmetros do controlador (Kp, Ki e Kd). Com exceção do *clock* e do *reset*, todas as entradas são valores inteiros de 16 bits. Para cada uma destas entradas existe também um sinal de entrada de atribuição (set) que habilita sua gravação no respectivo *Attribute* de entrada. A saída do controlador também é um sinal de 16

bits, e tem o nome de “sinal de controle”. Estes sinais de atribuição não estão presentes no diagrama da Figura 71, porém são necessários na implementação em PON-HD 1.0 para o gerenciamento do armazenamento dos dados nos *Attributes.*

Para realizar as comparações propostas o mesmo circuito foi implementado na ferramenta Vivado HLS. A seção a seguir apresenta esta implementação.

4.8.1.2 Desenvolvimento do controlador PID em Vivado HLS

Com o auxílio da ferramenta Vivado HLS, versão 2018.1, o controlador PID foi também implementado em C++. A Figura 74 apresenta um trecho do código em C++ para o controlador PID.

O código completo elaborado em C++ para o controlador PID pode ser visualizado no Apêndice U.

```
10  void controladorPID(bool reset, int16 referencia, int16 realimentacao, int16 *
    sinalControle, int16 kp, int16 ki, int16 kd)
11  {
12      static int16 erroIntegral;
13      static int16 erroAnterior;
14      int16 erro;
15      int16 termo_P;
16      int16 termo_I;
17      int16 termo_D;
18
19      if(reset==true)
20      {
21          // reinicia os valores do controlador
22          erroIntegral=0;
23          erroAnterior=0;
24          *sinalControle=0;
25      }
26      else
27      {
28          // calcula o erro atual
29          erro=referencia-realimentacao;
30
31          // acumula o erro
32          erroIntegral=verificaLimites((int32)erroIntegral+(int32)erro);
33
34          // calcula o termo proporcional
35          termo_P=verificaLimites((int32)kp*(int32)erro);
36
37          // calcula o termo integral
38          termo_I=verificaLimites((int32)ki*(int32)erroIntegral);
39
40          // calcula o termo derivativo
41          termo_D=verificaLimites((int32)ki*(int32)(erro-erroAnterior));
42
43          // calcula a saída do controlador
44          *sinalControle=verificaLimites((int32)termo_P+(int32)termo_I+(int32)termo_D);
45
46          // armazena o erro para a próxima iteração
47          erroAnterior=erro;
48      }
49  }
50
```

Figura 74 – Trecho do código em C++ do experimento 8.

Fonte: Autoria própria.

Após a elaboração do algoritmo do controlador PID em C++ iniciaram-se as primeiras simulações e testes de funcionamento. Estes testes foram realizados ainda no código em C++, e a verificação do funcionamento do algoritmo do controlador PID se deu antes mesmo da geração do código VHDL.

Uma vez verificado o funcionamento do código C++ do experimento a geração do código VHDL se deu através da própria ferramenta Vivado HLS.

Assim, como aconteceu para o PON-HD 1.0, o código VHDL gerado pelo Vivado HLS foi inspecionado e, estando de acordo com o esperado, prosseguiu-se para o processo de síntese do circuito. O código VHDL completo pode ser observado no Apêndice V.

O código VHDL gerado pelo Vivado HLS foi então sintetizado pela ferramenta Vivado 2018.1. A FPGA escolhida para a síntese e para as simulações é, assim como no caso anterior, do modelo XA7A12TCPG238-2L da família Artix-7 fabricada pela Xilinx.

Para verificar o funcionamento do circuito foram realizados alguns testes. Inicialmente verificou-se o esquemático (RTL) do circuito gerado pela ferramenta de síntese. Na sequência, vários conjuntos de dados de entrada foram apresentados ao circuito e as saídas correspondentes foram verificadas, de forma a testar seu correto funcionamento.

A Figura 75 apresenta o bloco de circuito gerado pela síntese do código VHDL para o controlador PID desta implementação.

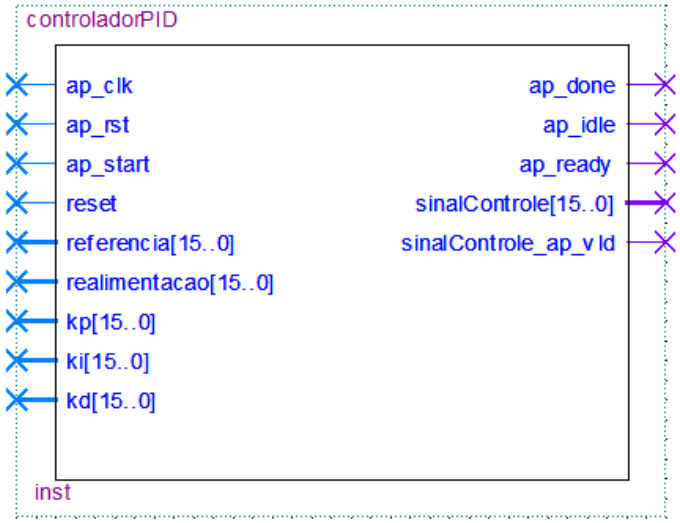

Figura 75 - Bloco gerado pela síntese do código C++ do experimento 8.

Fonte: Autoria própria.

Na figura, é possível observar os sinais de entrada que são: referência, realimentação, Kp, Ki e Kd, todos inteiros com 16 bits. O sinal de *reset* implementado em C++, o sinal de *reset* implementado pelo próprio Vivado HLS. O sinal de *clock* e o sinal de *start*, também são sinais de entrada, porém de apenas 1 bit. Também é possível observar o sinal de saída do controlador com seus 16 bits. O Vivado HLS produz ainda um conjunto de sinais de saída que indicam o estado do circuito, que são: *done*, *idle*, *ready* e saída válida.

Diferentemente da implementação em PON-HD 1.0, a qual produziu uma saída a cada ciclo de *clock*, a implementação em Vivado HLS do controlador PID produz uma saída a cada três ciclos de *clock*. O sinal de saída válida ("sinalControle_ap_vld") indica quando a saída do controlador é válida e pode ser usada nas computações.

4.8.2 Resultados do Experimento 8 (Controlador PID)

Neste experimento ambos as implementações utilizam sinal de *clock*, ou seja, são circuitos sequenciais e assim as comparações de performance são mais precisas.

Ambas as implementações executaram o algoritmo do controlador de forma correta. As estratégias utilizadas, no entanto, foram ligeiramente diferentes. Estas diferenças podem ser observadas nos dados apresentados na Tabela 12. Esta tabela demostra as principais características de performance e utilização de recursos da FPGA apresentado por cada uma das implementações.

Tabela 12 - Características de implementação do experimento 8.

	LUT	FF	DSP	Frequência Máxima	ciclos de *clock*
PON-HD	164	145	3	90.9MHz	6 ciclos de *clock*
Vivado HLS	279	101	3	83.3MHz	3 ciclos de *clock*

Fonte: Autoria própria.

Assim como nos experimentos anteriores, adotou-se na tabela o padrão de nomenclatura da Xilinx. Os dados que compõem a tabela foram obtidos com a ferramenta Vivado, após a síntese do circuito.

Na Tabela 12, é possível observar que a implementação em PON-HD 1.0 utilizou 164 LUTs, enquanto a implementação em Vivado HLS utilizou 279 LUTs. Aparentemente, esta

diferença está relacionada a diferença de latência entre os dois circuitos. Como a implementação em Vivado HLS realiza mais tarefas por ciclo de *clock* é necessário mais *hardware*.

Com relação ao número de registradores (FFs), a implementação em PON-HD 1.0 ocupa um pouco mais *hardware*. Isso acontece, como esperado, porque no PON-HD 1.0, todos os resultados intermediários são armazenados nos *Attibutes*.

Ainda com relação a frequência de operação, a implementação em PON-HD 1.0 leva um pouco de vantagem. Esta diferença também está relacionada a latência. Como a implementação em PON-HD 1.0 realiza mais tarefas por ciclo de *clock*, é natural que sua operação seja mais lenta (menor frequência de operação).

Um resultado bastante interessante é a latência, ou seja, o número de ciclos de *clock* necessários para executar uma tarefa. A implementação em Vivado HLS executa o algoritmo do controlador em 3 ciclos de *clock*. Porém, só apresenta um resultado válido no final do terceiro ciclo, permanecendo a saída inválida nos outros dois ciclos. Já a implementação em PON-HD 1.0 apresenta saídas válidas em todos os ciclos de *clock*, porém necessita de 6 ciclos para executar o algoritmo, ou seja, tem-se um atraso de 6 ciclos mas não se tem estados inválidos na saída. Este maior número de ciclos observados na implementação em PON-HD 1.0, está relacionado ao fato de que os resultados intermediários necessitam ser armazenados nos *Attributes* e, assim sendo, necessitam de uma borda de subida do *clock* para ativar a entrada destes registradores. Ou seja, A implementação em PON possui uma latência inicial de 6 ciclos para a primeira saída, enquanto a implementação em Vivado HLS possui 3, porém, a implementação em PON produz uma saída a cada ciclo enquanto a implementação em Vivado HLS produz uma saída a cada 3 ciclos.

Também é possível observar nos resultados que ambas as implementações utilizaram 3 blocos DSP do *hardware* para realizar as operações matemáticas. Esta característica não está diretamente relacionada com o PON-HD 1.0 e sim com a síntese do circuito realizada pela ferramenta Vivado 2018.1.

4.8.3 Conclusões do Experimento 8 (Controlador PID)

O último experimento apresentado serviu para comparar o PON-HD 1.0 com a ferramenta Vivado HLS em um circuito mais complexo e de caráter mais prático. Novamente, as ferramentas se utilizaram de estratégias diferentes na implementação do algoritmo. A implementação em PON-HD 1.0 necessitou de mais ciclos de *clock* para executar a tarefa,

porém conseguiu trabalhar em uma frequência um pouco maior. Além disso, esta implementação apresentou um novo resultado em sua saída a cada ciclo de *clock,* enquanto a implementação em Vivado HLS apresentou um novo resultado a cada três ciclos de *clock.* A implementação em Vivado, por sua vez, necessitou apenas três ciclos de *clock* para executar o algoritmo. Assim, conclui-se com este experimento que a utilização do ferramental PON-HD 1.0 não gera grande degradação na performance dos circuitos com ele sintetizados, ao menos para a classe de problemas analisada.

4.9 Considerações sobre o capítulo

Com o objetivo de verificar o funcionamento e as características do ferramental PON-HD 1.0, foram realizados e descritos oito experimentos.

Os resultados de forma geral foram positivos, com o PON-HD 1.0 se mostrando uma ferramenta funcional para o desenvolvimento de *hardware* digital. A possibilidade de descrição do circuito em alto nível proporcionada pela linguagem LingPON-HD 1.0 e seu compilador tendem a permitir que desenvolvedores sem grandes conhecimentos de *hardware* desenvolvam circuitos digitais. A performance dos circuitos gerados, por sua vez, não deixou muito a desejar, ao menos nos casos estudados.

É importante lembrar que, de um ponto de vista industrial, o PON-HD 1.0 ainda está em fase experimental de desenvolvimento e certamente muitas melhorias surgirão em versões futuras.

Levando em consideração que este trabalho se propôs a verificar se o PON é apropriado ao desenvolvimento de *hardware* em lógica reconfigurável e considerando ainda que, neste momento, não é pretensão do PON-HD 1.0 ser a melhor ferramenta de síntese de *hardware* em alto nível, pode-se considerar que os experimentos apresentaram resultados bastante positivos neste sentido.

É importante destacar que existem ameaças e imprecisões que podem afetar os resultados dos experimentos. Os fatores humanos devem ser considerados neste contexto, pois os resultados são em certa parte dependentes da habilidade apresentada pelos desenvolvedores, envolvidos nos experimentos, na descrição dos códigos, tanto em PON-HD 1.0 como em outras linguagens de síntese.

5 Conclusões

Este capítulo apresenta as conclusões obtidas com este trabalho de doutorado. Inicialmente, são apresentadas as principais contribuições deste trabalho. Em seguida, é apresentada discussão sobre os resultados obtidos nos experimentos e sua relação com os objetivos traçados para este trabalho. Para finalizar, são discutidas as dificuldades e limitações observadas no PON-HD 1.0 e uma proposta de trabalhos futuros é apresentada.

5.1 Principais contribuições

A proposição e o desenvolvimento de um ferramental para a utilização do PON no desenvolvimento em alto nível de *hardware* digital, através de uma linguagem de alto nível e um conjunto de ferramentas específico é uma contribuição importante deste trabalho. Essa contribuição, chamada de ferramental PON-HD 1.0, é composta pelo *Framework* PON-HD 1.0, pela linguagem LingPON-HD 1.0, pelo compilador PON-HD 1.0 e pelo simulador PON-HD 1.0.

Entretanto, a principal contribuição deste trabalho de doutorado é verificar experimentalmente que o Paradigma Orientado a Notificação (PON) é viável para o desenvolvimento de circuitos em hardware digital. Neste sentido, os resultados dos experimentos demostram que o ferramental PON-HD 1.0 permite descrever, através de regras em alto nível, o comportamento desejado para os circuitos em lógica reconfigurável. Na verdade, por meio de experimentos comparativos em *hardware* digital, foram demostradas propriedades importantes do PON via este ferramental PON-HD 1.0, como nível de abstração, desempenho e paralelismo implícito. Este paralelismo implícito pode ser observado na forma de execução do circuito implementado no *hardware*.

Neste âmbito, a descrição dos algoritmos por meio de regras em alto nível em formato declarativo, propiciado pelo ferramental PON-HD 1.0, permite que desenvolvedores com poucos conhecimentos sobre circuitos digitais e suas ferramentas de síntese utilizem esta plataforma de computação paralela para compor suas aplicações de maneira nativamente paralela, mas sem preocupação técnica a respeito, dado que o paralelismo se dá de maneira implícita. De forma mais ampla, o ferramental PON-HD 1.0 se apresenta como uma nova

alternativa para o domínio da computação paralela onde o desenvolvedor não necessariamente se preocupa com o paralelismo, uma vez que o paralelismo é intrínseco ao paradigma.

Neste contexto, uma importante contribuição deste trabalho é a colaboração do ferramental PON-HD 1.0 para demonstrar as propriedades do próprio PON, as quais não eram possíveis com plenitude nas implementações anteriores realizadas nos ambientes sequenciais Von Neumann, mesmo em ambientes *multi-core*. Pode-se destacar dentre estas propriedades do PON-HD 1.0 o paralelismo de execução de seus elementos e a comunicação através de notificações pontuais, bem como a coesão e o desacoplamento apresentados pelos elementos que compõe este paradigma.

Implementações anteriores do PON em *hardware*, como o CoPON e o ArqPON, já permitiam demostrar parte dessas propriedades, mas não em plenitude pois nem tudo era necessariamente paralelo naquelas arquiteturas. Assim, o ferramental PON-HD 1.0 as mostra de maneira mais pontual sem as limitações das materializações anteriores, inclusive do PON-HD Prototipal, que serviu apenas para verificar se os componentes PON poderiam ser ou não executados em FPGA, sem maiores refinamentos de conformação do PON em hardware digital e refinamento de estudos comparativos.

Um exemplo de refinamento de estudos, no tocante ao PON-HD 1.0 proposto neste trabalho, é o seu posicionamento em relação a outras ferramentas de alto nível pertinentes no âmbito de *hardware* digital. Considerando a classificação das ferramentas de desenvolvimento em alto nível já apresentada anteriormente (MEEUS *et al*., 2012; NANE *et al*., 2016), bem como os critérios utilizados por estes autores em sua classificação (facilidade de implementação, nível de abstração, tipos de dados, ajuste do *design*, capacidade de verificação, qualidade dos resultados, documentação e curva de aprendizado), pode-se classificar o ferramental PON-HD 1.0 como ferramenta de desenvolvimento efetivamente em alto nível.

De forma a comparar as características do ferramental PON-HD 1.0 com as características de outras ferramentas de síntese em alto nível é apresentada a Tabela 13. Nesta tabela, quanto maior o número de estrelas (asteriscos) maior é a qualidade da ferramenta no critério avaliado.

Nesta tabela foi incluído o ferramental PON-HD 1.0, com suas características estimadas segundo os critérios de Meeus *et al*., (2012), mencionados no capítulo 2. Uma justificativa da pontuação para cada um dos critérios é apresentada nos parágrafos seguintes. Porém, é importante destacar que estes resultados são apenas uma estimativa estando sujeitos ao entendimento pessoal e experiências práticas.

Tabela 13 - Classificação do PON-HD 1.0 em relação as ferramentas de alto nível.

	Facilidade de implementação (Algoritmos C, C++)	Nível de abstração	Tipos de dados	Ajuste do *design*	Capacidade de verificação	Qualidade dos resultados	Documentação	Curva de aprendizado
AccelDSP	*****	****	*****	****	****	**	***	*****
Agility Compiler	****	***	*	***	*	**	****	**
AutoPilot	*****	*****	****	*****	*****	*****	*****	*****
BlueSpec	***	***	*	*	-	*****	****	**
Catapult C	*****	*****	****	*****	*****	****	*****	*****
Compaan	****	*****	*****	****	*	*	**	***
C to Silicon	****	****	****	**	****	***	****	*
CyberWorkBench	****	*****	****	*****	****	*****	***	****
DK Design	***	***	****	**	***	*	***	**
Impulse CoDeveloper	**	***	****	***	*	*	*	***
ROCCC	*	****	*	***	*	*	****	**
Synphony C	*****	*****	****	***	*****	***	****	****
PON-HD 1.0	**	*****	*	*	***	*****	*	*****

Fonte: Adaptado de Meeus *et al*., (2012).

A facilidade de utilização ou implementação é uma característica importante no ferramental PON-HD 1.0, uma vez que ele organizou a implementação dos circuitos por meio da utilização de elementos pré-definidos e da descrição por meio de regras em alto nível. O desenvolvedor é aliviado da tarefa de escrever o código em linguagem de descrição de *hardware,* pois, o ferramental PON-HD 1.0 fornece a maior parte do código nos arquivos que definem os seus elementos. Com relação a classificação segundo Meeus et al., (2012), é importante destacar que o ferramental PON-HD 1.0 não favorece o reaproveitamento de códigos previamente escritos em linguagens como C ou C++, exigindo uma transcrição dos mesmos para LingPON-HD 1.0. É um tanto injusto mensurar a facilidade de utilização apenas pelo reaproveitamento de códigos escritos em C ou C++, porém este foi o critério adotado pelos autores do trabalho de classificação.

A linguagem LingPON-HD 1.0 utilizada no ferramental PON-HD 1.0 é considerada fácil de aprender, isto porque o desenvolvedor precisaria apenas se habituar a usar um conjunto relativamente pequeno de elementos, com uma sintaxe simples, herdando características das linguagens ditas declarativas. Desta forma o desenvolvedor não necessita ter conhecimentos profundos nas linguagens de descrição de *hardware* e, assim, o PON-HD 1.0 fornece um nível de abstração adequado ao objetivo proposto para este ferramental. A expressividade propiciada pela descrição do circuito através de regras em alto nível é fator importante neste critério.

Pode-se afirmar ainda que o PON-HD 1.0 é uma solução genérica, pois não foi desenvolvida para resolver um tipo específico de problema. As características herdadas do PON e a linguagem LingPON-HD 1.0 favorecem esta característica.

Uma característica importante das ferramentas de síntese em alto nível é o conjunto de tipos de dados suportados. Atualmente o PON-HD 1.0 suporta dados inteiros e booleanos. No entanto, é consideravelmente simples, ainda que trabalhoso, editar o código VHDL do *Framework* PON-HD 1.0 para adicionar suporte para outros tipos de dados, como ponto flutuante por exemplo.

A possibilidade de ajustes no *design* é ainda limitada no PON-HD 1.0, pois não é possível ajustar a ferramenta de forma a otimizar os resultados para velocidade ou consumo de recursos por exemplo. Porém, como o resultado da compilação do código em LingPON-HD 1.0 é um conjunto de arquivos VHDL, tais ajustes de otimização podem ser realizados diretamente no compilador VHDL, quando do momento da síntese do circuito.

Em tempo, as ferramentas de desenvolvimento de *hardware* devem fornecer formas de testar as aplicações. No ferramental PON-HD 1.0, o simulador PON-HD 1.0 desenvolvido no âmbito deste trabalho é responsável por esta tarefa. É possível testar o comportamento do programa sem a necessidade de compilar e testar o projeto em uma FPGA.

Os arquivos VHDL gerados pelo ferramental podem ainda ser verificados pelas ferramentas disponibilizadas para este fim pelos fabricantes de *hardware*. Estas facilidades demostraram-se importantes na diminuição do tempo de desenvolvimento de soluções corretas para diferentes problemas propostos.

Ainda é necessário um pouco de trabalho no que diz respeito à documentação do ferramental PON-HD 1.0, pois como este ferramental esteve até recentemente em ativo desenvolvimento e teste, o mesmo não possui uma documentação definitiva, sendo este presente trabalho e seus apêndices um primeiro esforço de documentação. Por sua vez, a linguagem LingPON-HD 1.0 por se tratar de uma extensão da linguagem LingPON, encontra nesta outras documentações e um conjunto de trabalhos publicados por outros membros do grupo de pesquisa do PON.

Outrossim, as características de performance apresentadas pelas aplicações desenvolvidas durante os experimentos com o ferramental PON-HD 1.0, assemelhando-se a performance de aplicações desenvolvidas em outras linguagens ou outras ferramentas, sugerindo que o método proposto e a qualidade de seus resultados são, no mínimo, adequados a descrição em alto nível de circuitos em *hardware* digital.

A independência de arquitetura caracterizada no PON-HD 1.0 pela utilização do VHDL em seus componentes é ainda uma vantagem. O resultado da compilação da linguagem LingPON-HD 1.0, no ferramental PON-HD 1.0, é um código VHDL completo em *Framework* PON-HD 1.0 o qual pode ser compilado para qualquer arquitetura, plataforma ou fabricante.

A título de comparação a Tabela 14 apresenta as principais ferramentas de desenvolvimento em alto nível em utilização no momento (NANE *et al*. 2016), apresentando seu fabricante, suas linguagens de entrada e saída, a possibilidade de verificação ou simulação do *design* e o suporte ou não a dados do tipo ponto flutuante, isso segundo Nane *et al*. (2016).

Tabela 14 - Principais ferramentas de síntese em alto nível e suas características.

Ferramenta	Proprietário	Entrada	Saída	Verificação / Simulação	Ponto Flutuante
eCXite	Y Explorations	C	VHDL / Verilog	Sim	Não
CoDeveloper	Impulse Accelerated	Impulse-C	VHDL / Verilog	Sim	Sim
Catapult-C	Calypto Design Systems	C / C++ / SystemC	VHDL / Verilog / SystemC	Sim	Não
Cynthesizer	FORTE	SystemC	Verilog	Sim	Sim
Bluespec	BlueSpec Inc.	BSV	System Verilog	Não	Não
CHC	Altium	Subconjunto C	VHDL / Verilog	Não	Sim
CtoS	Cadence	SystemC / TLM / C++	Verilog / SystemC	Não	Não
DK Design Suite	Mentor Graphics	Handel-C	VHDL / Verilog	Não	Não
GAUT	U. Bretagne	C / C++	VHDL	Sim	Não
MaxCompiler	Maxeler	MaxJ	RTL	Não	Sim
ROCCC	Jacquard Comp.	Subconjunto C	VHDL	Não	Sim
Synphony C	Synopsys	C / C++	VHDL / Verilog / SystemC	Sim	Não
Cyber-WorkBench	NEC	BDL	VHDL / Verilog	Sim	Sim
LegUp	U. Toronto	C	Verilog	Sim	Sim
Bambu	PoliMi	C	Verilog	Sim	Sim
DWARV	TU. Delft	Subconjunto C	VHDL	Sim	Sim
VivadoHLS	Xilinx	C / C++ / SystemC	VHDL / Verilog / SystemC	Sim	Sim
PON-HD 1.0	**UTFPR**	**LingPON-HD 1.0**	**VHDL**	**Sim**	**Não (Ainda)**

Fonte: Adaptado de Nane *et al.* (2016).

O ferramental PON-HD 1.0 foi adicionado a esta tabela permitindo uma comparação deste ferramental com outras ferramentas de desenvolvimento em alto nível disponíveis no mercado.

5.2 Discussão dos resultados à luz dos objetivos

Levando em consideração que o objetivo geral deste trabalho é o desenvolvimento e análise do Paradigma Orientado a Notificação (PON) para a geração de VHDL, visando a subsequente síntese de *Hardware* Digital (HD) dito notificante por meio da proposição de ferramental PON-HD específico, bem como a verificação das vantagens e desvantagens das propriedades intrínsecas do PON na qualidade dos circuitos digitais obtidos, isto por meio de experimentos comparativos com outras abordagens de projeto de *Hardware* Digital em alto nível, pode-se concluir que os resultados apresentados pelos experimentos confirmam a relevância das contribuições deste trabalho.

O ferramental PON-HD 1.0 se mostrou funcional, empregando os fundamentos do PON na síntese em alto nível de *hardware* digital. Algumas propriedades do PON, como paralelismo de execução, por exemplo, foram observadas experimentalmente. Os experimentos realizados também posicionaram o PON-HD 1.0 em relação a implementações em VHDL e em Vivado HLS. Os resultados dos experimentos mostraram ainda que o PON é apropriado ao desenvolvimento de circuitos em *hardware.*

Uma discussão mais detalhada dos resultados a luz de cada um dos objetivos específicos é apresentada a seguir, iniciando pelo primeiro objetivo específico.

O primeiro objetivo específico deste trabalho trata de realizar revisão dos aspectos técnicos e teóricos a respeito dos dispositivos de lógica reconfigurável e suas técnicas de programação, das chamadas ferramentas de síntese em alto nível em geral, dos paradigmas de desenvolvimento e principalmente do Paradigma Orientado a Notificações a fim de identificar e caracterizar a lacuna existente referente à síntese em alto nível e como o PON pode ser empregado para o desenvolvimento em alto nível de circuitos em lógica reconfigurável.

As discussões apresentadas no capítulo 2 mostraram uma carência no mercado de ferramentas que combinem ao mesmo tempo características de facilidade de utilização e expressividade, facilitando assim seu aprendizado e utilização e ao mesmo tempo permitindo desenvolver circuitos com boa performance. Neste sentido, observando os resultados dos experimentos, nota-se que estes sugerem que o ferramental PON-HD 1.0 permite a descrição em alto nível de aplicações destinadas ao desenvolvimento em lógica reconfigurável. Desta forma é possível conceber que o ferramental PON-HD 1.0 como um todo permite um caminho para suprir ao menos parte desta lacuna.

O segundo objetivo específico visa desenvolver um conjunto de técnicas e ferramentas que permitam a utilização do PON no desenvolvimento em alto nível de circuitos em lógica reconfigurável, por meio de VHDL a luz das notificações que permita a partir deste gerar circuitos ditos notificantes. Este conjunto de ferramentas denominado PON-HD 1.0 será composto por um f*ramework* para VHDL chamado *Framework* PON-HD 1.0, por uma linguagem chamada LingPON-HD 1.0 e seu compilador e por um simulador para o PON-HD 1.0.

Observando os resultados obtidos nos vários experimentos nota-se que o ferramental que compõe o PON-HD 1.0 cumpre este objetivo. O *Framework* PON-HD 1.0 fornece uma base de componentes que, com a ajuda da linguagem LingPON-HD 1.0, respectivo compilador PON-HD 1.0 e correlato simulador PON-HD 1.0, permitem descrever e mesmo testar de antemão de forma mais fácil e rápida circuitos em lógica reconfigurável, se comparado a síntese diretamente em VHDL.

Uma análise mais profunda dos circuitos gerados com o ferramental PON-HD 1.0 permite observar que o comportamento destes circuitos reflete a estrutura de notificações característica do PON.

O terceiro objetivo específico visa tornar a implementação do PON em *hardware* digital tecnologicamente viável inclusive por circuitaria resultante com *clock* e utilização de recursos apropriados, bem como pela programação de mais alto nível, isto por meio da programação em regras, na linguagem de programação LingPON-HD 1.0 e seu compilador, permitindo que desenvolvedores sem experiência em síntese de hardware desenvolvam circuitos digitais em alto nível.

Os experimentos apresentaram consideráveis indícios que sinalizam que o ferramental PON-HD 1.0 colabora com o desenvolvimento de circuitos em *hardware* digital, permitindo a utilização de entidades coesas e desacopladas na descrição dos circuitos e o desenvolvimento declarativo em alto nível. A linguagem LingPON-HD 1.0 tem grande importância neste sentido pois propicia expressividade por meio da orientação a regras. Adicionalmente o autor observou nos experimentos executados uma significativa diminuição no tempo de desenvolvimento dos circuitos sintetizados, particularmente se comparado com o desenvolvimento realizado diretamente em VHDL.

O quarto objetivo específico trata de realizar testes e experimentos para verificar se o PON é apropriado ao desenvolvimento de circuitos em hardware digital, por meio da geração de VHDL intermediário orientado a notificações.

Em todos os experimentos foram realizadas implementações de circuitos em *hardware* digital utilizando o PON-HD 1.0. Levando em consideração que o PON-HD 1.0 é uma materialização do PON na forma de um ferramental para desenvolvimento de circuitos em *hardware* digital e observando através dos resultados dos experimentos que os circuitos desenvolvidos com este ferramental são completamente funcionais, é possível concluir que o PON é apropriado para o desenvolvimento de circuitos em *hardware* digital.

O quinto objetivo específico visa verificar se a performance do PON-HD 1.0 é apropriada quando comparada a outras abordagens de síntese de *hardware* em alto nível, particularmente o VHDL e o Vivado HLS (*Vivado high level synthesis da Xilinx*), em termos de velocidade de operação e utilização de recursos da FPGA.

Os resultados dos experimentos demonstram que o PON-HD 1.0 apresenta algumas limitações de performance, principalmente para circuitos predominantemente combinacionais, porém, de forma geral, a utilização do ferramental PON-HD 1.0 não gera grande degradação na performance dos circuitos com ele sintetizados, ao menos para a classe de problemas analisada.

A performance dos circuitos implementados com o ferramental PON-HD 1.0 pode ser considerada polo menos satisfatória. Com relação a frequência de *clock*, a latência e a utilização de recursos, quando comparadas as implementações em PON-HD 1.0 e em outras ferramentas, não são observadas discrepâncias tão significativas a ponto de inviabilizar a utilização do PON-HD 1.0 como ferramenta de desenvolvimento de circuitos em *hardware* digital, ao menos para os experimentos realizados.

O quinto e último objetivo específico está relacionado a apresentar os resultados obtidos à comunidade acadêmica para avaliação e discussão dos mesmos.

Neste sentido foram realizados os seguintes trabalhos.

- Publicação de um artigo nos anais do XLII Congresso Brasileiro de Educação em Engenharia, realizado em Juiz de Fora, MG, com o título “O Simulador de Robô V-Rep no Ensino das Técnicas de Controle de Robôs Autônomos” (KERSCHBAUMER, 2014). Este artigo não tem relação direta com a implementação do PON em *hardware*, porém nele são relatados experimentos com a ferramenta V-REP que foi utilizada posteriormente no experimento do controlador de robô hexápode.
- Publicação de um artigo no congresso 12º CBIC (Brazilian) Congress on Computational Intelligence com o título “Paradigma Orientado a Notificações para a Síntese de Lógica Reconfigurável” (KERSCHBAUMER *et al.*, 2015). Este artigo apresentou os primeiros resultados obtidos com o *framework* PON-HD 1.0.

- Coautoria na publicação de um artigo na revista SODEBRAS com o título “Notification Oriented Paradigm to Digital Hardware” (PORDEUS *et al.*, 2016). Este artigo apresentou os resultados da implementação do ordenador de dados com o *framework* PON-HD 1.0.
- Apresentação de um minicurso sobre o Paradigma Orientado a Notificações no VII Simpósio Brasileiro de Engenharia de Sistemas de Computação (SBESC), realizado em Curitiba, de 7 a 10 de novembro de 2017.
- Publicação de um artigo na revista Journal of Circuits, Systems, and Computers, com o título “Notication-Oriented Paradigm to Implement Digital Hardware” (KERSCHBAUMER *et al.*, 2018a). Este artigo tem maior relevância que os anteriores a apresentou os primeiros resultados da utilização do *framework* PON-HD 1.0 na implementação de *hardware* digital em alto nível.
- Publicação de um artigo na revista IEEE Latin America Transactions, com o título “A Tool for Digital Circuits Synthesis Based on Notification Oriented Paradigm” (KERSCHBAUMER *et al.*, 2018b). Assim como o anterior, este artigo tem grande relevância em relação aos trabalhos realizados, pois nele são apresentados os resultados obtidos com a utilização do ferramental PON-HD 1.0, como um todo, no desenvolvimento em alto nível de circuitos em *hardware* digital.

Existem ainda dois artigos em desenvolvimento, o primeiro irá apresentar os resultados do experimento da esteira transportadora, onde os acadêmicos utilizaram o PON-HD 1.0 no desenvolvimento de seu projeto. O segundo artigo em desenvolvimento irá apresentar os resultados dos experimentos onde o PON-HD 1.0 é comparado a ferramenta Vivado HLS.

De forma geral, levando em consideração os resultados obtidos nos experimentos, conclui-se que o ferramental PON-HD 1.0 se mostra funcional no desenvolvimento de *hardware* digital em alto nível. Algumas das principais características do PON se mostraram presentes no ferramental PON-HD 1.0, como o nível de abstração (observado através da expressividade da descrição dos circuitos por meio de regras em alto nível), desempenho apropriado (observado nos experimentos de forma geral) e paralelismo intrínseco (observado nos circuitos efetivamente sintetizados).

Com relação a viabilidade do uso do PON no desenvolvimento em alto nível de circuitos em lógica reconfigurável, pode-se afirmar que o modelo de execução do PON, bem como suas características de coesão e desacoplamento encontram no *hardware* reconfigurável uma plataforma propícia a sua implementação. O modelo adotado no PON de representação

dos algoritmos através de regras permite naturalmente a expressividade em alto nível. Além disso, os resultados dos experimentos demostraram que os algoritmos implementados através do ferramental PON-HD 1.0 em lógica reconfigurável funcionam corretamente e não apresentam grandes perdas de performance. Levando isso tudo em consideração, conclui-se que o PON é viável para o desenvolvimento em alto nível de circuitos em lógica reconfigurável.

5.3 Dificuldades e limitações

Analisando a estrutura proposta para o ferramental PON-HD 1.0, observa-se que ele não é ainda otimizado para resolução de todos os tipos de problemas. Devido à forma como o PON é estruturado, a sua utilização em aplicações que necessitam manipulação de fluxos de dados, como em filtros digitais ou em processamento de imagem é até possível, porém é bastante complexa. Como ainda não estão implementados na linguagem LingPON e consequentemente na LingPON-HD 1.0 *Attributes* na forma de vetores, a manipulação deste tipo de dados se torna complexa e trabalhosa, ainda que possível usando diretamente o *Framework* PON-HD 1.0.

De fato, a implementação atual do ferramental PON-HD 1.0 como um todo é mais apropriada para problemas de controle discreto ou orientado a eventos, onde a mudança de estado de um elemento exige uma tomada de decisão e a consequente alteração do estado de outros elementos. Esta característica é intrínseca do próprio PON, e permanece presente em sua implementação em *hardware*.

Outra questão relacionada a estrutura do PON é a necessidade de se registrar os dados nos *Attributes*. Em circuitos altamente combinacionais, a necessidade de armazenas resultados intermediários, entradas e saídas nos *Attributes* causa perda de performance e aumento do uso de recursos na FPGA. Esta característica, apesar de prejudicar a performance de alguns circuitos desenvolvidos com o PON-HD 1.0, não impossibilita sua utilização nesta classe de circuitos.

Além disso, a versão atual do ferramental PON-HD 1.0 é limitada com relação aos tipos de dados. Atualmente são aceitos pelo compilador PON-HD 1.0 (do LingPON-HD 1.0) apenas *Attributes* inteiros com qualquer número de bits e *Attributes* do tipo booleano de um bit. O suporte a outros tipos de dados, principalmente a ponto flutuante, faz-se necessário para muitas aplicações. Por outro lado, é natural que o ferramental PON-HD 1.0 ainda tenha essas limitações, por ainda estar em desenvolvimento.

Outrossim, uma dificuldade apresentada pelo ferramental PON-HD 1.0 e que está relacionada às características do próprio PON, é a dificuldade de se utilizar memória RAM (interna ou externa a FPGA) no armazenamento e tratamento dos *Attributes*. Como um *Attribute* deve notificar outros elementos do circuito e como todos os *Attributes* podem potencialmente ser acessados ao mesmo tempo (paralelamente), a estrutura da memória RAM convencional não é adequada. A utilização de registradores para o armazenamento dos *Attributes* permite a execução realmente paralela das aplicações, mas limita a quantidade de dados que pode ser armazenada ao número de registradores disponível na FPGA.

Estas são as principais limitações identificadas durante o desenvolvimento do ferramental PON-HD 1.0. Algumas destas limitações podem ser superadas com a implementação de melhorias no *Framework* PON-HD 1.0, na linguagem LingPON-HD e no compilador PON-HD 1.0, e serão consideradas em trabalhos futuros. A limitação relacionada da utilização da memória RAM para o armazenamento dos dados dos *Attributes* é mais difícil de se contornar, pois advêm da forma como o PON é estruturado.

5.4 Trabalhos futuros

Nesta seção são apresentadas sugestões de trabalhos futuros. Estas sugestões são continuações diretas das pesquisas realizadas neste trabalho ou ainda, sugestões derivadas das conclusões e das limitações apresentadas nas seções passadas.

Melhorias importantes para o desenvolvimento do ferramental PON-HD 1.0, mas que são mais difíceis de serem implementadas são: o uso de memória RAM interna da FPGA para o armazenamento do *Attributes* e o interfaceamento com memória externa a FPGA. O desenvolvimento de um novo tipo de memória intrinsicamente notificante seria ideal para a implementação do PON em *hardware* digital.

A implementação de novos tipos de dados como por exemplo ponto flutuante e cadeias de caracteres também são necessárias para ampliar o leque de aplicações do ferramental PON-HD 1.0. Da mesma forma o suporte a mais operadores matemáticos é fundamental para facilitar a utilização desta ferramenta.

Também é importante a elaboração de experimentos que verifiquem o consumo de energia das aplicações desenvolvidas em PON-HD 1.0 se comparadas as mesmas aplicações desenvolvidas com ferramentas tradicionais de síntese de *hardware* em alto nível. Desta forma será possível verificar se o PON-HD 1.0 propicia alguma redução no consumo de energia do

circuito. Porém, para a realização destes experimentos é necessária a aquisição ou o acesso a uma plataforma de *hardware* com os sensores apropriados a este fim.

São ainda necessárias melhorias na linguagem LingPON como um todo, como por exemplo o suporte a vetores de *Attributes* e a inclusão de bibliotecas. Estas melhorias dependem do grupo de pesquisa do PON como um todo, pois afetam todas as plataformas de execução do PON e não apenas sua implementação em *hardware* reconfigurável. Estes esforços já estão em andamento através dos trabalhos realizados por Adriano Francisco Ronszcka (RONSZCKA, 2018).

Uma nova área de pesquisa vislumbrada durante as pesquisas realizadas neste trabalho é a implementação do PON em *hardware* de forma assíncrona, ou seja, sem o uso de *clock*. Muitas das limitações do PON-HD 1.0 poderiam ser superadas se uma implementação assíncrona deste paradigma em *hardware* for possível. É claro que o sincronismo e a resolução de conflitos em tal implementação não seriam tarefa trivial.

6 Referências

AWAD, Mariette. FPGA supercomputing platforms: a survey. In: **Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on.** IEEE, 2009, p. 564-568.

BAILEY, Donald. G. Advantages and Limitations of High Level Synthesis for FPGA Based Image Processing. **ICDSC '15 Proceedings of the 9th International Conference on Distributed Smart Cameras.** Seville, Spain. Doi:10.1145/2789116.2789145. 2015

BANASZEWSKI, Roni. F.; STADZISZ, Paulo. C.; TACLA, Cesar. A.; SIMÃO, Jean. M. Notification Oriented Paradigm (NOP): A Software Development Approach based on Artificial Intelligence Concepts. **Logic Applied to Technology,** 2007.

BANASZEWSKI, Roni. F. **Paradigma Orientado a Notificações: Avanços e Comparações.** Dissertação de Mestrado. CPGEI, UTFPR. Curitiba, Brasil, 2009.

BARRETO, Wagner R. M. **Notification Oriented Paradigm in the Context of Distributed Systems Aplicação em *Framework* 1.0 NOP Java.** Mestrando PPGCA/UTFPR, 2016. Disciplina sobre Paradigma Orientado a Notificações (PON), CPGEI-PPGCA/UTFPR (Prof. J. M. Simão & Prof. H. Panetto [visitante CPGEI & UL-França]), Curitiba - PR, Brasil, 2016.

BARRETO, W. R. M. ; VENDRAMIN, A. C. B. K. ; SIMÃO, J. M. . Notification Oriented Paradigm for Distributed Systems. **In: Computer on the Beach 2018,** Florianópolis - SC. Anais Computer on the Beach. 2018.

BATISTA, Márcio. V.; BANASZEWSKI, Roni. F.; RONSZCKA, Adriano. F.; VALENCA, Glauber. Z.; LINHARES, Robson. R.; STADZISZ, Paulo. C.; TACLA, Cesar. A.; SIMAO, Jean. M. Uma Comparação entre o Paradigma Orientado a Notificações (PON) e o Paradigma Orientado a Objetos (POO) realizado por meio da implementação de um Sistema de Vendas. Em: **III Congreso Internacional de Computación y Telecomunicaciones - COMTEL**, 2011, Lima, Perú. Memoria: COMTEL 2011. Lima, Perú : Fondo Editorial de la UIGV, p. 53-56, 2011.

BEEL, Joeran; GIPP, Bela; LANGER, Stefan; GENZMEHR Marcel. Docear: An Academic Literature Suite for Searching, Organizing and Creating Academic Literature**. In Proceedings of the 11th annual international ACM/IEEE Joint Conference on Digital Libraries** (JCDL'11), pages 465–466. ACM, 2011. doi: 10.1145/1998076.1998188.

BELMONTE, Danilo.; SIMÃO, Jean. M.; STADZISZ, Paulo. C. Proposta de um Método para Distribuição da Carga de Trabalho Usando o Paradigma Orientado a Notificações (PON). **Revista SODEBRAS**, 7(84), p. 10-17, 2012.

BELMONTE, Danilo. **Método para Distribuição da Carga de Trabalho dos *Softwares* PON em Multicore.** Trabalho de Qualificação de Doutorado, CPGEI, UTFPR. Curitiba, Brasil, 2012.

BELMONTE, Danilo.; LINHARES, Robson. R.; STADZISZ, Paulo C.; SIMÃO, Jean. M. A new Method for Dynamic Balancing of Workload and Scalability in Multicore Systems. **IEEE Latin America Transactions,** Vol. 14, Issue 7, Jul 2016 Pg. 3335-3344. ISSN: 1548-0992. 2016. DOI: 10.1109/TLA.2016.7587639 Paper aceito em 27/05/2016.

BEN-KIKI, Oren; EVANS, Clark; DÖT NET, Ingy; **YAML Ain't Markup Language (YAML™) Version 1.2**. 2009. Disponível em: http://www.yaml.org/spec/1.2/spec.html

BLUESPEC, Inc; **BSV High-Level HDL – Bluespec** 2017. Disponível em: http://bluespec.com/54621-2/

BORKAR, Shekhar; CHIEN, Andrew A. The future of microprocessors. **Communications of the ACM**, v. 54, n. 5, p. 67-77, 2011.

BUNKER, A.; GOPALAKRISHNAN, G.; MCKEE, S. A. Formal *hardware* specification languages for protocol compliance verification. **ACM Trans. Des. Autom. Electron. Syst., ACM**, New York, NY, USA, v. 9, p. 1–32, January 2004. ISSN 1084-4309. Disponível em: <http://doi.acm.org/10.1145/966137.966138>.

COUSSY, Philippe; GAJSKI, Daniel D.; MEREDITH, Michael; *et al*. An introduction to high-level synthesis**. IEEE Design & Test of Computers**, v. 26, n. 4, p. 8-17, 2009.

CROSBIE, Roy. Using field-programmable gate arrays for high-speed real-time simulation. **International Journal of Modeling, Simulation, and Scientific Computing,** v. 1, n. 01, p. 99-115, 2010.

EVANS, Clark C. **YAML Ain't Markup Language.** 2017. Disponível em: < http://yaml.org/> Acesso em: 27 set. 2017, 11:20.

FAN, Yuehong; WINKEL, Casey; KULKARNI, Devdatta. Analytical Design Methodology for Liquid Based Cooling Solution for High TDP CPUs. **17th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm).** San Diego, CA, USA. DOI: 10.1109/ITHERM.2018.8419562. 2018.

FERREIRA, Cleverson A. **Linguagem e compilador para o paradigma orientado a notificações (PON): avanços e comparações.** 2015. Dissertação de Mestrado, Programa de Pós-Graduação em Computação Aplicada (PPGCA), UTFPR. Curitiba, Brasil, 2015.

FORGY, Charles L. RETE: A fast algorithm for the many pattern/many object pattern match problem. **Artificial Intelligence,** vol. 19, pg 17 – 37 1982.

GABBRIELLI, Maurizio; MARTINI, Simone. **Programming Languages: Principles and Paradigms.** Springer Science & Business Media, 2010.

GEORGOPOULOS, Konstantinos; CHRYSOS, Grigorios; MALAKONAKIS, Pavlos; NIKITAKIS, Antonis; TAMPOURATZIS, Nikos; DOLLAS, Apostolos; PNEVMATIKATOS, Dionisios; PAPAEFSTATHIOU, Yannis. An Evaluation of Vivado HLS for Efficient System Design. **2016 International Symposium ELMAR.** Zadar, Croatia. DOI: 10.1109/ELMAR.2016.7731785. 2016.

GUPTA, Gagan; SOHI, Gurindar S. Dataflow execution of sequential imperative programs on multicore architectures. In: **Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture.** ACM, 2011, p. 59-70.

HEMATIAN, A., CHUPRAT, S., MANAF, A. A., PARSAZADEH, N. Zero-Delay FPGA-Based Odd-Even Sorting Network. **IEEE Symposium on Computers & Informatics (ISCI)** IEEE, Langkawi, Malaysia, 2013. DOI: 10.1109/ISCI.2013.6612389

HENZEN, Alexandre F. **Portabilidade do *framework* PON de C++ standard para C# e Java. *Framework* NOP/PON C++ 1.0/1.1**. Doutorando CPGEI/UTFPR, 2015. Disciplina sobre Paradigma Orientado a Notificações (PON), CPGEI-PPGCA/UTFPR (Prof. J. M. Simão), Curitiba - PR, Brasil, 2015.

HUGHES, C., HUGHES, T. **Parallel and Distributed Programming Using C++**. Addison Wesley, 2003.

INTEL CORPORATION. **Intel Quartus Prime Pro Edition Handbook.** 2017. Disponível em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/qts/qts-qpp-handbook.pdf> Acesso em: 16 ago. 2017, 9:10.

JASINSKI, Ricardo. P. *Framework* para Geração de *Hardware* em VHDL a Partir de Modelos em PON (Paradigma Orientado a Notificações). **Relatório da disciplina de Lógica Reconfigurável por *Hardware*.** Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial. Universidade Tecnológica Federal do Paraná, 2012.

KERSCHBAUMER, Ricardo; LINHARES, Robson. R.; SIMÃO, Jean. M.; STADZISZ, Paulo. C; LIMA, Carlos. R. E. Notication-Oriented Paradigm to Implement Digital Hardware. **Journal of Circuits, Systems, and Computers.** Vol. 27, No. 8 World Scientic DOI: 10.1142/S0218126618501244 2018a.

KERSCHBAUMER, Ricardo; SIMÃO, Jean. M.; FABRO, João. A.; ERIG LIMA, Carlos. R.; LINHARES, Robson. R. A Tool for Digital Circuits Synthesis Based on Notification Oriented Paradigm. **IEEE Latin America Transactions**, v. 16, n. 6, p. 1574-1586, 2018b.

KERSCHBAUMER, Ricardo; SIMÃO, Jean. M.; LINHARES, Robson. R.; STADZISZ, Paulo. C; LIMA, Carlos. R. E. Paradigma Orientado a Notificações para a Síntese de Lógica Reconfigurável. **12. CBIC (Brazilian) Congress on Computational Intelligence.** Curitiba, Brazil. ISBN 9788569972006 UTFPR, 2015.

KERSCHBAUMER, Ricardo. **Paradigma Orientado a Notificações para Síntese de Lógica Reconfigurável.** Trabalho de Qualificação de Doutorado. Defendido em banca dia 7 de novembro de 2017. Banca composta por: Prof. Dr. Volnei Antônio Pedroni, Prof. Dr. Antônio Augusto Medeiros Fröhlich e Prof. Dr. Marco Aurélio Wehrmeister. CPGEI, UTFPR. Curitiba, Brasil, 2017.

KERSCHBAUMER, R.; LIMA, C. R. E.; SIMAO, J. M. O Simulador de Robô V-Rep no Ensino das Técnicas de Controle de Robôs Autônomos. In: **XLII Congresso Brasileiro de Educação em Engenharia,** 2014, Juiz de Fora - MG. Anais do Cobenge. v. 1. p. 1-10. 2014.

KOCUR, Michal; KOZAK, Stefan; DVORSCAK, Branislav. Design and Implementation of FPGA - digital based PID controller**. IEEE - Proceedings of the 2014 15th International Carpathian Control Conference (ICCC),** Velke Karlovice, Czech Republic. DOI: 10.1109/CarpathianCC.2014.6843603, 2014.

KOSSOSKI, Clayton. **Proposta de um método de teste para processos de desenvolvimento de *software* usando o Paradigma Orientado a Notificações**, Dissertação de Mestrado, CPGEI, UTFPR, 2015. Orientadores: Prof. P. C. Stadzisz e Prof. J. M. Simão.

KOSSOSKI, Clayton; SIMÃO, Jean. M.; STADZISZ, Paulo. C. Introdução ao teste funcional de *software* no Paradigma Orientado a Notificações. **COMTEL - Internacional Congress of Computation and Telecomunications** (V Congreso Internacional de Computación y Telecomunicaciones), Lima, Peru, 2014.

LEFFINGWELL, Dean; WIDRIG, Don. **Managing Software Requirements: A Use Case Approach**: Addison Wesley, 2003.

LINHARES, R. Robson.; RONSZCKA, Adriano. F.; VALENCA, Glauber. Z.; BATISTA, Márcio. V.; WITT, Fernando. A.; LIMA, Carlos. R. E.; SIMAO, Jean. M.; STADZISZ, Paulo. C. Comparações entre o Paradigma Orientado a Objetos e o Paradigma Orientado a Notificações sob o contexto de um simulador de sistema telefônico. En: III **Congreso Internacional de Computación y Telecomunicaciones - COMTEL**, 2011, Lima, Perú. Memoria: COMTEL 2011. Lima, Perú : Fondo Editorial de la UIGV, 2011. p. 103-106.

LINHARES, Robson R.; RENAUX, Douglas P. B.; SIMÃO, Jean M.; *et al*. Evaluation of the Notification Oriented Paradigm Applied to Real-Time Systems. In: **Computing Systems Engineering (SBESC), 2014 Brazilian Symposium on**. IEEE, 2014, p. 132-137.

LINHARES, Robson R.; SIMÃO, Jean M.; STADZISZ, Paulo C. NOCA - A Notification-Oriented Computer Architecture. **IEEE Latin America Transactions**, v. 13, n. 5, p. 1593-1604, 2015.

LINHARES, Robson R.; **Contribuição Para o Desenvolvimento de uma Arquitetura de Computação Própria ao Paradigma Orientado a Notificações**. Tese de Doutorado, Universidade Tecnológica Federal do Paraná - UTFPR, Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial - CPGEI, Curitiba, 2015.

LEONG, Philip. H. W. Recent Trends in FPGA Architectures and Applications. **4th IEEE International Symposium on Electronic Design, Test and Applications.** Hong Kong, China. DOI: 10.1109/DELTA.2008.14. 2008.

MARCONDES, Hugo; FRÖHLICH, Antônio. A. **On Hybrid Hw/Sw Components for Embedded System Design**, In: Proceedings of the 17th IFAC World Congress, pages 9290-9295, Seoul, Korea, July 2008.

MEEUS, Wim; VAN BEECK, Kristof; GOEDEMÉ, Toon; *et al*. An overview of today's high-level synthesis tools. **Design Automation for Embedded Systems**, v. 16, n. 3, p. 31-51, 2012.

MELO, Carlos V.; SIMÃO, Jean M.; FABRO, João. A., Adaptação do Paradigma Orientado a Notificações para o desenvolvimento de sistemas fuzzy. **III CBSF - Terceiro Congresso Brasileiro de Sistemas Fuzzy,** 2014, João Pessoa - PB, Brasil.

MELO, Carlos V.; **Adaptação do Paradigma Orientado a Notificações para Desenvolvimento de Sistemas Fuzzy.** Dissertação de Mestrado. PPGCA/UTFPR, 2016.

MENDONÇA, Igor T.; SIMÃO, Jean M.; WIECHETECK, Luciana. V. B.; STADZISZ, Paulo C. Método para Desenvolvimento de Sistemas Orientados a Regras utilizando o Paradigma Orientado a Notificações. **2nd LA-CCI/CBIC – Latin-American Congress on Computational Intelligence & 12th Congresso Brasileiro de Inteligência Computacional,** October, Curitiba-PR, Brazil, 2015. ISBN: 9788569972006.

MENDONÇA, Igor T. **Metodologia de Projeto de *Software* Orientado a Notificações.** Qualificação de Doutorado. CPGEI/UTFPR, 2015.

MOZAFFARI, Behzad. Optimization of Odd-Even Transposition Network. In: **2010 2nd International Conference on Education Technology and Computer**. 2010.

NANE, Razvan; SIMA, Vlad-Mihai; PILATO, Christian; *et al*. A survey and evaluation of FPGA high-level synthesis tools. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 35, n. 10, p. 1591-1604, 2016.

NEWELL, Allen.; SIMON, Herbert. A. **Human Problem Solving**. Englewood Cliffs, NJ, USA: Prentice-Hall, 1972.

NOVAES, P. J. D. ; SIMÃO, J. M. ; STADZISZ, P. C. . Integration between Requirements Modeling and Software Development in the Notification Oriented Paradigm: A Security System Case Study. In**: Computer on the Beach 2018,** Florianópolis - SC. Anais Computer on the Beach 2018.

OLDS, Dan. Are FPGAs the answer to the Compute Gap, **Research Report: InsideHPC**, **LCC**, 2016.

OLIVEIRA, R. N.; ROTH, V.; HENZEN, A. F.; SIMÃO, J. M.; WILLE, E. C. G.; NOHAMA, P. Notification Oriented Paradigm Applied to Ambient Assisted Living Tool. **IEEE Latin America Transactions**, v. 16, p. 647-653, 2018.

PAES, Carlos E. B.; HIRATA, Celso M. RUP Extension for the Software Performance. **32nd Annual IEEE International Computer Software and Application (COMPSAC `08),** pp 732-738, July 28, 2008.

PEDRONI, Volnei A. **Eletrônica Digital Moderna com VHDL.** Rio de Janeiro, RJ: Elsevier, 2010.

PETERS, Eduardo; JASINSKI, Ricardo P; PEDRONI, Volnei A; *et al*. A new *hardware* coprocessor for accelerating Notification-Oriented applications. In: **Field-Programmable Technology (FPT), 2012 International Conference on.** IEEE, 2012, p. 257-260.

PETERS, Eduardo. **Coprocessador para aceleração de aplicações desenvolvidas utilizando paradigma orientado a notificações.** Dissertação Mestrado, CPGEI/UTFPR, 2012 Orientadores: Prof. V. Pedroni e Prof. J. M. Simão.

PORDEUS Leonardo F. **Simulação de uma arquitetura de computação própria ao paradigma orientado a notificações.** Dissertação de Mestrado. CPGEI/ UTFPR, 2017. Orientadores: Prof. J. M. Simão e Prof. R. R. Linhares.

PORDEUS, L. F.; KERSCHBAUMER, R.; LINHARES, R. R.; WITT, F. A.; STADZISZ, P. C.; LIMA, C. R. E.; SIMÃO, J. M.. Notification Oriented Paradigm to Digital *Hardware*. **Revista SODEBRAS**, v. 11, p. 116-122, 2016

QIAN, Jiang-bo; XU, Hong-bing; DONG, YI-SHENG; *et al*. FPGA acceleration window joins over multiple data streams. **Journal of Circuits, Systems, and Computers,** v. 14, n. 04, p. 813-830, 2005.

ROHMER, Eric; SINGH, Surya P. N.; FREESE, Marc. V-REP: A Versatile and Scalable Robot Simulation *Framework*. **IEEE/RSJ Int. Conf. on Intelligent Robots and Systems**, 2013.

RONSZCKA, Adriano. F.; BELMONTE, Danilo. L.; VALENCA, Glauber. Z.; BATISTA, Márcio. V.; LINHARES, Robson. R.; TACLA, Cesar. A.; STADZISZ, Paulo. C.; SIMAO, Jean. M. Comparações quantitativas e qualitativas entre o Paradigma Orientado a Objetos e o Paradigma Orientado a Notificações sobre um simulador de jogo. Em: III **Congreso Internacional de Computación y Telecomunicaciones - COMTEL**, 2011, Lima, Perú. Memoria: COMTEL 2011. Lima, Perú : Fondo Editorial de la UIGV, 2011. p. 61-64.

RONSZCKA, Adriano. F. **Contribuição para a concepção de aplicações no paradigma orientado a notificações (PON) sob o viés de padrões.** Dissertação de Mestrado, CPGEI/UTFPR. 2012.

RONSZCKA, Adriano. F.; BANASZEWSKI, Roni. F.; LINHARES, Robson. R.; TACLA, Cesar. A.; STADZISZ, Paulo. C.; SIMAO, Jean. M. Notification-Oriented and Rete Network Inference: A Comparative Study. IEEE SMC 2015 - **International Conference On Systems, Man and Cybernetics Conference,** Hong-Kong, China, October, 2015.

RONSZCKA, Adriano. F.; FERREIRA, Cleverson. A.; STADZISZ, Paulo. C.; FABRO, João. A.; SIMAO, Jean. M. Notification-Oriented Programming Language and Compiler. **SBESC 2017** (Artigo Aceito) – 07 a 10 de Novembro de 2017. Curitiba – PR.

RONSZCKA, Adriano. F.; VALENÇA, Glauber. Z.; LINHARES, Robson. R.; STADZISZ, Paulo. C.; SIMAO, Jean. M. Notification-Oriented Paradigm *Framework* 2.0: An Implementation Based On Design Patterns. **IEEE LA - IEEE Latin America Transactions**, Vol. 15, Issue 11, Nov. 2017. ISSN: 1548-0992.

RONSZCKA, Adriano. F. LINGPON - LINGUAGEM DE PROGRAMAÇÃO E COMPILADOR PARA O PARADIGMA ORIENTADO A NOTIFICAÇÕES (PON) – UMA MATERIALIZAÇÃO EFETIVA PARA A VALIDAÇÃO DAS PROPRIEDADES ELEMENTARES DO PON**.** Trabalho de Qualificação de Doutorado, CPGEI, UTFPR. Curitiba, Brasil, 2018.

SANTOS, Leonardo A. **Linguagem e Compilador para o Paradigma Orientado a Notificações: Avanças para Facilitar a Codificação e sua Validação em uma Aplicação de Controle de Futebol de Robôs.** 2017. Dissertação de Mestrado. CPGEI, UTFPR. Curitiba, Brasil, 2017.

SANTOS, Leonardo A.; FABRO, João. A.; SIMAO, Jean. M. Linguagem e Compilador para o Paradigma Orientado a Notificações: avanços para a redução de complexidade de código. **SBESC 2017** (Artigo Aceito) – 07 a 10 de Novembro de 2017. Curitiba – PR 2017.

SCHÜTZ, Fernando; FABRO, João. A.; LIMA, Carlos. R. E.; RONSZCKA, Adriano. F.; STADZISZ, Paulo. C.; SIMAO, Jean. M. Training of an Artificial Neural Network with Backpropagation Algorithm Using Notification Oriented Paradigm**. 2nd LA-CCI/CBIC – Latin-American Congress on Computational Intelligence & 12th Congresso Brasileiro de Inteligência Computacional,** October, Curitiba-PR, Brazil, 2015. http://ieeexplore.ieee.org/document/7435978/

SCHUTZ, F. ; Fabro J. A.; RONSZCKA, A. F.; STADZISZ, P. C.; SIMÃO, J. M. Proposal of a declarative and parallelizable artificial neural network using the notification-oriented paradigm**. Neural Computing & Applications (Internet)**, p. 1-12, 2018.

SCOTT, Michael. L. **Programming Language Pragmatics.** San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 2000.

SIMÃO, Jean. M. **Proposta de uma Arquitetura de Controle para Sistemas Flexíveis de Manufatura Baseada em Regras e Agentes.** Dissertação de Mestrado, Universidade Tecnológica Federal do Paraná - UTFPR, Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial - CPGEI, Curitiba, 2001.

SIMÃO, Jean. M.; STADZISZ, Paulo. C.; MOREL, Gérard. Manufacturing Execution System for Customized Production. **Journal of Material Processing Technology**, 179 (1-3), 268, 2006.

SIMÃO, Jean. M.; STADZISZ, Paulo. C.; BANASZEWSKI, Roni. F.; TACLA, Cesar. A. Holonic Manufacturing Execution Systems for Customised and Agile Production: Manufacturing Plant Simulation. **V Congresso Brasileiro de Engenharia de Fabricação**, 2008.

SIMÃO, Jean. M.; STADZISZ, Paulo. C. An Agent-Oriented Inference Engine applied for Supervisory Control of Automated Manufacturing Systems. **Em: J. Abe, & J. Silva Filho, Advances in Logic, Artificial Intelligence and Robotics** (Vol. 85, p. 234-241). Amsterdam, The Netherlands: IOS Press Books, 2002.

SIMÃO, Jean. M. **A Contribution To The Development Of A HMS Simulation Tool And Proposition Of A Meta-Model For Holonic Control.** Tese de doutorado. CPGEI, CEFET-PR. Curitiba, Brasil, 2005.

SIMÃO, Jean M.; RENAUX, Douglas P. B.; LINHARES, Robson R.; *et al*. Evaluation of the Notification Oriented Paradigm Applied to Sentient Computing. In: 2014 **IEEE 17th**

International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. IEEE, 2014, p. 253-260.

SIMÃO, Jean M.; STADZISZ, Paulo C. Inference based on notifications: a holonic metamodel applied to control issues. **IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans**, v. 39, n. 1, p. 238-250, 2009.

SIMÃO, Jean M.; STADZISZ, Paulo C. **Paradigma Orientado a Notificações (PON) - Uma Técnica de Composição e Execução de *Software* Orientada a Notificações.** 2008, Brasil. Patente: Privilégio de Inovação. Número do registro: PI08055181, data de depósito: 26/11/2008, título: "PEDIDO DE PATENTE: Paradigma Orientado a Notificações (PON) Uma Técnica de Composição e Execução de *Software* Orientada a Notificações.", Instituição de registro: INPI - Instituto Nacional da Propriedade Industrial. Instituição financiadora: Universidade Tecnológica Federal do Paraná.

SIMÃO, Jean. M.; TACLA, Cesar. A.; STADZISZ, Paulo. C.; BANASZEWSKI, Roni. F. Notification Oriented Paradigm (NOP) and Imperative Paradigm: A Comparative Study. **Journal of Software Engineering and Applications**, 5(6), p. 402-416, 2012a.

SIMÃO, Jean. M.; BANASZEWSKI, Roni. F.; TACLA, Cesar. A.; STADZISZ, Paulo. C. R. ***Framework* PON C++ 1.0**, INPI/Brasil, 2017. Pedido de Registro de *Software*. Número do registro provisório: BR 51 2017 001015 3, *Software* Criado em 2008. Data de depósito no INPI: 03/08/2017. Data de depósito na Agência de Inovação (AI) da UTFPR: 11/07/2017.

SIMÃO, Jean. M.; LINHARES, Robson. R.; WITT, Fernando. A.; LIMA, Carlos. R. E.; STADZISZ, Paulo. C. **Paradigma Orientado a Notificações em *Hardware* Digital.** 2012, Brasil. Patente: Privilégio de Inovação. Número do registro: BR102012026429, data de depósito: 16/10/2012, título: "PEDIDO DE PATENTE: Paradigma Orientado a Notificações em *Hardware* Digital", Instituição de registro: INPI - Instituto Nacional da Propriedade Industrial. Instituição financiadora: UTFPR, 2012b.

SIMÃO, Jean. M.; STADZISZ, Paulo. C.; TACLA, Cesar. A.; LINHARES, Robson. R.; BELMONTE, Danilo. L., & BANASZEWSKI, Roni. F. Comparações entre duas materializações do Paradigma Orientado a Notificações (PON): *Framework* PON Prototipal versus *Framework* PON Primário. Em**: IV Congreso Internacional de Computación y Telecomunicaciones - COMTEL** 2012. Lima, Peru: Universidad Inca Garcilaso de la Vega, 2012, p. 13-20, 2012c.

SIMÃO, Jean. M.; STADZISZ, Paulo. C.; WIECHETECK, Luciana. V. B. **Perfil UML para o Paradigma Orientado a Notificações (PON), Perfil UML para o Paradigma Orientado a Regras (POR), Método de Desenvolvimento Orientado a Notificações (DON) e Método de Desenvolvimento Orientado a Regras (DOR).** 2012, Brasil. Patente: Privilégio de Inovação. Número do registro: BR1020120264307, data de depósito: 16/10/2012, título: "PEDIDO DE PATENTE: Perfil UML para o Paradigma Orientado a Notificações (PON), Perfil UML para o Paradigma Orientado a Regras (POR), Método de Desenvolvimento Orientado a Notificações (DON) e Método de Desenvolvimento Orientado a Regras (DOR)", Instituição de registro:INPI - Instituto Nacional da Propriedade Industrial. Instituição financiadora: UTFPR, 2012.

SIMÃO, Jean. M.; PANETTO, Hervé; LIAO, Yongxin; STADZISZ, Paulo. C. A Notification-Oriented Approach for Systems Requirements Engineering. **23rd ISPE - International Conference on Transdisciplinary Engineering,** v. 4. pp. 229-238. Advances in Transdisciplinary Engineering. Amsterdam: IIOS Press, Curitiba – PR, Brazil, 2016.

SIMÃO, Jean. M.; STADZISZ, Paulo. C. ***Framework*** **Referencial PON C++**, INPI/Brasil, 2017. Pedido de Registro de *Software*. Número do registro provisório: BR 51 2017 001014 5, *Software* Criado em 2007. Data de depósito no INPI: 03/08/2017. Data de depósito na Agência de Inovação (AI) da UTFPR: 28/06/2017.

SIMÃO, Jean. M.; RONSZCKA, Adriano. F.; VALENÇA, Glauber. Z.; LINHARES, Robson. R.; STADZISZ, Paulo. C. ***Framework*** **PON C++ 2.0,** INPI/Brasil, 2017 (A SER ENVIADO). Pedido de Registro de *Software*. Número do registro provisório: A SER DEFINIDO, *Software* Criado em 2011. Data de depósito no INPI: A SER FEITO. Data de depósito na Agência de Inovação (AI) da UTFPR: 28/08/2017a.

SOMMERVILLE, Ian. **Software Engineering**, 8th Ed. Addison Wesley, 2004.

TALAU, Marcos. **PONIP: Uso do Paradigma Orientado a Notificações em Redes IP. Aplicação em *Framework* PON 2.0 C++.** Doutorando CPGEI/UTFPR, 2016. Disciplina sobre Paradigma Orientado a Notificações (PON), CPGEI-PPGCA/UTFPR (Prof. J. M. Simão & Prof. H. Panetto [visitante CPGEI & UL-França]), Curitiba - PR, Brasil, 2016.

VALENÇA, Glauber. Z. **Contribuição para Materialização do Paradigma Orientado a Notificações (PON) Via *Framework* e Wizard.** 2012. Dissertação de Mestrado, Programa de Pós-Graduação em Computação Aplicada (PPGCA), UTFPR. Curitiba, Brasil, 2012.

VAN ROY, Peter; HARIDI, Seif. **Concepts, Techniques, and Models of Computer Programming.** MIT Press, 2004.

VAN ROY, Peter. Programming Paradigms for Dummies: What Every Programmer Should Know. In: **New Computational Paradigms for Computer Music.** p. 9-47. G. Assayag and A. Gerzso (eds.), IRCAM/Delatour France, 2009.

XAVIER, R. D.; Fabro J. A. ; STADZISZ, P. C. ; SIMÃO, J. M. . PARADIGMAS DE DESENVOLVIMENTO DE SOFTWARE: COMPARAÇÃO ENTRE ABORDAGENS ORIENTADA A EVENTOS E ORIENTADA A NOTIFICAÇÕES. Revista SODEBRAS, v. 9, p. 104-112, 2014.

WANG, Jimin; PING, Lingdi; WANG, Jiebing. A New Computing Model for C-Like Hardware Description Languages. **IEEE - First International Multi-Symposiums on Computer and Computational Sciences,** IMSCCS '06. 2006.

WATSON, Gregory R.; RASMUSSEN, Craig E.; TIBBITTS, Beth R. An integrated approach to improving the parallel application development process**. IEEE International Symposium on Parallel & Distributed Processing,** 2009. Pp 1 – 8, IPDPS 2009

WATT, David. A. **Programming Language Design Concepts**. J. Willey & Sons, 2004.

WAZLAWICK, R. S**. Metodologia de Pesquisa para Ciência da Computação**. Rio de Janeiro, RJ: Elsevier, 2008.

WEBER, Lucas.; BELMONTE, Danilo. L.; BANASZEWSKI, Roni. F.; STADZISZ, Paulo. C.; SIMÃO, Jean. M. Viabilidade De Controle Orientado A Notificações (CON) Em Ambiente Concorrente Baseado Em Threads. **SICITE**, Brasil, 2010.

WIECHETECK, Luciana. V. B. **Método para Projeto de *Software* usando o Paradigma Orientado a Notificações – PON.** 2011. Dissertação de Mestrado. CPGEI, UTFPR. Curitiba, Brasil, 2011. Disponível em http://files.dirppg.ct.utfpr.edu.br/cpgei/Ano_2011/dissertacoes/CPGEI_Dissertacao_578_2011.pdf.

WIECHETECK, L. V. B. ; STADZISZ, P. C. ; SIMÃO, J. M. . Um Perfil UML para o Paradigma Orientado a Notificações (PON). In**: III Congreso Internacional de Computación y Telecomunicaciones - COMTEL 2011**, Peru - Lima. III Congreso Internacional de Computación y Telecomunicaciones - COMTEL 2011.

WINDH, Skyler; MA, Xiaoyin; HALSTEAD, Robert J; *et al*. High-Level Language Tools for Reconfigurable Computing. **Proceedings of the IEEE**, v. 103, n. 3, p. 390-408, 2015.

WITT, Fernando. A.; SIMAO, Jean. M.; LINHARES, Robson. R.; STADZISZ, Paulo. C.; LIMA, Carlos. R. E. Comparação entre o Paradigma Orientado a Objetos (POO) e o Paradigma Orientado a Notificações (PON) em um Controle Discreto em Lógica Reconfigurável. Em: **XVI SICITE - Seminário de Iniciação Científica e Tecnológica da UTFPR**, 2011, Ponta Grossa - PR. Anais do XVI SICITE, 2011.

APÊNDICES

Nesta seção são apresentados os apêndices, os quais são constituídos por documentos que foram criados pelo autor deste trabalho.

No Apêndice A é apresentado um conjunto de requisitos que serviu como norteador no desenvolvimento do ferramental PON-HD 1.0

Nos Apêndices B, C e D são apresentados os códigos em linguagem VHDL dos componentes do *Frameworks* PON-HD 1.0, respectivamente, *Attribute*, *Method* e *Premise*.

Na sequência, o Apêndice E apresenta o código em linguagem VHDL para o experimento do contador.

Os Apêndices F, G, H e I estão relacionados ao experimento do *driver* para *display* de 7 segmentos e são respectivamente: o código em LingPON-HD 1.0 do *driver* para *display* de 7 segmentos, o código em VHDL gerado a partir do anterior, o código em C++ para este experimento e o código VHDL gerado a parir deste código C++.

Os Apêndices J e K, por sua vez, apresentam respectivamente o código em LingPON-HD 1.0 e em VHDL para o experimento do contador decimal.

Os Apêndices L, M e N estão relacionados ao experimento de cálculo de média e são respectivamente: o código em LingPON-HD 1.0 do cálculo de média, o código em VHDL gerado a partir do anterior e o código VHDL gerado a parir do código C++.

Os códigos escritos em VHDL para o experimento do ordenador de dados são apresentados nos Apêndices O e P. O primeiro elaborado com o auxílio do *Framenwork* PON-HD 1.0 e o segundo elaborado diretamente em VHDL tradicional.

O Apêndice Q apresenta o código em LingPON-HD 1.0 elaborado para o ordenador de dados utilizando o ferramental PON-HD 1.0.

Ainda, o Apêndice R apresenta o código em LingPON-HD 1.0 para o experimento do controlador de robô hexápode.

Os Apêndices S, T, U e V estão relacionados ao experimento do controlador PID e são respectivamente: o código em LingPON-HD 1.0 do controlador PID, o código em VHDL gerado a partir do anterior, o código em C++ para este experimento e o código VHDL gerado a parir deste código C++.

Apêndice A – Diretrizes para o PON-HD 1.0

Com o objetivo de utilizar o PON para a síntese de *hardware* digital foi definido um conjunto de diretrizes que o PON-HD 1.0 deve atender, estas diretrizes são apresentadas a seguir.

Diretrizes
O PON-HD 1.0 deve permitir a execução do PON em *hardware* digital.
O PON-HD 1.0 deve fornecer um ferramental para o desenvolvimento de *hardware* digital em alto nível.
O PON-HD 1.0 deve permitir a demonstração das características do PON em circuitos de lógica reconfigurável.
O PON-HD 1.0 deve permitir a utilização de FPGAs por parte de profissionais que não tenham conhecimento aprofundado de desenvolvimento em *hardware* digital através da programação em regras que possibilitem expressividade em alto nível.
O PON-HD 1.0 deve contemplar o nível de abstração do PON.
O PON-HD 1.0 deve contemplar o desempenho propiciado pelo paralelismo natural de execução do PON.
O PON-HD 1.0 deve contemplar o paralelismo intrínseco de execução do PON.
O PON-HD 1.0 deve permitir verificar a viabilidade da utilização do PON para o desenvolvimento de circuitos digitais em lógica reconfigurável.
Toda a cadeia de notificações do PON deve ser sintetizada no *hardware* através do PON-HD 1.0.
O PON-HD 1.0 deve executar o cálculo lógico por meio de notificações, a luz da teoria do PON.
Os circuitos sintetizados com o PON-HD 1.0 devem respeitar a teoria do PON em suas interfaces de entrada e saída.
O PON-HD 1.0 deve realizar cálculo factual por meio de notificações, utilizando *Attributes* de *FBEs*, a luz da teoria do PON.
O PON-HD 1.0 deve armazenar os dados em *Attributes*, a luz da teoria do PON.
O PON-HD 1.0 deve realizar cálculo lógico utilizando *Premises*, luz da teoria do PON.
O PON-HD 1.0 deve realizar cálculo causal utilizando *Conditions* e *Actions*, a luz da teoria do PON.
O PON-HD 1.0 deve realizar alterações nos dados (mecanismo execucional) através de *Methods*, a luz da teoria do PON.
O PON-HD 1.0 deve realizar as operações de entradas e saídas do circuito através dos *Attributes*, a luz da teoria do PON.

O ferramental PON-HD 1.0 deve permitir a expressão através de regras em alto nível.
O ferramental PON-HD 1.0 deve permitir o desenvolvimento escalável por meio de componentes padronizados e desacoplados a luz da orientação notificações, componentes estes para processamento lógico-causal e para processamento facto-execucional.
O PON-HD 1.0 deve aplicar as características de redução das redundâncias temporais e estruturais do PON no desenvolvimento de *hardware* digital.
O PON-HD 1.0 deve permitir a execução paralela, de forma concorrente, de toda a cadeia de notificações do PON, independentemente do tamanho da aplicação (respeitando, obviamente, as restrições de *hardware*).
O circuito sintetizado com o PON-HD 1.0 deve executar os componentes do PON com o mínimo de acoplamento (a luz do próprio PON) e, portanto, de forma paralela.
O PON-HD 1.0 deve ser baseado em linguagem específica de alto nível, compatível com a linguagem LingPON já existente.
A linguagem LingPON-HD 1.0 deve ser compatível com a linguagem LingPON 1.0, adicionando a esta as funcionalidades necessárias ao desenvolvimento de circuitos em lógica reconfigurável.
O PON-HD 1.0 deve fornecer resultados que possibilitem sua utilização com os componentes de *hardware* reconfigurável existentes no mercado.
O PON-HD 1.0 deve gerar código VHDL compatível com as ferramentas de síntese de *hardware* existentes no mercado.
O PON-HD 1.0 deve permitir a utilização das ferramentas de depuração já existentes, como, "*RTL Viewer*" (INTEL CORPORATION, 2017), "*SignalTap II*" (INTEL CORPORATION, 2017) e "*ModelSim*" (INTEL CORPORATION, 2017), bem como ferramentas similares de outros fabricantes.
Os arquivos gerados pelo compilador PON-HD 1.0 devem estar de acordo com as especificações do VHDL
O PON-HD 1.0 deve permitir a verificação do comportamento do programa (simulação) antes da síntese do circuito.
O PON-HD 1.0 deve ser independente de fabricante ou arquitetura de *hardware* reconfigurável.
Os circuitos desenvolvidos com o PON-HD 1.0 devem comunicar-se com componentes de *hardware* escritos em outras linguagens de descrição de *hardware*.
Os circuitos desenvolvidos com o PON-HD 1.0 devem operar de forma independente, sem a necessidade de elementos externos, com exceção do sinal de *clock*.
O PON-HD 1.0 deve resolver conflitos em seu mecanismo de execução.
Nas aplicações desenvolvidas com o PON-HD 1.0 todo o ciclo de notificações deve acontecer em um único ciclo de *clock*.

Fonte: Autoria própria.

Apêndice B – NOP_attribute.vhd

Código fonte em VHDL para o *Attribute* desenvolvido para o PON-HD 1.0.

```
- ------------- PON ATTRIBUTE ---------------------------
-- version 3.0
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.data_type_pkg.all;

ENTITY NOP_attribute IS
 GENERIC (
 N_bits: INTEGER;
 N_new_values: INTEGER;
 initial_value: STD_LOGIC_VECTOR
 );
 PORT(
 -- inputs
 att_clock :IN STD_LOGIC;
 att_new_value :IN data(0 TO N_new_values-1)(N_bits-1 downto 0);
-- the new values for the attribute
 att_set_value :IN STD_LOGIC_VECTOR(N_new_values-1 downto 0); -
- write the new values to the attribute

 -- outputs
 att_value      :OUT      STD_LOGIC_VECTOR(N_bits-1      downto
0):=initial_value -- the value of the attribute
 );
END NOP_attribute;

ARCHITECTURE NOP_attribute_arq OF NOP_attribute IS
BEGIN
 PROCESS(att_clock)
 BEGIN
  IF (att_clock'EVENT and att_clock = '1') THEN
   FOR i IN 0 to N_new_values-1 LOOP
    IF(att_set_value(i)='1') THEN
     att_value<=att_new_value(i);
     EXIT;
    END IF;
   END LOOP;
  END IF;
 END PROCESS;
END NOP_attribute_arq;
```

Apêndice C – NOP_method.vhd

Código fonte em VHDL para o *Method* desenvolvido para o PON-HD 1.0.

```
-------------------- METHOD -------------------------
-- version 3.0
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--USE ieee.std_logic_signed.all;
USE ieee.numeric_std.all;


ENTITY NOP_method IS
 GENERIC (
 N_bits: INTEGER:=1;
 dataType: INTEGER:=1;
 operation: INTEGER:=1
 );
 PORT(
 -- inputs
 in_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0); -- the value of
input a
 in_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0); -- the value of
input b
 execute :IN STD_LOGIC; -- signals to execute the method

 -- outputs
 result :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0); -- the value
of output
 notify :OUT STD_LOGIC -- notify the attribute
 );
END NOP_method;

ARCHITECTURE NOP_method_arq OF NOP_method IS
SIGNAL temp: STD_LOGIC_VECTOR(N_bits*2-1 downto 0);
BEGIN
 PROCESS(execute,in_a,in_b)
 BEGIN
 -- Attribution
 CASE operation IS
  WHEN 0 => result<=in_a;
  WHEN 1 => result<=std_logic_vector(signed(in_a)+signed(in_b));
  WHEN 2 => result<=std_logic_vector(signed(in_a)-signed(in_b));

  WHEN 3 => temp<=std_logic_vector((signed(in_a)*signed(in_b)));
   result<=temp(N_bits-1 downto 0);
  WHEN 4 => result<=std_logic_vector(signed(in_a)/signed(in_b));

  WHEN 5 => result<=std_logic_vector(ABS(signed(in_a)));
  WHEN    6    =>    result<=std_logic_vector(signed(in_a)    REM
signed(in_b));
```

```
 WHEN   7   =>   result<=std_logic_vector(signed(in_a)   MOD
signed(in_b));
 WHEN 8 => result<= NOT in_a;
 WHEN 9 => result<= in_a AND in_b;
 WHEN 10 => result<= in_a NAND in_b;
 WHEN 11 => result<= in_a OR in_b;
 WHEN 12 => result<= in_a NOR in_b;
 WHEN 13 => result<= in_a XOR in_b;
 WHEN 14=> result<= in_a XNOR in_b;
 WHEN 15 => result<= to_stdlogicvector(to_bitvector(in_a) SLL
to_integer(unsigned(in_b)));
 WHEN 16 => result<= to_stdlogicvector(to_bitvector(in_a) SRL
to_integer(unsigned(in_b)));
 WHEN 17 => result<= to_stdlogicvector(to_bitvector(in_a) SLA
to_integer(unsigned(in_b)));
 WHEN 18 => result<= to_stdlogicvector(to_bitvector(in_a) SRA
to_integer(unsigned(in_b)));
 WHEN 19 => result<= to_stdlogicvector(to_bitvector(in_a) ROL
to_integer(unsigned(in_b)));
 WHEN 20 => result<= to_stdlogicvector(to_bitvector(in_a) ROR
to_integer(unsigned(in_b)));
 WHEN OTHERS => result<=in_a;
 END CASE;
 --------------------------
 notify<=execute;
END PROCESS;
END NOP_method_arq;
```

Apêndice D – NOP_premise.vhd

Código fonte em VHDL para a *Premise* desenvolvido para o PON-HD 1.0.

```
-------------------- PREMISE -------------------------
-- version 3.0
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--USE work.float_pkg.all;
USE ieee.numeric_std.all;

ENTITY NOP_premise IS
 GENERIC (
 N_bits: INTEGER:=1;
 dataType: INTEGER:=0;
 operation: INTEGER:=1
 );
 PORT(
 -- inputs
 att_value_a  :IN  STD_LOGIC_VECTOR(N_bits-1  downto  0);  --  the
value of attribute a to compare
 att_value_b  :IN  STD_LOGIC_VECTOR(N_bits-1  downto  0);  --  the
value of attribute b to compare

 -- outputs
 pre_result :OUT STD_LOGIC -- the result of the premise
 );
END NOP_premise;

ARCHITECTURE NOP_premise_arq OF NOP_premise IS
 SIGNAL triger:STD_LOGIC;
BEGIN
 PROCESS(att_value_a,att_value_b)
 BEGIN
  CASE operation IS
  WHEN 1 => --EQUAL = 1;
   IF (signed(att_value_a) = signed(att_value_b)) THEN
    pre_result<='1';
   ELSE
    pre_result<='0';
   END IF;

  WHEN 2 => --DIFFERENT = 2;
   IF (signed(att_value_a) /= signed(att_value_b)) THEN
    pre_result<='1';
   ELSE
    pre_result<='0';
   END IF;

  WHEN 3 => --LESS_THAN = 3;
   IF (signed(att_value_a) < signed(att_value_b)) THEN
```

```
 pre_result<='1';
 ELSE
 pre_result<='0';
 END IF;

 WHEN 4 => --GREATER_THAN = 4;
 IF (signed(att_value_a) > signed(att_value_b)) THEN
 pre_result<='1';
 ELSE
 pre_result<='0';
 END IF;

 WHEN 5 => --LESS_EQUAL = 5;
 IF (signed(att_value_a) <= signed(att_value_b)) THEN
 pre_result<='1';
 ELSE
 pre_result<='0';
 END IF;

 WHEN 6 => --GREATER_EQUAL = 6;
 IF (signed(att_value_a) >= signed(att_value_b)) THEN
 pre_result<='1';
 ELSE
 pre_result<='0';
 END IF;

 WHEN OTHERS =>
 pre_result<='0';
 END CASE;
END PROCESS;
END NOP_premise_arq;
```

Apêndice E – Contador.vhd

Código fonte em VHDL para o experimento do contador gerado automaticamente pelo Compilador PON-HD 1.0.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.data_type_pkg.all;

ENTITY contador IS
PORT(
 out_counter_atCounter :OUT STD_LOGIC_VECTOR(5-1 downto 0);
 in_counter_atOn :IN STD_LOGIC;
 in_set_counter_atOn :IN STD_LOGIC;
 clock :IN STD_LOGIC
 );
END contador;

ARCHITECTURE contador_arq OF contador IS
---------- COMPONENTS ----------
COMPONENT NOP_attribute
 GENERIC (
  N_bits: INTEGER;
  N_new_values: INTEGER;
  initial_value: STD_LOGIC_VECTOR
  );
 PORT(
  att_clock :IN STD_LOGIC;
  att_new_value :IN data(0 TO N_new_values-1)(N_bits-1 downto 0);
  att_set_value :IN STD_LOGIC_VECTOR(N_new_values-1 downto 0);
  att_value :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0)
  );
END COMPONENT;
COMPONENT NOP_premise
 GENERIC (
  N_bits: INTEGER:=1;
  dataType: INTEGER:=0;
  operation: INTEGER:=1 --EQUAL 1; DIFFERENT 2; LESS_THAN 3; GREATER_THAN
4; LESS_EQUAL 5; GREATER_EQUAL 6;
  );
 PORT(
  att_value_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
  att_value_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
  pre_result :OUT STD_LOGIC
  );
END COMPONENT;
COMPONENT NOP_method
 GENERIC (
   N_bits: INTEGER:=1;
   dataType: INTEGER:=1;
```

```
 operation: INTEGER:=1
 );
 PORT(
 in_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 in_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 execute :IN STD_LOGIC;
 result :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0);
 notify :OUT STD_LOGIC
 );
END COMPONENT;
---------- SIGNALS ----------
 SIGNAL counter_atCounter_new_value: data(0 TO 1-1)(5-1 downto 0);
 SIGNAL counter_atOn_new_value: data(0 TO 1-1)(0 downto 0);

 SIGNAL counter_atCounter_value: STD_LOGIC_VECTOR(5-1 downto 0);
 SIGNAL counter_atOn_value: STD_LOGIC_VECTOR(0 downto 0);

 SIGNAL counter_atCounter_set: STD_LOGIC_VECTOR(1-1 downto 0);
 SIGNAL counter_atOn_set: STD_LOGIC_VECTOR(1-1 downto 0);

 SIGNAL prLess_result: STD_LOGIC;
 SIGNAL prOn_result: STD_LOGIC;

 SIGNAL prLess_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
 SIGNAL prOn_imput_a: STD_LOGIC_VECTOR(0 downto 0);

 SIGNAL prLess_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
 SIGNAL prOn_imput_b: STD_LOGIC_VECTOR(0 downto 0);

 SIGNAL counter_mtIncrement_input_a: STD_LOGIC_VECTOR(5-1 downto 0);

 SIGNAL counter_mtIncrement_input_b: STD_LOGIC_VECTOR(5-1 downto 0);

 SIGNAL counter_mtIncrement_output: STD_LOGIC_VECTOR(5-1 downto 0);

 SIGNAL counter_mtIncrement_execute: STD_LOGIC;

 SIGNAL counter_mtIncrement_notify: STD_LOGIC;
-----------------------------------------------
BEGIN
---------- PON ELEMENTS INSTANCES ----------
 counter_atCounter: NOP_attribute GENERIC MAP(N_bits => 5, N_new_values => 1,
initial_value => "00000") PORT MAP(att_clock => clock, att_new_value =>
counter_atCounter_new_value, att_set_value => counter_atCounter_set, att_value =>
counter_atCounter_value);
 counter_atOn: NOP_attribute GENERIC MAP(N_bits => 1, N_new_values => 1,
initial_value => "0") PORT MAP(att_clock => clock, att_new_value =>
counter_atOn_new_value, att_set_value => counter_atOn_set, att_value =>
counter_atOn_value);
```

```
 prLess_LESS: NOP_premise GENERIC MAP(N_bits => 5, dataType => 0, operation => 3)
PORT MAP(att_value_a => prLess_imput_a, att_value_b => prLess_imput_b, pre_result =>
prLess_result);
 prOn_EQUAL: NOP_premise GENERIC MAP(N_bits => 1, dataType => 0, operation => 1)
PORT MAP(att_value_a => prOn_imput_a, att_value_b => prOn_imput_b, pre_result =>
prOn_result);

 counter_mtIncrement: NOP_method GENERIC MAP(N_bits => 5, dataType => 1, operation
=> 1) PORT MAP(in_a => counter_mtIncrement_input_a, in_b =>
counter_mtIncrement_input_b, execute => counter_mtIncrement_execute, result =>
counter_mtIncrement_output, notify => counter_mtIncrement_notify);
---------- PON ELEMENTS CONNECTIONS ----------
 ---------- attributes inputs ----------
 counter_atCounter_new_value(0) <= counter_mtIncrement_output;
 counter_atCounter_set(0) <= counter_mtIncrement_notify;
 counter_atOn_new_value(0)(0) <= in_counter_atOn;
 counter_atOn_set(0) <= in_set_counter_atOn;

 ---------- attributes outputs ----------
 out_counter_atCounter <= counter_atCounter_value;

 ---------- premises inputs ----------
 prLess_imput_a <= counter_atCounter_value;
 prOn_imput_a <= counter_atOn_value;

 prLess_imput_b <= "01010";
 prOn_imput_b <= "1";

 ---------- Methods Imputs ----------
 counter_mtIncrement_execute <= (prOn_result AND prLess_result);
 counter_mtIncrement_input_a <= counter_atCounter_value;
 counter_mtIncrement_input_b <= "00001";
END contador_arq;
```

Apêndice F – driverDisplay7Segmentos.pon

Código fonte em LingPON-HD 1.0 para o experimento do driver de *display* de 7 segmentos.

```
fbe fbeDriverDisplay
 attributes
  integer entrada 64
  integer saida 0
  boolean reset false
 end_attributes
 methods
  method mtVal0(saida = 64)
  method mtVal1(saida = 121)
  method mtVal2(saida = 36)
  method mtVal3(saida = 48)
  method mtVal4(saida = 25)
  method mtVal5(saida = 18)
  method mtVal6(saida = 2)
  method mtVal7(saida = 120)
  method mtVal8(saida = 0)
  method mtVal9(saida = 16)
 end_methods
end_fbe

inst
 fbeDriverDisplay DD
end_inst

strategy
 no_one
end_strategy

rule rlReset
 condition
  subcondition SubReset
   premise prReset DD.reset == true
  end_subcondition
 end_condition
 action
  instigation inReset DD.mtVal0();
 end_action
end_rule

rule rlV0
 condition
  subcondition SubV0
   premise prV0 DD.entrada == 0 and
   premise prNotReset DD.reset == false
  end_subcondition
 end_condition
```

```
 action
  instigation inV0 DD.mtVal0();
 end_action
end_rule

rule rlV1
 condition
  subcondition SubV1
   premise prV1 DD.entrada == 1 and
   premise prNotReset DD.reset == false
  end_subcondition
 end_condition
 action
  instigation inV1 DD.mtVal1();
 end_action
end_rule

rule rlV2
 condition
  subcondition SubV2
   premise prV2 DD.entrada == 2 and
   premise prNotReset DD.reset == false
  end_subcondition
 end_condition
 action
  instigation inV2 DD.mtVal2();
 end_action
end_rule

rule rlV3
 condition
  subcondition SubV3
   premise prV3 DD.entrada == 3 and
   premise prNotReset DD.reset == false
  end_subcondition
 end_condition
 action
  instigation inV3 DD.mtVal3();
 end_action
end_rule

rule rlV4
 condition
  subcondition SubV4
   premise prV4 DD.entrada == 4 and
   premise prNotReset DD.reset == false
  end_subcondition
 end_condition
 action
  instigation inV4 DD.mtVal4();
 end_action
```

```
end_rule

rule rlV5
 condition
  subcondition SubV5
   premise prV5 DD.entrada == 5 and
   premise prNotReset DD.reset == false
  end_subcondition
 end_condition
 action
  instigation inV5 DD.mtVal5();
 end_action
end_rule

rule rlV6
 condition
  subcondition SubV6
   premise prV6 DD.entrada == 6 and
   premise prNotReset DD.reset == false
  end_subcondition
 end_condition
 action
  instigation inV6 DD.mtVal6();
 end_action
end_rule

rule rlV7
 condition
  subcondition SubV7
   premise prV7 DD.entrada == 7 and
   premise prNotReset DD.reset == false
  end_subcondition
 end_condition
 action
  instigation inV7 DD.mtVal7();
 end_action
end_rule

rule rlV8
 condition
  subcondition SubV8
   premise prV8 DD.entrada == 8 and
   premise prNotReset DD.reset == false
  end_subcondition
 end_condition
 action
  instigation inV8 DD.mtVal8();
 end_action
end_rule

rule rlV9
```

```
 condition
  subcondition SubV9
   premise prV9 DD.entrada == 9 and
   premise prNotReset DD.reset == false
  end_subcondition
 end_condition
 action
  instigation inV9 DD.mtVal9();
 end_action
end_rule

main {
 entity driverDisplay7Segmentos

 in DD.reset
 in DD.entrada
 out DD.saida
 NBits DD.entrada 4
 NBits DD.saida 7
 }
```

Apêndice G – driverDisplay7Segmentos.vhd

Código fonte em VHDL para o experimento do driver de *display* de 7 segmentos gerado automaticamente pelo Compilador PON-HD 1.0.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.data_type_pkg.all;

ENTITY driverDisplay7Segmentos IS
PORT(
 in_DD_reset :IN STD_LOGIC;
 in_set_DD_reset :IN STD_LOGIC;
 in_DD_entrada :IN STD_LOGIC_VECTOR(4-1 downto 0);
 in_set_DD_entrada :IN STD_LOGIC;
 out_DD_saida :OUT STD_LOGIC_VECTOR(7-1 downto 0);
 clock :IN STD_LOGIC
 );
END driverDisplay7Segmentos;

ARCHITECTURE driverDisplay7Segmentos_arq OF
driverDisplay7Segmentos IS
----------------- COMPONENTS ---------------------
COMPONENT NOP_attribute
 GENERIC (
 N_bits: INTEGER;
 N_new_values: INTEGER;
 initial_value: STD_LOGIC_VECTOR
 );
 PORT(
 att_clock :IN STD_LOGIC;
 att_new_value :IN data(0 TO N_new_values-1)(N_bits-1 downto
0);
 att_set_value :IN STD_LOGIC_VECTOR(N_new_values-1 downto 0);
 att_value :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0)
 );
END COMPONENT;
COMPONENT NOP_premise
 GENERIC (
 N_bits: INTEGER:=1;
 dataType: INTEGER:=0;
 operation: INTEGER:=1 --EQUAL 1; DIFFERENT 2; LESS_THAN 3;
GREATER_THAN 4; LESS_EQUAL 5; GREATER_EQUAL 6;
 );
 PORT(
 att_value_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 att_value_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 pre_result :OUT STD_LOGIC
 );
END COMPONENT;
COMPONENT NOP_method
```

```
GENERIC (
 N_bits: INTEGER:=1;
 dataType: INTEGER:=1;
operation: INTEGER:=1
);
PORT(
in_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
in_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
execute :IN STD_LOGIC;
result :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0);
notify :OUT STD_LOGIC
);
END COMPONENT;
----------------- SIGNALS --------------------
SIGNAL DD_entrada_new_value: data(0 TO 1-1)(4-1 downto 0);
SIGNAL DD_saida_new_value: data(0 TO 10-1)(7-1 downto 0);
SIGNAL DD_reset_new_value: data(0 TO 1-1)(0 downto 0);

SIGNAL DD_entrada_value: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL DD_saida_value: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_reset_value: STD_LOGIC_VECTOR(0 downto 0);

SIGNAL DD_entrada_set: STD_LOGIC_VECTOR(1-1 downto 0);
SIGNAL DD_saida_set: STD_LOGIC_VECTOR(10-1 downto 0);
SIGNAL DD_reset_set: STD_LOGIC_VECTOR(1-1 downto 0);

SIGNAL prNotReset_result: STD_LOGIC;
SIGNAL prReset_result: STD_LOGIC;
SIGNAL prV0_result: STD_LOGIC;
SIGNAL prV1_result: STD_LOGIC;
SIGNAL prV2_result: STD_LOGIC;
SIGNAL prV3_result: STD_LOGIC;
SIGNAL prV4_result: STD_LOGIC;
SIGNAL prV5_result: STD_LOGIC;
SIGNAL prV6_result: STD_LOGIC;
SIGNAL prV7_result: STD_LOGIC;
SIGNAL prV8_result: STD_LOGIC;
SIGNAL prV9_result: STD_LOGIC;

SIGNAL prNotReset_imput_a: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL prReset_imput_a: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL prV0_imput_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV1_imput_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV2_imput_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV3_imput_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV4_imput_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV5_imput_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV6_imput_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV7_imput_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV8_imput_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV9_imput_a: STD_LOGIC_VECTOR(4-1 downto 0);
```

```
SIGNAL prNotReset_imput_b: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL prReset_imput_b: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL prV0_imput_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV1_imput_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV2_imput_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV3_imput_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV4_imput_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV5_imput_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV6_imput_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV7_imput_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV8_imput_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL prV9_imput_b: STD_LOGIC_VECTOR(4-1 downto 0);

SIGNAL DD_mtVal0_input_a: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal1_input_a: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal2_input_a: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal3_input_a: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal4_input_a: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal5_input_a: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal6_input_a: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal7_input_a: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal8_input_a: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal9_input_a: STD_LOGIC_VECTOR(7-1 downto 0);

SIGNAL DD_mtVal0_input_b: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal1_input_b: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal2_input_b: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal3_input_b: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal4_input_b: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal5_input_b: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal6_input_b: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal7_input_b: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal8_input_b: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal9_input_b: STD_LOGIC_VECTOR(7-1 downto 0);

SIGNAL DD_mtVal0_output: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal1_output: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal2_output: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal3_output: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal4_output: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal5_output: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal6_output: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal7_output: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal8_output: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL DD_mtVal9_output: STD_LOGIC_VECTOR(7-1 downto 0);

SIGNAL DD_mtVal0_execute: STD_LOGIC;
SIGNAL DD_mtVal1_execute: STD_LOGIC;
SIGNAL DD_mtVal2_execute: STD_LOGIC;
SIGNAL DD_mtVal3_execute: STD_LOGIC;
```

```
SIGNAL DD_mtVal4_execute: STD_LOGIC;
SIGNAL DD_mtVal5_execute: STD_LOGIC;
SIGNAL DD_mtVal6_execute: STD_LOGIC;
SIGNAL DD_mtVal7_execute: STD_LOGIC;
SIGNAL DD_mtVal8_execute: STD_LOGIC;
SIGNAL DD_mtVal9_execute: STD_LOGIC;

SIGNAL DD_mtVal0_notify: STD_LOGIC;
SIGNAL DD_mtVal1_notify: STD_LOGIC;
SIGNAL DD_mtVal2_notify: STD_LOGIC;
SIGNAL DD_mtVal3_notify: STD_LOGIC;
SIGNAL DD_mtVal4_notify: STD_LOGIC;
SIGNAL DD_mtVal5_notify: STD_LOGIC;
SIGNAL DD_mtVal6_notify: STD_LOGIC;
SIGNAL DD_mtVal7_notify: STD_LOGIC;
SIGNAL DD_mtVal8_notify: STD_LOGIC;
SIGNAL DD_mtVal9_notify: STD_LOGIC;
-----------------------------------------------
BEGIN
---------- PON ELEMENTS INSTANCES -------------
 DD_entrada: NOP_attribute GENERIC MAP(N_bits => 4,
N_new_values => 1, initial_value => "0000") PORT MAP(att_clock
=> clock, att_new_value => DD_entrada_new_value, att_set_value
=> DD_entrada_set, att_value => DD_entrada_value);
 DD_saida: NOP_attribute GENERIC MAP(N_bits => 7, N_new_values
=> 10, initial_value => "0000000") PORT MAP(att_clock =>
clock, att_new_value => DD_saida_new_value, att_set_value =>
DD_saida_set, att_value => DD_saida_value);
 DD_reset: NOP_attribute GENERIC MAP(N_bits => 1, N_new_values
=> 1, initial_value => "0") PORT MAP(att_clock => clock,
att_new_value => DD_reset_new_value, att_set_value =>
DD_reset_set, att_value => DD_reset_value);

 prNotReset_EQUAL: NOP_premise GENERIC MAP(N_bits => 1,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prNotReset_imput_a, att_value_b => prNotReset_imput_b,
pre_result => prNotReset_result);
 prReset_EQUAL: NOP_premise GENERIC MAP(N_bits => 1, dataType
=> 0, operation => 1) PORT MAP(att_value_a => prReset_imput_a,
att_value_b => prReset_imput_b, pre_result => prReset_result);
 prV0_EQUAL: NOP_premise GENERIC MAP(N_bits => 4, dataType =>
0, operation => 1) PORT MAP(att_value_a => prV0_imput_a,
att_value_b => prV0_imput_b, pre_result => prV0_result);
 prV1_EQUAL: NOP_premise GENERIC MAP(N_bits => 4, dataType =>
0, operation => 1) PORT MAP(att_value_a => prV1_imput_a,
att_value_b => prV1_imput_b, pre_result => prV1_result);
 prV2_EQUAL: NOP_premise GENERIC MAP(N_bits => 4, dataType =>
0, operation => 1) PORT MAP(att_value_a => prV2_imput_a,
att_value_b => prV2_imput_b, pre_result => prV2_result);
```

```
prV3_EQUAL: NOP_premise GENERIC MAP(N_bits => 4, dataType =>
0, operation => 1) PORT MAP(att_value_a => prV3_imput_a,
att_value_b => prV3_imput_b, pre_result => prV3_result);
prV4_EQUAL: NOP_premise GENERIC MAP(N_bits => 4, dataType =>
0, operation => 1) PORT MAP(att_value_a => prV4_imput_a,
att_value_b => prV4_imput_b, pre_result => prV4_result);
prV5_EQUAL: NOP_premise GENERIC MAP(N_bits => 4, dataType =>
0, operation => 1) PORT MAP(att_value_a => prV5_imput_a,
att_value_b => prV5_imput_b, pre_result => prV5_result);
prV6_EQUAL: NOP_premise GENERIC MAP(N_bits => 4, dataType =>
0, operation => 1) PORT MAP(att_value_a => prV6_imput_a,
att_value_b => prV6_imput_b, pre_result => prV6_result);
prV7_EQUAL: NOP_premise GENERIC MAP(N_bits => 4, dataType =>
0, operation => 1) PORT MAP(att_value_a => prV7_imput_a,
att_value_b => prV7_imput_b, pre_result => prV7_result);
prV8_EQUAL: NOP_premise GENERIC MAP(N_bits => 4, dataType =>
0, operation => 1) PORT MAP(att_value_a => prV8_imput_a,
att_value_b => prV8_imput_b, pre_result => prV8_result);
prV9_EQUAL: NOP_premise GENERIC MAP(N_bits => 4, dataType =>
0, operation => 1) PORT MAP(att_value_a => prV9_imput_a,
att_value_b => prV9_imput_b, pre_result => prV9_result);

DD_mtVal0: NOP_method GENERIC MAP(N_bits => 7, dataType => 1,
operation => 0) PORT MAP(in_a => DD_mtVal0_input_a, in_b =>
DD_mtVal0_input_b, execute => DD_mtVal0_execute, result =>
DD_mtVal0_output, notify => DD_mtVal0_notify);
DD_mtVal1: NOP_method GENERIC MAP(N_bits => 7, dataType => 1,
operation => 0) PORT MAP(in_a => DD_mtVal1_input_a, in_b =>
DD_mtVal1_input_b, execute => DD_mtVal1_execute, result =>
DD_mtVal1_output, notify => DD_mtVal1_notify);
DD_mtVal2: NOP_method GENERIC MAP(N_bits => 7, dataType => 1,
operation => 0) PORT MAP(in_a => DD_mtVal2_input_a, in_b =>
DD_mtVal2_input_b, execute => DD_mtVal2_execute, result =>
DD_mtVal2_output, notify => DD_mtVal2_notify);
DD_mtVal3: NOP_method GENERIC MAP(N_bits => 7, dataType => 1,
operation => 0) PORT MAP(in_a => DD_mtVal3_input_a, in_b =>
DD_mtVal3_input_b, execute => DD_mtVal3_execute, result =>
DD_mtVal3_output, notify => DD_mtVal3_notify);
DD_mtVal4: NOP_method GENERIC MAP(N_bits => 7, dataType => 1,
operation => 0) PORT MAP(in_a => DD_mtVal4_input_a, in_b =>
DD_mtVal4_input_b, execute => DD_mtVal4_execute, result =>
DD_mtVal4_output, notify => DD_mtVal4_notify);
DD_mtVal5: NOP_method GENERIC MAP(N_bits => 7, dataType => 1,
operation => 0) PORT MAP(in_a => DD_mtVal5_input_a, in_b =>
DD_mtVal5_input_b, execute => DD_mtVal5_execute, result =>
DD_mtVal5_output, notify => DD_mtVal5_notify);
DD_mtVal6: NOP_method GENERIC MAP(N_bits => 7, dataType => 1,
operation => 0) PORT MAP(in_a => DD_mtVal6_input_a, in_b =>
DD_mtVal6_input_b, execute => DD_mtVal6_execute, result =>
DD_mtVal6_output, notify => DD_mtVal6_notify);
```

```
DD_mtVal7: NOP_method GENERIC MAP(N_bits => 7, dataType => 1,
operation => 0) PORT MAP(in_a => DD_mtVal7_input_a, in_b =>
DD_mtVal7_input_b, execute => DD_mtVal7_execute, result =>
DD_mtVal7_output, notify => DD_mtVal7_notify);
DD_mtVal8: NOP_method GENERIC MAP(N_bits => 7, dataType => 1,
operation => 0) PORT MAP(in_a => DD_mtVal8_input_a, in_b =>
DD_mtVal8_input_b, execute => DD_mtVal8_execute, result =>
DD_mtVal8_output, notify => DD_mtVal8_notify);
DD_mtVal9: NOP_method GENERIC MAP(N_bits => 7, dataType => 1,
operation => 0) PORT MAP(in_a => DD_mtVal9_input_a, in_b =>
DD_mtVal9_input_b, execute => DD_mtVal9_execute, result =>
DD_mtVal9_output, notify => DD_mtVal9_notify);
---------- PON ELEMENTS CONNECTIONS -------------
---------- attributes inputs -------------
DD_entrada_new_value(0) <= in_DD_entrada;
DD_entrada_set(0) <= in_set_DD_entrada;
DD_saida_new_value(0) <= DD_mtVal0_output;
DD_saida_set(0) <= DD_mtVal0_notify;
DD_saida_new_value(1) <= DD_mtVal1_output;
DD_saida_set(1) <= DD_mtVal1_notify;
DD_saida_new_value(2) <= DD_mtVal2_output;
DD_saida_set(2) <= DD_mtVal2_notify;
DD_saida_new_value(3) <= DD_mtVal3_output;
DD_saida_set(3) <= DD_mtVal3_notify;
DD_saida_new_value(4) <= DD_mtVal4_output;
DD_saida_set(4) <= DD_mtVal4_notify;
DD_saida_new_value(5) <= DD_mtVal5_output;
DD_saida_set(5) <= DD_mtVal5_notify;
DD_saida_new_value(6) <= DD_mtVal6_output;
DD_saida_set(6) <= DD_mtVal6_notify;
DD_saida_new_value(7) <= DD_mtVal7_output;
DD_saida_set(7) <= DD_mtVal7_notify;
DD_saida_new_value(8) <= DD_mtVal8_output;
DD_saida_set(8) <= DD_mtVal8_notify;
DD_saida_new_value(9) <= DD_mtVal9_output;
DD_saida_set(9) <= DD_mtVal9_notify;
DD_reset_new_value(0)(0) <= in_DD_reset;
DD_reset_set(0) <= in_set_DD_reset;

---------- attributes outputs -------------
out_DD_saida <= DD_saida_value;

---------- premises inputs -------------
prNotReset_imput_a <= DD_reset_value;
prReset_imput_a <= DD_reset_value;
prV0_imput_a <= DD_entrada_value;
prV1_imput_a <= DD_entrada_value;
prV2_imput_a <= DD_entrada_value;
prV3_imput_a <= DD_entrada_value;
prV4_imput_a <= DD_entrada_value;
prV5_imput_a <= DD_entrada_value;
```

```
prV6_imput_a <= DD_entrada_value;
prV7_imput_a <= DD_entrada_value;
prV8_imput_a <= DD_entrada_value;
prV9_imput_a <= DD_entrada_value;

prNotReset_imput_b <= "0";
prReset_imput_b <= "1";
prV0_imput_b <= "0000";
prV1_imput_b <= "0001";
prV2_imput_b <= "0010";
prV3_imput_b <= "0011";
prV4_imput_b <= "0100";
prV5_imput_b <= "0101";
prV6_imput_b <= "0110";
prV7_imput_b <= "0111";
prV8_imput_b <= "1000";
prV9_imput_b <= "1001";

---------- Methods Imputs -------------
DD_mtVal0_execute <= (prReset_result) OR (prNotReset_result
AND prV0_result);
DD_mtVal1_execute <= (prNotReset_result AND prV1_result);
DD_mtVal2_execute <= (prNotReset_result AND prV2_result);
DD_mtVal3_execute <= (prNotReset_result AND prV3_result);
DD_mtVal4_execute <= (prNotReset_result AND prV4_result);
DD_mtVal5_execute <= (prNotReset_result AND prV5_result);
DD_mtVal6_execute <= (prNotReset_result AND prV6_result);
DD_mtVal7_execute <= (prNotReset_result AND prV7_result);
DD_mtVal8_execute <= (prNotReset_result AND prV8_result);
DD_mtVal9_execute <= (prNotReset_result AND prV9_result);
DD_mtVal0_input_a <= "1000000";
DD_mtVal0_input_b <= "0000000";
DD_mtVal1_input_a <= "1111001";
DD_mtVal1_input_b <= "0000000";
DD_mtVal2_input_a <= "0100100";
DD_mtVal2_input_b <= "0000000";
DD_mtVal3_input_a <= "0110000";
DD_mtVal3_input_b <= "0000000";
DD_mtVal4_input_a <= "0011001";
DD_mtVal4_input_b <= "0000000";
DD_mtVal5_input_a <= "0010010";
DD_mtVal5_input_b <= "0000000";
DD_mtVal6_input_a <= "0000010";
DD_mtVal6_input_b <= "0000000";
DD_mtVal7_input_a <= "1111000";
DD_mtVal7_input_b <= "0000000";
DD_mtVal8_input_a <= "0000000";
DD_mtVal8_input_b <= "0000000";
DD_mtVal9_input_a <= "0010000";
DD_mtVal9_input_b <= "0000000";
END driverDisplay7Segmentos_arq;
```

Apêndice H – driverDisplay7segmentos.cpp

Código fonte em C++ do experimento do driver de *display* de 7 segmentos utilizado com a ferramenta Vivado HLS.

```
#include "driverDisplay7segmentos.h"

void driverDisplay7segmentos(bool reset, uint4 entrada, uint7
*saida)
{
 if(reset==true)
 {
  *saida=64;
 }
 else
 {
  if(entrada==0)
  {
   *saida=64;
  }

  if(entrada==1)
  {
   *saida=121;
  }

  if(entrada==2)
  {
   *saida=36;
  }

  if(entrada==3)
  {
   *saida=48;
  }

  if(entrada==4)
  {
   *saida=25;
  }

  if(entrada==5)
  {
   *saida=18;
  }

  if(entrada==6)
  {
   *saida=2;
  }
```

```
 if(entrada==7)
 {
  *saida=120;
 }

 if(entrada==8)
 {
  *saida=0;
 }

 if(entrada==9)
 {
  *saida=16;
 }
}
}
```

Apêndice I – driverDisplay7segmentos.vhd

Código fonte em VHDL para o experimento do driver de *display* de 7 segmentos gerado automaticamente pela ferramenta Vivado HLS.

```
--
==========================================================
-- RTL generated by Vivado(TM) HLS - High-Level Synthesis from
C, C++ and SystemC
-- Version: 2018.1_AR70908
-- Copyright (C) 1986-2018 Xilinx, Inc. All Rights Reserved.
--
-- ==========================================================

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity driverDisplay7segmentos is
port (
    ap_start : IN STD_LOGIC;
    ap_done : OUT STD_LOGIC;
    ap_idle : OUT STD_LOGIC;
    ap_ready : OUT STD_LOGIC;
    reset : IN STD_LOGIC;
    entrada : IN STD_LOGIC_VECTOR (3 downto 0);
    saida : OUT STD_LOGIC_VECTOR (6 downto 0);
    saida_ap_vld : OUT STD_LOGIC );
end;


architecture behav of driverDisplay7segmentos is
    attribute CORE_GENERATION_INFO : STRING;
    attribute CORE_GENERATION_INFO of behav : architecture is

"driverDisplay7segmentos,hls_ip_2018_1_AR70908,{HLS_INPUT_TYPE
=cxx,HLS_INPUT_FLOAT=0,HLS_INPUT_FIXED=1,HLS_INPUT_PART=xa7a12
tcsg325-
1q,HLS_INPUT_CLOCK=20.000000,HLS_INPUT_ARCH=others,HLS_SYN_CLO
CK=6.920000,HLS_SYN_LAT=0,HLS_SYN_TPT=none,HLS_SYN_MEM=0,HLS_S
YN_DSP=0,HLS_SYN_FF=0,HLS_SYN_LUT=197}";
    constant ap_const_logic_1 : STD_LOGIC := '1';
    constant ap_const_boolean_1 : BOOLEAN := true;
    constant ap_const_logic_0 : STD_LOGIC := '0';
    constant ap_const_lv1_1 : STD_LOGIC_VECTOR (0 downto 0) :=
"1";
    constant ap_const_lv4_1 : STD_LOGIC_VECTOR (3 downto 0) :=
"0001";
    constant ap_const_lv7_79 : STD_LOGIC_VECTOR (6 downto 0)
:= "1111001";
```

```
    constant ap_const_lv7_40 : STD_LOGIC_VECTOR (6 downto 0)
:= "1000000";
    constant ap_const_lv4_2 : STD_LOGIC_VECTOR (3 downto 0) :=
"0010";
    constant ap_const_lv4_3 : STD_LOGIC_VECTOR (3 downto 0) :=
"0011";
    constant ap_const_lv7_30 : STD_LOGIC_VECTOR (6 downto 0)
:= "0110000";
    constant ap_const_lv7_24 : STD_LOGIC_VECTOR (6 downto 0)
:= "0100100";
    constant ap_const_lv4_4 : STD_LOGIC_VECTOR (3 downto 0) :=
"0100";
    constant ap_const_lv4_5 : STD_LOGIC_VECTOR (3 downto 0) :=
"0101";
    constant ap_const_lv7_12 : STD_LOGIC_VECTOR (6 downto 0)
:= "0010010";
    constant ap_const_lv7_19 : STD_LOGIC_VECTOR (6 downto 0)
:= "0011001";
    constant ap_const_lv4_6 : STD_LOGIC_VECTOR (3 downto 0) :=
"0110";
    constant ap_const_lv4_7 : STD_LOGIC_VECTOR (3 downto 0) :=
"0111";
    constant ap_const_lv7_78 : STD_LOGIC_VECTOR (6 downto 0)
:= "1111000";
    constant ap_const_lv7_2 : STD_LOGIC_VECTOR (6 downto 0) :=
"0000010";
    constant ap_const_lv4_8 : STD_LOGIC_VECTOR (3 downto 0) :=
"1000";
    constant ap_const_lv4_9 : STD_LOGIC_VECTOR (3 downto 0) :=
"1001";
    constant ap_const_lv7_10 : STD_LOGIC_VECTOR (6 downto 0)
:= "0010000";
    constant ap_const_lv7_0 : STD_LOGIC_VECTOR (6 downto 0) :=
"0000000";
    constant ap_const_lv4_A : STD_LOGIC_VECTOR (3 downto 0) :=
"1010";
    constant ap_const_lv1_0 : STD_LOGIC_VECTOR (0 downto 0) :=
"0";

    signal reset_read_read_fu_64_p2 : STD_LOGIC_VECTOR (0
downto 0);
    signal or_cond_fu_242_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_1_fu_77_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_3_fu_97_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_2_fu_91_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_fu_111_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal p_saida_new_1_cast_ca_fu_103_p3 : STD_LOGIC_VECTOR
(6 downto 0);
    signal saida_new_1_fu_83_p3 : STD_LOGIC_VECTOR (6 downto
0);
    signal tmp_5_fu_131_p2 : STD_LOGIC_VECTOR (0 downto 0);
```

```
    signal tmp_4_fu_125_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_s_fu_145_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal p_saida_new_3_cast_ca_fu_137_p3 : STD_LOGIC_VECTOR
(6 downto 0);
    signal saida_new_3_fu_117_p3 : STD_LOGIC_VECTOR (6 downto
0);
    signal tmp_7_fu_165_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_6_fu_159_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_10_fu_179_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal p_saida_new_5_cast_ca_fu_171_p3 : STD_LOGIC_VECTOR
(6 downto 0);
    signal saida_new_5_fu_151_p3 : STD_LOGIC_VECTOR (6 downto
0);
    signal tmp_9_fu_199_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_8_fu_193_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_11_fu_213_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal p_saida_new_7_cast_ca_fu_205_p3 : STD_LOGIC_VECTOR
(6 downto 0);
    signal saida_new_7_fu_185_p3 : STD_LOGIC_VECTOR (6 downto
0);
    signal saida_new_s_fu_227_p0 : STD_LOGIC_VECTOR (0 downto
0);
    signal saida_new_9_fu_219_p3 : STD_LOGIC_VECTOR (6 downto
0);
    signal switch_fu_236_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal or_cond_fu_242_p1 : STD_LOGIC_VECTOR (0 downto 0);


begin



    ap_done <= ap_start;
    ap_idle <= ap_const_logic_1;
    ap_ready <= ap_start;
    or_cond_fu_242_p1 <= (0=>reset, others=>'-');
    or_cond_fu_242_p2 <= (switch_fu_236_p2 or
or_cond_fu_242_p1);
    p_saida_new_1_cast_ca_fu_103_p3 <=
        ap_const_lv7_30 when (tmp_3_fu_97_p2(0) = '1') else
        ap_const_lv7_24;
    p_saida_new_3_cast_ca_fu_137_p3 <=
        ap_const_lv7_12 when (tmp_5_fu_131_p2(0) = '1') else
        ap_const_lv7_19;
    p_saida_new_5_cast_ca_fu_171_p3 <=
        ap_const_lv7_78 when (tmp_7_fu_165_p2(0) = '1') else
        ap_const_lv7_2;
    p_saida_new_7_cast_ca_fu_205_p3 <=
        ap_const_lv7_10 when (tmp_9_fu_199_p2(0) = '1') else
        ap_const_lv7_0;
    reset_read_read_fu_64_p2 <= (0=>reset, others=>'-');
```

```
    saida <=
        ap_const_lv7_40 when (saida_new_s_fu_227_p0(0) = '1')
else
        saida_new_9_fu_219_p3;

    saida_ap_vld_assign_proc : process(ap_start,
or_cond_fu_242_p2)
    begin
        if (((ap_start = ap_const_logic_1) and
(or_cond_fu_242_p2 = ap_const_lv1_1))) then
            saida_ap_vld <= ap_const_logic_1;
        else
            saida_ap_vld <= ap_const_logic_0;
        end if;
    end process;

    saida_new_1_fu_83_p3 <=
        ap_const_lv7_79 when (tmp_1_fu_77_p2(0) = '1') else
        ap_const_lv7_40;
    saida_new_3_fu_117_p3 <=
        p_saida_new_1_cast_ca_fu_103_p3 when (tmp_fu_111_p2(0)
= '1') else
        saida_new_1_fu_83_p3;
    saida_new_5_fu_151_p3 <=
        p_saida_new_3_cast_ca_fu_137_p3 when
(tmp_s_fu_145_p2(0) = '1') else
        saida_new_3_fu_117_p3;
    saida_new_7_fu_185_p3 <=
        p_saida_new_5_cast_ca_fu_171_p3 when
(tmp_10_fu_179_p2(0) = '1') else
        saida_new_5_fu_151_p3;
    saida_new_9_fu_219_p3 <=
        p_saida_new_7_cast_ca_fu_205_p3 when
(tmp_11_fu_213_p2(0) = '1') else
        saida_new_7_fu_185_p3;
    saida_new_s_fu_227_p0 <= (0=>reset, others=>'-');
    switch_fu_236_p2 <= "1" when (unsigned(entrada) <
unsigned(ap_const_lv4_A)) else "0";
    tmp_10_fu_179_p2 <= (tmp_7_fu_165_p2 or tmp_6_fu_159_p2);
    tmp_11_fu_213_p2 <= (tmp_9_fu_199_p2 or tmp_8_fu_193_p2);
    tmp_1_fu_77_p2 <= "1" when (entrada = ap_const_lv4_1) else
"0";
    tmp_2_fu_91_p2 <= "1" when (entrada = ap_const_lv4_2) else
"0";
    tmp_3_fu_97_p2 <= "1" when (entrada = ap_const_lv4_3) else
"0";
    tmp_4_fu_125_p2 <= "1" when (entrada = ap_const_lv4_4)
else "0";
    tmp_5_fu_131_p2 <= "1" when (entrada = ap_const_lv4_5)
else "0";
```

```
    tmp_6_fu_159_p2 <= "1" when (entrada = ap_const_lv4_6)
else "0";
    tmp_7_fu_165_p2 <= "1" when (entrada = ap_const_lv4_7)
else "0";
    tmp_8_fu_193_p2 <= "1" when (entrada = ap_const_lv4_8)
else "0";
    tmp_9_fu_199_p2 <= "1" when (entrada = ap_const_lv4_9)
else "0";
    tmp_fu_111_p2 <= (tmp_3_fu_97_p2 or tmp_2_fu_91_p2);
    tmp_s_fu_145_p2 <= (tmp_5_fu_131_p2 or tmp_4_fu_125_p2);
end behav;
```

Apêndice J – ContadorDecimal.pon

Código fonte em LingPON-HD 1.0 para o experimento do contador decimal.

```
fbe fbeContadorDecimal
 attributes
  integer atContClock 0
  integer atUnidade 0
  integer atDezena 0
  integer atCentena 0
  boolean atLigado false
 end_attributes
 methods
  method mtZera(atContClock = 0)
  method mtIncrementaCont(atContClock = atContClock + 1)
  method mtIncUnid(atUnidade = atUnidade + 1)
  method mtIncDezena(atDezena = atDezena + 1)
  method mtIncCentena(atCentena = atCentena + 1)
  method mtZeraUnidade(atUnidade = 0)
  method mtZeraDezena(atDezena = 0)
  method mtZeraCentena(atCentena = 0)
 end_methods
end_fbe

fbe fbeDisplay0
 attributes
  integer atDisp0 0
 end_attributes
 methods
  method mtDisp0Val0(atDisp0 = 64)
  method mtDisp0Val1(atDisp0 = 121)
  method mtDisp0Val2(atDisp0 = 36)
  method mtDisp0Val3(atDisp0 = 48)
  method mtDisp0Val4(atDisp0 = 25)
  method mtDisp0Val5(atDisp0 = 18)
  method mtDisp0Val6(atDisp0 = 2)
  method mtDisp0Val7(atDisp0 = 120)
  method mtDisp0Val8(atDisp0 = 0)
  method mtDisp0Val9(atDisp0 = 16)
 end_methods
end_fbe

fbe fbeDisplay1
 attributes
  integer atDisp1 0
 end_attributes
 methods
  method mtDisp1Val0(atDisp1 = 64)
  method mtDisp1Val1(atDisp1 = 121)
  method mtDisp1Val2(atDisp1 = 36)
  method mtDisp1Val3(atDisp1 = 48)
```

```
 method mtDisp1Val4(atDisp1 = 25)
 method mtDisp1Val5(atDisp1 = 18)
 method mtDisp1Val6(atDisp1 = 2)
 method mtDisp1Val7(atDisp1 = 120)
 method mtDisp1Val8(atDisp1 = 0)
 method mtDisp1Val9(atDisp1 = 16)
end_methods
end_fbe

fbe fbeDisplay2
 attributes
  integer atDisp2 0
 end_attributes
 methods
 method mtDisp2Val0(atDisp2 = 64)
 method mtDisp2Val1(atDisp2 = 121)
 method mtDisp2Val2(atDisp2 = 36)
 method mtDisp2Val3(atDisp2 = 48)
 method mtDisp2Val4(atDisp2 = 25)
 method mtDisp2Val5(atDisp2 = 18)
 method mtDisp2Val6(atDisp2 = 2)
 method mtDisp2Val7(atDisp2 = 120)
 method mtDisp2Val8(atDisp2 = 0)
 method mtDisp2Val9(atDisp2 = 16)
end_methods
end_fbe

inst
 fbeContadorDecimal contador
 fbeDisplay0 display0
 fbeDisplay1 display1
 fbeDisplay2 display2
end_inst

strategy
 no_one
end_strategy

rule rlZeCont
 condition
  subcondition SZeCont
   premise prContClockMaior contador.atContClock > 50000000
  end_subcondition
 end_condition
 action
  instigation inZera contador.mtZera();
 end_action
end_rule

rule rlInUnid
 condition
```

```
 subcondition SInUnid
  premise prContClockMaior contador.atContClock > 9
 end_subcondition
end_condition
action
 instigation inIncUnid contador.mtIncUnid();
end_action
end_rule

rule rlConta
condition
 subcondition SConta
  premise prOn contador.atLigado == true
 end_subcondition
end_condition
action
 instigation inIncrementaCont contador.mtIncrementaCont();
end_action
end_rule

rule rlZeUnidade
condition
 subcondition SZeUnidade
  premise prUnidadeMaior contador.atUnidade > 9
 end_subcondition
end_condition
action
 instigation inZeraUnidade contador.mtZeraUnidade();
end_action
end_rule

rule rlInDez
condition
 subcondition SInDez
  premise prUnidadeMaior contador.atUnidade > 9
 end_subcondition
end_condition
action
 instigation inIncDezena contador.mtIncDezena();
end_action
end_rule

rule rlZeDezena
condition
 subcondition SZeDezena
  premise prDezenaMaior contador.atDezena > 9
 end_subcondition
end_condition
action
 instigation inZeraDezena contador.mtZeraDezena();
end_action
```

```
end_rule

rule rlInCen
 condition
  subcondition SInCen
   premise  prDezenaMaior contador.atDezena > 9
  end_subcondition
 end_condition
 action
  instigation inIncCentena contador.mtIncCentena();
 end_action
end_rule

rule rlZeCen
 condition
  subcondition SZeCen
   premise prCentenaMaior contador.atCentena > 9
  end_subcondition
 end_condition
 action
  instigation inZeraCentena contador.mtZeraCentena();
 end_action
end_rule

// Display 0 digito 0
rule rlDisp0Val0
 condition
  subcondition SDisp0Val0
   premise prDisp0Val0 contador.atUnidade == 0
  end_subcondition
 end_condition
 action
  instigation inDisp0Val0 display0.mtDisp0Val0();
 end_action
end_rule
// Display 0 digito 1
rule rlDisp0Val1
 condition
  subcondition SDisp0Val1
   premise prDisp0Val1 contador.atUnidade == 1
  end_subcondition
 end_condition
 action
  instigation inDisp0Val1 display0.mtDisp0Val1();
 end_action
end_rule
// Display 0 digito 2
rule rlDisp0Val2
 condition
  subcondition SDisp0Val2
   premise prDisp0Val2 contador.atUnidade == 2
```

```
 end_subcondition
end_condition
action
 instigation inDisp0Val2 display0.mtDisp0Val2();
end_action
end_rule
// Display 0 digito 3
rule rlDisp0Val3
condition
 subcondition SDisp0Val3
  premise prDisp0Val3 contador.atUnidade == 3
 end_subcondition
end_condition
action
 instigation inDisp0Val3 display0.mtDisp0Val3();
end_action
end_rule
// Display 0 digito 4
rule rlDisp0Val4
condition
 subcondition SDisp0Val4
  premise prDisp0Val4 contador.atUnidade == 4
 end_subcondition
end_condition
action
 instigation inDisp0Val4 display0.mtDisp0Val4();
end_action
end_rule
// Display 0 digito 5
rule rlDisp0Val5
condition
 subcondition SDisp0Val5
  premise prDisp0Val5 contador.atUnidade == 5
 end_subcondition
end_condition
action
 instigation inDisp0Val5 display0.mtDisp0Val5();
end_action
end_rule
// Display 0 digito 6
rule rlDisp0Val6
condition
 subcondition SDisp0Val6
  premise prDisp0Val6 contador.atUnidade == 6
 end_subcondition
end_condition
action
 instigation inDisp0Val6 display0.mtDisp0Val6();
end_action
end_rule
// Display 0 digito 7
```

```
rule rlDisp0Val7
 condition
  subcondition SDisp0Val7
   premise prDisp0Val7 contador.atUnidade == 7
  end_subcondition
 end_condition
 action
  instigation inDisp0Val7 display0.mtDisp0Val7();
 end_action
end_rule
// Display 0 digito 8
rule rlDisp0Val8
 condition
  subcondition SDisp0Val8
   premise prDisp0Val8 contador.atUnidade == 8
  end_subcondition
 end_condition
 action
  instigation inDisp0Val8 display0.mtDisp0Val8();
 end_action
end_rule
// Display 0 digito 9
rule rlDisp0Val9
 condition
  subcondition SDisp0Val9
   premise prDisp0Val9 contador.atUnidade == 9
  end_subcondition
 end_condition
 action
  instigation inDisp0Val9 display0.mtDisp0Val9();
 end_action
end_rule
// fim display 0

// Display 1 digito 0
rule rlDisp1Val0
 condition
  subcondition SDisp1Val0
   premise prDisp1Val0 contador.atDezena == 0
  end_subcondition
 end_condition
 action
  instigation inDisp1Val0 display1.mtDisp1Val0();
 end_action
end_rule
// Display 1 digito 1
rule rlDisp1Val1
 condition
  subcondition SDisp1Val1
   premise prDisp1Val1 contador.atDezena == 1
  end_subcondition
```

```
 end_condition
 action
  instigation inDisp1Val1 display1.mtDisp1Val1();
 end_action
end_rule
// Display 1 digito 2
rule rlDisp1Val2
 condition
  subcondition SDisp1Val2
   premise prDisp1Val2 contador.atDezena == 2
  end_subcondition
 end_condition
 action
  instigation inDisp1Val2 display1.mtDisp1Val2();
 end_action
end_rule
// Display 1 digito 3
rule rlDisp1Val3
 condition
  subcondition SDisp1Val3
   premise prDisp1Val3 contador.atDezena == 3
  end_subcondition
 end_condition
 action
  instigation inDisp1Val3 display1.mtDisp1Val3();
 end_action
end_rule
// Display 1 digito 4
rule rlDisp1Val4
 condition
  subcondition SDisp1Val4
   premise prDisp1Val4 contador.atDezena == 4
  end_subcondition
 end_condition
 action
  instigation inDisp1Val4 display1.mtDisp1Val4();
 end_action
end_rule
// Display 1 digito 5
rule rlDisp1Val5
 condition
  subcondition SDisp1Val5
   premise prDisp1Val5 contador.atDezena == 5
  end_subcondition
 end_condition
 action
  instigation inDisp1Val5 display1.mtDisp1Val5();
 end_action
end_rule
// Display 1 digito 6
rule rlDisp1Val6
```

```
 condition
  subcondition SDisp1Val6
   premise prDisp1Val6 contador.atDezena == 6
  end_subcondition
 end_condition
 action
  instigation inDisp1Val6 display1.mtDisp1Val6();
 end_action
end_rule
// Display 1 digito 7
rule rlDisp1Val7
 condition
  subcondition SDisp1Val7
   premise prDisp1Val7 contador.atDezena == 7
  end_subcondition
 end_condition
 action
  instigation inDisp1Val7 display1.mtDisp1Val7();
 end_action
end_rule
// Display 1 digito 8
rule rlDisp1Val8
 condition
  subcondition SDisp1Val8
   premise prDisp1Val8 contador.atDezena == 8
  end_subcondition
 end_condition
 action
  instigation inDisp1Val8 display1.mtDisp1Val8();
 end_action
end_rule
// Display 1 digito 9
rule rlDisp1Val9
 condition
  subcondition SDisp1Val9
   premise prDisp1Val9 contador.atDezena == 9
  end_subcondition
 end_condition
 action
  instigation inDisp1Val9 display1.mtDisp1Val9();
 end_action
end_rule
// fim display 1

// Display 2 digito 0
rule rlDisp2Val0
 condition
  subcondition SDisp2Val0
   premise prDisp2Val0 contador.atCentena == 0
  end_subcondition
 end_condition
```

```
 action
  instigation inDisp2Val0 display2.mtDisp2Val0();
 end_action
end_rule
// Display 2 digito 1
rule rlDisp2Val1
 condition
  subcondition SDisp2Val1
   premise prDisp2Val1 contador.atCentena == 1
  end_subcondition
 end_condition
 action
  instigation inDisp2Val1 display2.mtDisp2Val1();
 end_action
end_rule
// Display 2 digito 2
rule rlDisp2Val2
 condition
  subcondition SDisp2Val2
   premise prDisp2Val2 contador.atCentena == 2
  end_subcondition
 end_condition
 action
  instigation inDisp2Val2 display2.mtDisp2Val2();
 end_action
end_rule
// Display 2 digito 3
rule rlDisp2Val3
 condition
  subcondition SDisp2Val3
   premise prDisp2Val3 contador.atCentena == 3
  end_subcondition
 end_condition
 action
  instigation inDisp2Val3 display2.mtDisp2Val3();
 end_action
end_rule
// Display 2 digito 4
rule rlDisp2Val4
 condition
  subcondition SDisp2Val4
   premise prDisp2Val4 contador.atCentena == 4
  end_subcondition
 end_condition
 action
  instigation inDisp2Val4 display2.mtDisp2Val4();
 end_action
end_rule
// Display 2 digito 5
rule rlDisp2Val5
 condition
```

```
 subcondition SDisp2Val5
  premise prDisp2Val5 contador.atCentena == 5
 end_subcondition
end_condition
action
 instigation inDisp2Val5 display2.mtDisp2Val5();
end_action
end_rule
// Display 2 digito 6
rule rlDisp2Val6
condition
 subcondition SDisp2Val6
  premise prDisp2Val6 contador.atCentena == 6
 end_subcondition
end_condition
action
 instigation inDisp2Val6 display2.mtDisp2Val6();
end_action
end_rule
// Display 2 digito 7
rule rlDisp2Val7
condition
 subcondition SDisp2Val7
  premise prDisp2Val7 contador.atCentena == 7
 end_subcondition
end_condition
action
 instigation inDisp2Val7 display2.mtDisp2Val7();
end_action
end_rule
// Display 2 digito 8
rule rlDisp2Val8
condition
 subcondition SDisp2Val8
  premise prDisp2Val8 contador.atCentena == 8
 end_subcondition
end_condition
action
 instigation inDisp2Val8 display2.mtDisp2Val8();
end_action
end_rule
// Display 2 digito 9
rule rlDisp2Val9
condition
 subcondition SDisp2Val9
  premise prDisp2Val9 contador.atCentena == 9
 end_subcondition
end_condition
action
 instigation inDisp2Val9 display2.mtDisp2Val9();
end_action
```

```
end_rule
// fim display 2

main {
 entity ContadorDecimal

 in contador.atLigado

 out display0.atDisp0
 out display1.atDisp1
 out display2.atDisp2

 NBits contador.atUnidade 5
 NBits contador.atDezena 5
 NBits contador.atCentena 5

 NBits display0.atDisp0 7
 NBits display1.atDisp1 7
 NBits display2.atDisp2 7

 NBits contador.atContClock 27
 }
```

Apêndice K – ContadorDecimal.vhd

Código fonte em VHDL para o experimento do contador decimal gerado automaticamente pelo Compilador PON-HD 1.0.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.data_type_pkg.all;

ENTITY ContadorDecimal IS
PORT(
 in_contador_atLigado :IN STD_LOGIC;
 in_set_contador_atLigado :IN STD_LOGIC;
 out_display0_atDisp0 :OUT STD_LOGIC_VECTOR(7-1 downto 0);
 out_display1_atDisp1 :OUT STD_LOGIC_VECTOR(7-1 downto 0);
 out_display2_atDisp2 :OUT STD_LOGIC_VECTOR(7-1 downto 0);
 clock :IN STD_LOGIC
 );
END ContadorDecimal;

ARCHITECTURE ContadorDecimal_arq OF ContadorDecimal IS
----------------- COMPONENTS --------------------
COMPONENT NOP_attribute
 GENERIC (
 N_bits: INTEGER;
 N_new_values: INTEGER;
 initial_value: STD_LOGIC_VECTOR
 );
 PORT(
 att_clock :IN STD_LOGIC;
 att_new_value :IN data(0 TO N_new_values-1)(N_bits-1 downto
0);
 att_set_value :IN STD_LOGIC_VECTOR(N_new_values-1 downto 0);
 att_value :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0)
 );
END COMPONENT;
COMPONENT NOP_premise
 GENERIC (
 N_bits: INTEGER:=1;
 dataType: INTEGER:=0;
 operation: INTEGER:=1 --EQUAL 1; DIFFERENT 2; LESS_THAN 3;
GREATER_THAN 4; LESS_EQUAL 5; GREATER_EQUAL 6;
 );
 PORT(
 att_value_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 att_value_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 pre_result :OUT STD_LOGIC
 );
END COMPONENT;
COMPONENT NOP_method
 GENERIC (
```

```
 N_bits: INTEGER:=1;
 dataType: INTEGER:=1;
operation: INTEGER:=1
);
PORT(
in_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
in_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
execute :IN STD_LOGIC;
result :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0);
notify :OUT STD_LOGIC
);
END COMPONENT;
----------------- SIGNALS ---------------------
SIGNAL contador_atContClock_new_value: data(0 TO 2-1)(27-1
downto 0);
SIGNAL contador_atUnidade_new_value: data(0 TO 2-1)(5-1 downto
0);
SIGNAL contador_atDezena_new_value: data(0 TO 2-1)(5-1 downto
0);
SIGNAL contador_atCentena_new_value: data(0 TO 2-1)(5-1 downto
0);
SIGNAL contador_atLigado_new_value: data(0 TO 1-1)(0 downto
0);
SIGNAL display0_atDisp0_new_value: data(0 TO 10-1)(7-1 downto
0);
SIGNAL display1_atDisp1_new_value: data(0 TO 10-1)(7-1 downto
0);
SIGNAL display2_atDisp2_new_value: data(0 TO 10-1)(7-1 downto
0);

SIGNAL contador_atContClock_value: STD_LOGIC_VECTOR(27-1
downto 0);
SIGNAL contador_atUnidade_value: STD_LOGIC_VECTOR(5-1 downto
0);
SIGNAL contador_atDezena_value: STD_LOGIC_VECTOR(5-1 downto
0);
SIGNAL contador_atCentena_value: STD_LOGIC_VECTOR(5-1 downto
0);
SIGNAL contador_atLigado_value: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL display0_atDisp0_value: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL display1_atDisp1_value: STD_LOGIC_VECTOR(7-1 downto 0);
SIGNAL display2_atDisp2_value: STD_LOGIC_VECTOR(7-1 downto 0);

SIGNAL contador_atContClock_set: STD_LOGIC_VECTOR(2-1 downto
0);
SIGNAL contador_atUnidade_set: STD_LOGIC_VECTOR(2-1 downto 0);
SIGNAL contador_atDezena_set: STD_LOGIC_VECTOR(2-1 downto 0);
SIGNAL contador_atCentena_set: STD_LOGIC_VECTOR(2-1 downto 0);
SIGNAL contador_atLigado_set: STD_LOGIC_VECTOR(1-1 downto 0);
SIGNAL display0_atDisp0_set: STD_LOGIC_VECTOR(10-1 downto 0);
SIGNAL display1_atDisp1_set: STD_LOGIC_VECTOR(10-1 downto 0);
```

```
SIGNAL display2_atDisp2_set: STD_LOGIC_VECTOR(10-1 downto 0);

SIGNAL prCentenaMaior_result: STD_LOGIC;
SIGNAL prContClockMaior_result: STD_LOGIC;
SIGNAL prDezenaMaior_result: STD_LOGIC;
SIGNAL prDisp0Val0_result: STD_LOGIC;
SIGNAL prDisp0Val1_result: STD_LOGIC;
SIGNAL prDisp0Val2_result: STD_LOGIC;
SIGNAL prDisp0Val3_result: STD_LOGIC;
SIGNAL prDisp0Val4_result: STD_LOGIC;
SIGNAL prDisp0Val5_result: STD_LOGIC;
SIGNAL prDisp0Val6_result: STD_LOGIC;
SIGNAL prDisp0Val7_result: STD_LOGIC;
SIGNAL prDisp0Val8_result: STD_LOGIC;
SIGNAL prDisp0Val9_result: STD_LOGIC;
SIGNAL prDisp1Val0_result: STD_LOGIC;
SIGNAL prDisp1Val1_result: STD_LOGIC;
SIGNAL prDisp1Val2_result: STD_LOGIC;
SIGNAL prDisp1Val3_result: STD_LOGIC;
SIGNAL prDisp1Val4_result: STD_LOGIC;
SIGNAL prDisp1Val5_result: STD_LOGIC;
SIGNAL prDisp1Val6_result: STD_LOGIC;
SIGNAL prDisp1Val7_result: STD_LOGIC;
SIGNAL prDisp1Val8_result: STD_LOGIC;
SIGNAL prDisp1Val9_result: STD_LOGIC;
SIGNAL prDisp2Val0_result: STD_LOGIC;
SIGNAL prDisp2Val1_result: STD_LOGIC;
SIGNAL prDisp2Val2_result: STD_LOGIC;
SIGNAL prDisp2Val3_result: STD_LOGIC;
SIGNAL prDisp2Val4_result: STD_LOGIC;
SIGNAL prDisp2Val5_result: STD_LOGIC;
SIGNAL prDisp2Val6_result: STD_LOGIC;
SIGNAL prDisp2Val7_result: STD_LOGIC;
SIGNAL prDisp2Val8_result: STD_LOGIC;
SIGNAL prDisp2Val9_result: STD_LOGIC;
SIGNAL prOn_result: STD_LOGIC;
SIGNAL prUnidadeMaior_result: STD_LOGIC;

SIGNAL prCentenaMaior_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prContClockMaior_imput_a: STD_LOGIC_VECTOR(27-1 downto
0);
SIGNAL prDezenaMaior_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val0_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val1_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val2_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val3_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val4_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val5_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val6_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val7_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val8_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
```

```
SIGNAL prDisp0Val9_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val0_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val1_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val2_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val3_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val4_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val5_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val6_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val7_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val8_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val9_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val0_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val1_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val2_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val3_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val4_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val5_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val6_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val7_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val8_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val9_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prOn_imput_a: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL prUnidadeMaior_imput_a: STD_LOGIC_VECTOR(5-1 downto 0);

SIGNAL prCentenaMaior_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prContClockMaior_imput_b: STD_LOGIC_VECTOR(27-1 downto
0);
SIGNAL prDezenaMaior_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val0_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val1_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val2_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val3_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val4_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val5_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val6_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val7_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val8_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp0Val9_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val0_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val1_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val2_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val3_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val4_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val5_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val6_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val7_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val8_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp1Val9_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val0_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val1_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
SIGNAL prDisp2Val2_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
```

```
 SIGNAL prDisp2Val3_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
 SIGNAL prDisp2Val4_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
 SIGNAL prDisp2Val5_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
 SIGNAL prDisp2Val6_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
 SIGNAL prDisp2Val7_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
 SIGNAL prDisp2Val8_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
 SIGNAL prDisp2Val9_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);
 SIGNAL prOn_imput_b: STD_LOGIC_VECTOR(0 downto 0);
 SIGNAL prUnidadeMaior_imput_b: STD_LOGIC_VECTOR(5-1 downto 0);

 SIGNAL contador_mtZera_input_a: STD_LOGIC_VECTOR(27-1 downto
0);
 SIGNAL contador_mtIncrementaCont_input_a: STD_LOGIC_VECTOR(27-
1 downto 0);
 SIGNAL contador_mtIncUnid_input_a: STD_LOGIC_VECTOR(5-1 downto
0);
 SIGNAL contador_mtIncDezena_input_a: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtIncCentena_input_a: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtZeraUnidade_input_a: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtZeraDezena_input_a: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtZeraCentena_input_a: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL display0_mtDisp0Val0_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val1_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val2_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val3_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val4_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val5_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val6_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val7_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val8_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val9_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val0_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val1_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
```

```
 SIGNAL display1_mtDisp1Val2_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val3_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val4_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val5_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val6_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val7_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val8_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val9_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val0_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val1_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val2_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val3_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val4_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val5_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val6_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val7_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val8_input_a: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val9_input_a: STD_LOGIC_VECTOR(7-1
downto 0);

 SIGNAL contador_mtZera_input_b: STD_LOGIC_VECTOR(27-1 downto
0);
 SIGNAL contador_mtIncrementaCont_input_b: STD_LOGIC_VECTOR(27-
1 downto 0);
 SIGNAL contador_mtIncUnid_input_b: STD_LOGIC_VECTOR(5-1 downto
0);
 SIGNAL contador_mtIncDezena_input_b: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtIncCentena_input_b: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtZeraUnidade_input_b: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtZeraDezena_input_b: STD_LOGIC_VECTOR(5-1
downto 0);
```

```
 SIGNAL contador_mtZeraCentena_input_b: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL display0_mtDisp0Val0_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val1_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val2_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val3_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val4_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val5_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val6_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val7_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val8_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val9_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val0_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val1_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val2_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val3_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val4_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val5_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val6_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val7_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val8_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val9_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val0_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val1_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val2_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val3_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
```

```
 SIGNAL display2_mtDisp2Val4_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val5_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val6_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val7_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val8_input_b: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val9_input_b: STD_LOGIC_VECTOR(7-1
downto 0);

 SIGNAL contador_mtZera_output: STD_LOGIC_VECTOR(27-1 downto
0);
 SIGNAL contador_mtIncrementaCont_output: STD_LOGIC_VECTOR(27-1
downto 0);
 SIGNAL contador_mtIncUnid_output: STD_LOGIC_VECTOR(5-1 downto
0);
 SIGNAL contador_mtIncDezena_output: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtIncCentena_output: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtZeraUnidade_output: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtZeraDezena_output: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL contador_mtZeraCentena_output: STD_LOGIC_VECTOR(5-1
downto 0);
 SIGNAL display0_mtDisp0Val0_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val1_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val2_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val3_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val4_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val5_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val6_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val7_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val8_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display0_mtDisp0Val9_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val0_output: STD_LOGIC_VECTOR(7-1
downto 0);
```

```
 SIGNAL display1_mtDisp1Val1_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val2_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val3_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val4_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val5_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val6_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val7_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val8_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display1_mtDisp1Val9_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val0_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val1_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val2_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val3_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val4_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val5_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val6_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val7_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val8_output: STD_LOGIC_VECTOR(7-1
downto 0);
 SIGNAL display2_mtDisp2Val9_output: STD_LOGIC_VECTOR(7-1
downto 0);

 SIGNAL contador_mtZera_execute: STD_LOGIC;
 SIGNAL contador_mtIncrementaCont_execute: STD_LOGIC;
 SIGNAL contador_mtIncUnid_execute: STD_LOGIC;
 SIGNAL contador_mtIncDezena_execute: STD_LOGIC;
 SIGNAL contador_mtIncCentena_execute: STD_LOGIC;
 SIGNAL contador_mtZeraUnidade_execute: STD_LOGIC;
 SIGNAL contador_mtZeraDezena_execute: STD_LOGIC;
 SIGNAL contador_mtZeraCentena_execute: STD_LOGIC;
 SIGNAL display0_mtDisp0Val0_execute: STD_LOGIC;
 SIGNAL display0_mtDisp0Val1_execute: STD_LOGIC;
 SIGNAL display0_mtDisp0Val2_execute: STD_LOGIC;
 SIGNAL display0_mtDisp0Val3_execute: STD_LOGIC;
```

```
SIGNAL display0_mtDisp0Val4_execute: STD_LOGIC;
SIGNAL display0_mtDisp0Val5_execute: STD_LOGIC;
SIGNAL display0_mtDisp0Val6_execute: STD_LOGIC;
SIGNAL display0_mtDisp0Val7_execute: STD_LOGIC;
SIGNAL display0_mtDisp0Val8_execute: STD_LOGIC;
SIGNAL display0_mtDisp0Val9_execute: STD_LOGIC;
SIGNAL display1_mtDisp1Val0_execute: STD_LOGIC;
SIGNAL display1_mtDisp1Val1_execute: STD_LOGIC;
SIGNAL display1_mtDisp1Val2_execute: STD_LOGIC;
SIGNAL display1_mtDisp1Val3_execute: STD_LOGIC;
SIGNAL display1_mtDisp1Val4_execute: STD_LOGIC;
SIGNAL display1_mtDisp1Val5_execute: STD_LOGIC;
SIGNAL display1_mtDisp1Val6_execute: STD_LOGIC;
SIGNAL display1_mtDisp1Val7_execute: STD_LOGIC;
SIGNAL display1_mtDisp1Val8_execute: STD_LOGIC;
SIGNAL display1_mtDisp1Val9_execute: STD_LOGIC;
SIGNAL display2_mtDisp2Val0_execute: STD_LOGIC;
SIGNAL display2_mtDisp2Val1_execute: STD_LOGIC;
SIGNAL display2_mtDisp2Val2_execute: STD_LOGIC;
SIGNAL display2_mtDisp2Val3_execute: STD_LOGIC;
SIGNAL display2_mtDisp2Val4_execute: STD_LOGIC;
SIGNAL display2_mtDisp2Val5_execute: STD_LOGIC;
SIGNAL display2_mtDisp2Val6_execute: STD_LOGIC;
SIGNAL display2_mtDisp2Val7_execute: STD_LOGIC;
SIGNAL display2_mtDisp2Val8_execute: STD_LOGIC;
SIGNAL display2_mtDisp2Val9_execute: STD_LOGIC;

SIGNAL contador_mtZera_notify: STD_LOGIC;
SIGNAL contador_mtIncrementaCont_notify: STD_LOGIC;
SIGNAL contador_mtIncUnid_notify: STD_LOGIC;
SIGNAL contador_mtIncDezena_notify: STD_LOGIC;
SIGNAL contador_mtIncCentena_notify: STD_LOGIC;
SIGNAL contador_mtZeraUnidade_notify: STD_LOGIC;
SIGNAL contador_mtZeraDezena_notify: STD_LOGIC;
SIGNAL contador_mtZeraCentena_notify: STD_LOGIC;
SIGNAL display0_mtDisp0Val0_notify: STD_LOGIC;
SIGNAL display0_mtDisp0Val1_notify: STD_LOGIC;
SIGNAL display0_mtDisp0Val2_notify: STD_LOGIC;
SIGNAL display0_mtDisp0Val3_notify: STD_LOGIC;
SIGNAL display0_mtDisp0Val4_notify: STD_LOGIC;
SIGNAL display0_mtDisp0Val5_notify: STD_LOGIC;
SIGNAL display0_mtDisp0Val6_notify: STD_LOGIC;
SIGNAL display0_mtDisp0Val7_notify: STD_LOGIC;
SIGNAL display0_mtDisp0Val8_notify: STD_LOGIC;
SIGNAL display0_mtDisp0Val9_notify: STD_LOGIC;
SIGNAL display1_mtDisp1Val0_notify: STD_LOGIC;
SIGNAL display1_mtDisp1Val1_notify: STD_LOGIC;
SIGNAL display1_mtDisp1Val2_notify: STD_LOGIC;
SIGNAL display1_mtDisp1Val3_notify: STD_LOGIC;
SIGNAL display1_mtDisp1Val4_notify: STD_LOGIC;
SIGNAL display1_mtDisp1Val5_notify: STD_LOGIC;
```

```
 SIGNAL display1_mtDisp1Val6_notify: STD_LOGIC;
 SIGNAL display1_mtDisp1Val7_notify: STD_LOGIC;
 SIGNAL display1_mtDisp1Val8_notify: STD_LOGIC;
 SIGNAL display1_mtDisp1Val9_notify: STD_LOGIC;
 SIGNAL display2_mtDisp2Val0_notify: STD_LOGIC;
 SIGNAL display2_mtDisp2Val1_notify: STD_LOGIC;
 SIGNAL display2_mtDisp2Val2_notify: STD_LOGIC;
 SIGNAL display2_mtDisp2Val3_notify: STD_LOGIC;
 SIGNAL display2_mtDisp2Val4_notify: STD_LOGIC;
 SIGNAL display2_mtDisp2Val5_notify: STD_LOGIC;
 SIGNAL display2_mtDisp2Val6_notify: STD_LOGIC;
 SIGNAL display2_mtDisp2Val7_notify: STD_LOGIC;
 SIGNAL display2_mtDisp2Val8_notify: STD_LOGIC;
 SIGNAL display2_mtDisp2Val9_notify: STD_LOGIC;
-------------------------------------------------
BEGIN
---------- PON ELEMENTS INSTANCES -------------
 contador_atContClock: NOP_attribute GENERIC MAP(N_bits => 27,
N_new_values => 2, initial_value =>
"000000000000000000000000000") PORT MAP(att_clock => clock,
att_new_value => contador_atContClock_new_value, att_set_value
=> contador_atContClock_set, att_value =>
contador_atContClock_value);
 contador_atUnidade: NOP_attribute GENERIC MAP(N_bits => 5,
N_new_values => 2, initial_value => "00000") PORT
MAP(att_clock => clock, att_new_value =>
contador_atUnidade_new_value, att_set_value =>
contador_atUnidade_set, att_value =>
contador_atUnidade_value);
 contador_atDezena: NOP_attribute GENERIC MAP(N_bits => 5,
N_new_values => 2, initial_value => "00000") PORT
MAP(att_clock => clock, att_new_value =>
contador_atDezena_new_value, att_set_value =>
contador_atDezena_set, att_value => contador_atDezena_value);
 contador_atCentena: NOP_attribute GENERIC MAP(N_bits => 5,
N_new_values => 2, initial_value => "00000") PORT
MAP(att_clock => clock, att_new_value =>
contador_atCentena_new_value, att_set_value =>
contador_atCentena_set, att_value =>
contador_atCentena_value);
 contador_atLigado: NOP_attribute GENERIC MAP(N_bits => 1,
N_new_values => 1, initial_value => "0") PORT MAP(att_clock =>
clock, att_new_value => contador_atLigado_new_value,
att_set_value => contador_atLigado_set, att_value =>
contador_atLigado_value);
 display0_atDisp0: NOP_attribute GENERIC MAP(N_bits => 7,
N_new_values => 10, initial_value => "0000000") PORT
MAP(att_clock => clock, att_new_value =>
display0_atDisp0_new_value, att_set_value =>
display0_atDisp0_set, att_value => display0_atDisp0_value);
```

```
 display1_atDisp1: NOP_attribute GENERIC MAP(N_bits => 7,
N_new_values => 10, initial_value => "0000000") PORT
MAP(att_clock => clock, att_new_value =>
display1_atDisp1_new_value, att_set_value =>
display1_atDisp1_set, att_value => display1_atDisp1_value);
 display2_atDisp2: NOP_attribute GENERIC MAP(N_bits => 7,
N_new_values => 10, initial_value => "0000000") PORT
MAP(att_clock => clock, att_new_value =>
display2_atDisp2_new_value, att_set_value =>
display2_atDisp2_set, att_value => display2_atDisp2_value);

 prCentenaMaior_GREATER: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 4) PORT MAP(att_value_a =>
prCentenaMaior_imput_a, att_value_b => prCentenaMaior_imput_b,
pre_result => prCentenaMaior_result);
 prContClockMaior_GREATER: NOP_premise GENERIC MAP(N_bits =>
27, dataType => 0, operation => 4) PORT MAP(att_value_a =>
prContClockMaior_imput_a, att_value_b =>
prContClockMaior_imput_b, pre_result =>
prContClockMaior_result);
 prDezenaMaior_GREATER: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 4) PORT MAP(att_value_a =>
prDezenaMaior_imput_a, att_value_b => prDezenaMaior_imput_b,
pre_result => prDezenaMaior_result);
 prDisp0Val0_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp0Val0_imput_a, att_value_b => prDisp0Val0_imput_b,
pre_result => prDisp0Val0_result);
 prDisp0Val1_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp0Val1_imput_a, att_value_b => prDisp0Val1_imput_b,
pre_result => prDisp0Val1_result);
 prDisp0Val2_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp0Val2_imput_a, att_value_b => prDisp0Val2_imput_b,
pre_result => prDisp0Val2_result);
 prDisp0Val3_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp0Val3_imput_a, att_value_b => prDisp0Val3_imput_b,
pre_result => prDisp0Val3_result);
 prDisp0Val4_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp0Val4_imput_a, att_value_b => prDisp0Val4_imput_b,
pre_result => prDisp0Val4_result);
 prDisp0Val5_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp0Val5_imput_a, att_value_b => prDisp0Val5_imput_b,
pre_result => prDisp0Val5_result);
 prDisp0Val6_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
```

```
prDisp0Val6_imput_a, att_value_b => prDisp0Val6_imput_b,
pre_result => prDisp0Val6_result);
 prDisp0Val7_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp0Val7_imput_a, att_value_b => prDisp0Val7_imput_b,
pre_result => prDisp0Val7_result);
 prDisp0Val8_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp0Val8_imput_a, att_value_b => prDisp0Val8_imput_b,
pre_result => prDisp0Val8_result);
 prDisp0Val9_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp0Val9_imput_a, att_value_b => prDisp0Val9_imput_b,
pre_result => prDisp0Val9_result);
 prDisp1Val0_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp1Val0_imput_a, att_value_b => prDisp1Val0_imput_b,
pre_result => prDisp1Val0_result);
 prDisp1Val1_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp1Val1_imput_a, att_value_b => prDisp1Val1_imput_b,
pre_result => prDisp1Val1_result);
 prDisp1Val2_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp1Val2_imput_a, att_value_b => prDisp1Val2_imput_b,
pre_result => prDisp1Val2_result);
 prDisp1Val3_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp1Val3_imput_a, att_value_b => prDisp1Val3_imput_b,
pre_result => prDisp1Val3_result);
 prDisp1Val4_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp1Val4_imput_a, att_value_b => prDisp1Val4_imput_b,
pre_result => prDisp1Val4_result);
 prDisp1Val5_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp1Val5_imput_a, att_value_b => prDisp1Val5_imput_b,
pre_result => prDisp1Val5_result);
 prDisp1Val6_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp1Val6_imput_a, att_value_b => prDisp1Val6_imput_b,
pre_result => prDisp1Val6_result);
 prDisp1Val7_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp1Val7_imput_a, att_value_b => prDisp1Val7_imput_b,
pre_result => prDisp1Val7_result);
 prDisp1Val8_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp1Val8_imput_a, att_value_b => prDisp1Val8_imput_b,
pre_result => prDisp1Val8_result);
```

```
prDisp1Val9_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp1Val9_imput_a, att_value_b => prDisp1Val9_imput_b,
pre_result => prDisp1Val9_result);
prDisp2Val0_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp2Val0_imput_a, att_value_b => prDisp2Val0_imput_b,
pre_result => prDisp2Val0_result);
prDisp2Val1_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp2Val1_imput_a, att_value_b => prDisp2Val1_imput_b,
pre_result => prDisp2Val1_result);
prDisp2Val2_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp2Val2_imput_a, att_value_b => prDisp2Val2_imput_b,
pre_result => prDisp2Val2_result);
prDisp2Val3_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp2Val3_imput_a, att_value_b => prDisp2Val3_imput_b,
pre_result => prDisp2Val3_result);
prDisp2Val4_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp2Val4_imput_a, att_value_b => prDisp2Val4_imput_b,
pre_result => prDisp2Val4_result);
prDisp2Val5_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp2Val5_imput_a, att_value_b => prDisp2Val5_imput_b,
pre_result => prDisp2Val5_result);
prDisp2Val6_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp2Val6_imput_a, att_value_b => prDisp2Val6_imput_b,
pre_result => prDisp2Val6_result);
prDisp2Val7_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp2Val7_imput_a, att_value_b => prDisp2Val7_imput_b,
pre_result => prDisp2Val7_result);
prDisp2Val8_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp2Val8_imput_a, att_value_b => prDisp2Val8_imput_b,
pre_result => prDisp2Val8_result);
prDisp2Val9_EQUAL: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 1) PORT MAP(att_value_a =>
prDisp2Val9_imput_a, att_value_b => prDisp2Val9_imput_b,
pre_result => prDisp2Val9_result);
prOn_EQUAL: NOP_premise GENERIC MAP(N_bits => 1, dataType =>
0, operation => 1) PORT MAP(att_value_a => prOn_imput_a,
att_value_b => prOn_imput_b, pre_result => prOn_result);
prUnidadeMaior_GREATER: NOP_premise GENERIC MAP(N_bits => 5,
dataType => 0, operation => 4) PORT MAP(att_value_a =>
prUnidadeMaior_imput_a, att_value_b => prUnidadeMaior_imput_b,
pre_result => prUnidadeMaior_result);
```

```
 contador_mtZera: NOP_method GENERIC MAP(N_bits => 27, dataType
=> 1, operation => 0) PORT MAP(in_a =>
contador_mtZera_input_a, in_b => contador_mtZera_input_b,
execute => contador_mtZera_execute, result =>
contador_mtZera_output, notify => contador_mtZera_notify);
 contador_mtIncrementaCont: NOP_method GENERIC MAP(N_bits =>
27, dataType => 1, operation => 1) PORT MAP(in_a =>
contador_mtIncrementaCont_input_a, in_b =>
contador_mtIncrementaCont_input_b, execute =>
contador_mtIncrementaCont_execute, result =>
contador_mtIncrementaCont_output, notify =>
contador_mtIncrementaCont_notify);
 contador_mtIncUnid: NOP_method GENERIC MAP(N_bits => 5,
dataType => 1, operation => 1) PORT MAP(in_a =>
contador_mtIncUnid_input_a, in_b =>
contador_mtIncUnid_input_b, execute =>
contador_mtIncUnid_execute, result =>
contador_mtIncUnid_output, notify =>
contador_mtIncUnid_notify);
 contador_mtIncDezena: NOP_method GENERIC MAP(N_bits => 5,
dataType => 1, operation => 1) PORT MAP(in_a =>
contador_mtIncDezena_input_a, in_b =>
contador_mtIncDezena_input_b, execute =>
contador_mtIncDezena_execute, result =>
contador_mtIncDezena_output, notify =>
contador_mtIncDezena_notify);
 contador_mtIncCentena: NOP_method GENERIC MAP(N_bits => 5,
dataType => 1, operation => 1) PORT MAP(in_a =>
contador_mtIncCentena_input_a, in_b =>
contador_mtIncCentena_input_b, execute =>
contador_mtIncCentena_execute, result =>
contador_mtIncCentena_output, notify =>
contador_mtIncCentena_notify);
 contador_mtZeraUnidade: NOP_method GENERIC MAP(N_bits => 5,
dataType => 1, operation => 0) PORT MAP(in_a =>
contador_mtZeraUnidade_input_a, in_b =>
contador_mtZeraUnidade_input_b, execute =>
contador_mtZeraUnidade_execute, result =>
contador_mtZeraUnidade_output, notify =>
contador_mtZeraUnidade_notify);
 contador_mtZeraDezena: NOP_method GENERIC MAP(N_bits => 5,
dataType => 1, operation => 0) PORT MAP(in_a =>
contador_mtZeraDezena_input_a, in_b =>
contador_mtZeraDezena_input_b, execute =>
contador_mtZeraDezena_execute, result =>
contador_mtZeraDezena_output, notify =>
contador_mtZeraDezena_notify);
 contador_mtZeraCentena: NOP_method GENERIC MAP(N_bits => 5,
dataType => 1, operation => 0) PORT MAP(in_a =>
contador_mtZeraCentena_input_a, in_b =>
```

```
contador_mtZeraCentena_input_b, execute =>
contador_mtZeraCentena_execute, result =>
contador_mtZeraCentena_output, notify =>
contador_mtZeraCentena_notify);
 display0_mtDisp0Val0: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display0_mtDisp0Val0_input_a, in_b =>
display0_mtDisp0Val0_input_b, execute =>
display0_mtDisp0Val0_execute, result =>
display0_mtDisp0Val0_output, notify =>
display0_mtDisp0Val0_notify);
 display0_mtDisp0Val1: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display0_mtDisp0Val1_input_a, in_b =>
display0_mtDisp0Val1_input_b, execute =>
display0_mtDisp0Val1_execute, result =>
display0_mtDisp0Val1_output, notify =>
display0_mtDisp0Val1_notify);
 display0_mtDisp0Val2: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display0_mtDisp0Val2_input_a, in_b =>
display0_mtDisp0Val2_input_b, execute =>
display0_mtDisp0Val2_execute, result =>
display0_mtDisp0Val2_output, notify =>
display0_mtDisp0Val2_notify);
 display0_mtDisp0Val3: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display0_mtDisp0Val3_input_a, in_b =>
display0_mtDisp0Val3_input_b, execute =>
display0_mtDisp0Val3_execute, result =>
display0_mtDisp0Val3_output, notify =>
display0_mtDisp0Val3_notify);
 display0_mtDisp0Val4: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display0_mtDisp0Val4_input_a, in_b =>
display0_mtDisp0Val4_input_b, execute =>
display0_mtDisp0Val4_execute, result =>
display0_mtDisp0Val4_output, notify =>
display0_mtDisp0Val4_notify);
 display0_mtDisp0Val5: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display0_mtDisp0Val5_input_a, in_b =>
display0_mtDisp0Val5_input_b, execute =>
display0_mtDisp0Val5_execute, result =>
display0_mtDisp0Val5_output, notify =>
display0_mtDisp0Val5_notify);
 display0_mtDisp0Val6: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display0_mtDisp0Val6_input_a, in_b =>
display0_mtDisp0Val6_input_b, execute =>
display0_mtDisp0Val6_execute, result =>
```

```
display0_mtDisp0Val6_output, notify =>
display0_mtDisp0Val6_notify);
 display0_mtDisp0Val7: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display0_mtDisp0Val7_input_a, in_b =>
display0_mtDisp0Val7_input_b, execute =>
display0_mtDisp0Val7_execute, result =>
display0_mtDisp0Val7_output, notify =>
display0_mtDisp0Val7_notify);
 display0_mtDisp0Val8: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display0_mtDisp0Val8_input_a, in_b =>
display0_mtDisp0Val8_input_b, execute =>
display0_mtDisp0Val8_execute, result =>
display0_mtDisp0Val8_output, notify =>
display0_mtDisp0Val8_notify);
 display0_mtDisp0Val9: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display0_mtDisp0Val9_input_a, in_b =>
display0_mtDisp0Val9_input_b, execute =>
display0_mtDisp0Val9_execute, result =>
display0_mtDisp0Val9_output, notify =>
display0_mtDisp0Val9_notify);
 display1_mtDisp1Val0: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display1_mtDisp1Val0_input_a, in_b =>
display1_mtDisp1Val0_input_b, execute =>
display1_mtDisp1Val0_execute, result =>
display1_mtDisp1Val0_output, notify =>
display1_mtDisp1Val0_notify);
 display1_mtDisp1Val1: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display1_mtDisp1Val1_input_a, in_b =>
display1_mtDisp1Val1_input_b, execute =>
display1_mtDisp1Val1_execute, result =>
display1_mtDisp1Val1_output, notify =>
display1_mtDisp1Val1_notify);
 display1_mtDisp1Val2: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display1_mtDisp1Val2_input_a, in_b =>
display1_mtDisp1Val2_input_b, execute =>
display1_mtDisp1Val2_execute, result =>
display1_mtDisp1Val2_output, notify =>
display1_mtDisp1Val2_notify);
 display1_mtDisp1Val3: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display1_mtDisp1Val3_input_a, in_b =>
display1_mtDisp1Val3_input_b, execute =>
display1_mtDisp1Val3_execute, result =>
display1_mtDisp1Val3_output, notify =>
display1_mtDisp1Val3_notify);
```

```
 display1_mtDisp1Val4: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display1_mtDisp1Val4_input_a, in_b =>
display1_mtDisp1Val4_input_b, execute =>
display1_mtDisp1Val4_execute, result =>
display1_mtDisp1Val4_output, notify =>
display1_mtDisp1Val4_notify);
 display1_mtDisp1Val5: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display1_mtDisp1Val5_input_a, in_b =>
display1_mtDisp1Val5_input_b, execute =>
display1_mtDisp1Val5_execute, result =>
display1_mtDisp1Val5_output, notify =>
display1_mtDisp1Val5_notify);
 display1_mtDisp1Val6: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display1_mtDisp1Val6_input_a, in_b =>
display1_mtDisp1Val6_input_b, execute =>
display1_mtDisp1Val6_execute, result =>
display1_mtDisp1Val6_output, notify =>
display1_mtDisp1Val6_notify);
 display1_mtDisp1Val7: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display1_mtDisp1Val7_input_a, in_b =>
display1_mtDisp1Val7_input_b, execute =>
display1_mtDisp1Val7_execute, result =>
display1_mtDisp1Val7_output, notify =>
display1_mtDisp1Val7_notify);
 display1_mtDisp1Val8: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display1_mtDisp1Val8_input_a, in_b =>
display1_mtDisp1Val8_input_b, execute =>
display1_mtDisp1Val8_execute, result =>
display1_mtDisp1Val8_output, notify =>
display1_mtDisp1Val8_notify);
 display1_mtDisp1Val9: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display1_mtDisp1Val9_input_a, in_b =>
display1_mtDisp1Val9_input_b, execute =>
display1_mtDisp1Val9_execute, result =>
display1_mtDisp1Val9_output, notify =>
display1_mtDisp1Val9_notify);
 display2_mtDisp2Val0: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display2_mtDisp2Val0_input_a, in_b =>
display2_mtDisp2Val0_input_b, execute =>
display2_mtDisp2Val0_execute, result =>
display2_mtDisp2Val0_output, notify =>
display2_mtDisp2Val0_notify);
 display2_mtDisp2Val1: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
```

```
display2_mtDisp2Val1_input_a, in_b =>
display2_mtDisp2Val1_input_b, execute =>
display2_mtDisp2Val1_execute, result =>
display2_mtDisp2Val1_output, notify =>
display2_mtDisp2Val1_notify);
 display2_mtDisp2Val2: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display2_mtDisp2Val2_input_a, in_b =>
display2_mtDisp2Val2_input_b, execute =>
display2_mtDisp2Val2_execute, result =>
display2_mtDisp2Val2_output, notify =>
display2_mtDisp2Val2_notify);
 display2_mtDisp2Val3: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display2_mtDisp2Val3_input_a, in_b =>
display2_mtDisp2Val3_input_b, execute =>
display2_mtDisp2Val3_execute, result =>
display2_mtDisp2Val3_output, notify =>
display2_mtDisp2Val3_notify);
 display2_mtDisp2Val4: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display2_mtDisp2Val4_input_a, in_b =>
display2_mtDisp2Val4_input_b, execute =>
display2_mtDisp2Val4_execute, result =>
display2_mtDisp2Val4_output, notify =>
display2_mtDisp2Val4_notify);
 display2_mtDisp2Val5: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display2_mtDisp2Val5_input_a, in_b =>
display2_mtDisp2Val5_input_b, execute =>
display2_mtDisp2Val5_execute, result =>
display2_mtDisp2Val5_output, notify =>
display2_mtDisp2Val5_notify);
 display2_mtDisp2Val6: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display2_mtDisp2Val6_input_a, in_b =>
display2_mtDisp2Val6_input_b, execute =>
display2_mtDisp2Val6_execute, result =>
display2_mtDisp2Val6_output, notify =>
display2_mtDisp2Val6_notify);
 display2_mtDisp2Val7: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display2_mtDisp2Val7_input_a, in_b =>
display2_mtDisp2Val7_input_b, execute =>
display2_mtDisp2Val7_execute, result =>
display2_mtDisp2Val7_output, notify =>
display2_mtDisp2Val7_notify);
 display2_mtDisp2Val8: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display2_mtDisp2Val8_input_a, in_b =>
display2_mtDisp2Val8_input_b, execute =>
```

```
display2_mtDisp2Val8_execute, result =>
display2_mtDisp2Val8_output, notify =>
display2_mtDisp2Val8_notify);
 display2_mtDisp2Val9: NOP_method GENERIC MAP(N_bits => 7,
dataType => 1, operation => 0) PORT MAP(in_a =>
display2_mtDisp2Val9_input_a, in_b =>
display2_mtDisp2Val9_input_b, execute =>
display2_mtDisp2Val9_execute, result =>
display2_mtDisp2Val9_output, notify =>
display2_mtDisp2Val9_notify);
---------- PON ELEMENTS CONNECTIONS -------------
 ---------- attributes inputs -------------
 contador_atContClock_new_value(0) <= contador_mtZera_output;
 contador_atContClock_set(0) <= contador_mtZera_notify;
 contador_atContClock_new_value(1) <=
contador_mtIncrementaCont_output;
 contador_atContClock_set(1) <=
contador_mtIncrementaCont_notify;
 contador_atUnidade_new_value(0) <= contador_mtIncUnid_output;
 contador_atUnidade_set(0) <= contador_mtIncUnid_notify;
 contador_atUnidade_new_value(1) <=
contador_mtZeraUnidade_output;
 contador_atUnidade_set(1) <= contador_mtZeraUnidade_notify;
 contador_atDezena_new_value(0) <= contador_mtIncDezena_output;
 contador_atDezena_set(0) <= contador_mtIncDezena_notify;
 contador_atDezena_new_value(1) <=
contador_mtZeraDezena_output;
 contador_atDezena_set(1) <= contador_mtZeraDezena_notify;
 contador_atCentena_new_value(0) <=
contador_mtIncCentena_output;
 contador_atCentena_set(0) <= contador_mtIncCentena_notify;
 contador_atCentena_new_value(1) <=
contador_mtZeraCentena_output;
 contador_atCentena_set(1) <= contador_mtZeraCentena_notify;
 contador_atLigado_new_value(0)(0) <= in_contador_atLigado;
 contador_atLigado_set(0) <= in_set_contador_atLigado;
 display0_atDisp0_new_value(0) <= display0_mtDisp0Val0_output;
 display0_atDisp0_set(0) <= display0_mtDisp0Val0_notify;
 display0_atDisp0_new_value(1) <= display0_mtDisp0Val1_output;
 display0_atDisp0_set(1) <= display0_mtDisp0Val1_notify;
 display0_atDisp0_new_value(2) <= display0_mtDisp0Val2_output;
 display0_atDisp0_set(2) <= display0_mtDisp0Val2_notify;
 display0_atDisp0_new_value(3) <= display0_mtDisp0Val3_output;
 display0_atDisp0_set(3) <= display0_mtDisp0Val3_notify;
 display0_atDisp0_new_value(4) <= display0_mtDisp0Val4_output;
 display0_atDisp0_set(4) <= display0_mtDisp0Val4_notify;
 display0_atDisp0_new_value(5) <= display0_mtDisp0Val5_output;
 display0_atDisp0_set(5) <= display0_mtDisp0Val5_notify;
 display0_atDisp0_new_value(6) <= display0_mtDisp0Val6_output;
 display0_atDisp0_set(6) <= display0_mtDisp0Val6_notify;
 display0_atDisp0_new_value(7) <= display0_mtDisp0Val7_output;
```

```
display0_atDisp0_set(7) <= display0_mtDisp0Val7_notify;
display0_atDisp0_new_value(8) <= display0_mtDisp0Val8_output;
display0_atDisp0_set(8) <= display0_mtDisp0Val8_notify;
display0_atDisp0_new_value(9) <= display0_mtDisp0Val9_output;
display0_atDisp0_set(9) <= display0_mtDisp0Val9_notify;
display1_atDisp1_new_value(0) <= display1_mtDisp1Val0_output;
display1_atDisp1_set(0) <= display1_mtDisp1Val0_notify;
display1_atDisp1_new_value(1) <= display1_mtDisp1Val1_output;
display1_atDisp1_set(1) <= display1_mtDisp1Val1_notify;
display1_atDisp1_new_value(2) <= display1_mtDisp1Val2_output;
display1_atDisp1_set(2) <= display1_mtDisp1Val2_notify;
display1_atDisp1_new_value(3) <= display1_mtDisp1Val3_output;
display1_atDisp1_set(3) <= display1_mtDisp1Val3_notify;
display1_atDisp1_new_value(4) <= display1_mtDisp1Val4_output;
display1_atDisp1_set(4) <= display1_mtDisp1Val4_notify;
display1_atDisp1_new_value(5) <= display1_mtDisp1Val5_output;
display1_atDisp1_set(5) <= display1_mtDisp1Val5_notify;
display1_atDisp1_new_value(6) <= display1_mtDisp1Val6_output;
display1_atDisp1_set(6) <= display1_mtDisp1Val6_notify;
display1_atDisp1_new_value(7) <= display1_mtDisp1Val7_output;
display1_atDisp1_set(7) <= display1_mtDisp1Val7_notify;
display1_atDisp1_new_value(8) <= display1_mtDisp1Val8_output;
display1_atDisp1_set(8) <= display1_mtDisp1Val8_notify;
display1_atDisp1_new_value(9) <= display1_mtDisp1Val9_output;
display1_atDisp1_set(9) <= display1_mtDisp1Val9_notify;
display2_atDisp2_new_value(0) <= display2_mtDisp2Val0_output;
display2_atDisp2_set(0) <= display2_mtDisp2Val0_notify;
display2_atDisp2_new_value(1) <= display2_mtDisp2Val1_output;
display2_atDisp2_set(1) <= display2_mtDisp2Val1_notify;
display2_atDisp2_new_value(2) <= display2_mtDisp2Val2_output;
display2_atDisp2_set(2) <= display2_mtDisp2Val2_notify;
display2_atDisp2_new_value(3) <= display2_mtDisp2Val3_output;
display2_atDisp2_set(3) <= display2_mtDisp2Val3_notify;
display2_atDisp2_new_value(4) <= display2_mtDisp2Val4_output;
display2_atDisp2_set(4) <= display2_mtDisp2Val4_notify;
display2_atDisp2_new_value(5) <= display2_mtDisp2Val5_output;
display2_atDisp2_set(5) <= display2_mtDisp2Val5_notify;
display2_atDisp2_new_value(6) <= display2_mtDisp2Val6_output;
display2_atDisp2_set(6) <= display2_mtDisp2Val6_notify;
display2_atDisp2_new_value(7) <= display2_mtDisp2Val7_output;
display2_atDisp2_set(7) <= display2_mtDisp2Val7_notify;
display2_atDisp2_new_value(8) <= display2_mtDisp2Val8_output;
display2_atDisp2_set(8) <= display2_mtDisp2Val8_notify;
display2_atDisp2_new_value(9) <= display2_mtDisp2Val9_output;
display2_atDisp2_set(9) <= display2_mtDisp2Val9_notify;

---------- attributes outputs -------------
out_display0_atDisp0 <= display0_atDisp0_value;
out_display1_atDisp1 <= display1_atDisp1_value;
out_display2_atDisp2 <= display2_atDisp2_value;
```

```
---------- premises inputs -------------
prCentenaMaior_imput_a <= contador_atCentena_value;
prContClockMaior_imput_a <= contador_atContClock_value;
prDezenaMaior_imput_a <= contador_atDezena_value;
prDisp0Val0_imput_a <= contador_atUnidade_value;
prDisp0Val1_imput_a <= contador_atUnidade_value;
prDisp0Val2_imput_a <= contador_atUnidade_value;
prDisp0Val3_imput_a <= contador_atUnidade_value;
prDisp0Val4_imput_a <= contador_atUnidade_value;
prDisp0Val5_imput_a <= contador_atUnidade_value;
prDisp0Val6_imput_a <= contador_atUnidade_value;
prDisp0Val7_imput_a <= contador_atUnidade_value;
prDisp0Val8_imput_a <= contador_atUnidade_value;
prDisp0Val9_imput_a <= contador_atUnidade_value;
prDisp1Val0_imput_a <= contador_atDezena_value;
prDisp1Val1_imput_a <= contador_atDezena_value;
prDisp1Val2_imput_a <= contador_atDezena_value;
prDisp1Val3_imput_a <= contador_atDezena_value;
prDisp1Val4_imput_a <= contador_atDezena_value;
prDisp1Val5_imput_a <= contador_atDezena_value;
prDisp1Val6_imput_a <= contador_atDezena_value;
prDisp1Val7_imput_a <= contador_atDezena_value;
prDisp1Val8_imput_a <= contador_atDezena_value;
prDisp1Val9_imput_a <= contador_atDezena_value;
prDisp2Val0_imput_a <= contador_atCentena_value;
prDisp2Val1_imput_a <= contador_atCentena_value;
prDisp2Val2_imput_a <= contador_atCentena_value;
prDisp2Val3_imput_a <= contador_atCentena_value;
prDisp2Val4_imput_a <= contador_atCentena_value;
prDisp2Val5_imput_a <= contador_atCentena_value;
prDisp2Val6_imput_a <= contador_atCentena_value;
prDisp2Val7_imput_a <= contador_atCentena_value;
prDisp2Val8_imput_a <= contador_atCentena_value;
prDisp2Val9_imput_a <= contador_atCentena_value;
prOn_imput_a <= contador_atLigado_value;
prUnidadeMaior_imput_a <= contador_atUnidade_value;

prCentenaMaior_imput_b <= "01001";
prContClockMaior_imput_b <= "01011111010111100001000000";
prDezenaMaior_imput_b <= "01001";
prDisp0Val0_imput_b <= "00000";
prDisp0Val1_imput_b <= "00001";
prDisp0Val2_imput_b <= "00010";
prDisp0Val3_imput_b <= "00011";
prDisp0Val4_imput_b <= "00100";
prDisp0Val5_imput_b <= "00101";
prDisp0Val6_imput_b <= "00110";
prDisp0Val7_imput_b <= "00111";
prDisp0Val8_imput_b <= "01000";
prDisp0Val9_imput_b <= "01001";
prDisp1Val0_imput_b <= "00000";
```

```
prDisp1Val1_imput_b <= "00001";
prDisp1Val2_imput_b <= "00010";
prDisp1Val3_imput_b <= "00011";
prDisp1Val4_imput_b <= "00100";
prDisp1Val5_imput_b <= "00101";
prDisp1Val6_imput_b <= "00110";
prDisp1Val7_imput_b <= "00111";
prDisp1Val8_imput_b <= "01000";
prDisp1Val9_imput_b <= "01001";
prDisp2Val0_imput_b <= "00000";
prDisp2Val1_imput_b <= "00001";
prDisp2Val2_imput_b <= "00010";
prDisp2Val3_imput_b <= "00011";
prDisp2Val4_imput_b <= "00100";
prDisp2Val5_imput_b <= "00101";
prDisp2Val6_imput_b <= "00110";
prDisp2Val7_imput_b <= "00111";
prDisp2Val8_imput_b <= "01000";
prDisp2Val9_imput_b <= "01001";
prOn_imput_b <= "1";
prUnidadeMaior_imput_b <= "01001";

---------- Methods Imputs -------------
contador_mtIncrementaCont_execute <= (prOn_result);
display0_mtDisp0Val0_execute <= (prDisp0Val0_result);
display0_mtDisp0Val1_execute <= (prDisp0Val1_result);
display0_mtDisp0Val2_execute <= (prDisp0Val2_result);
display0_mtDisp0Val3_execute <= (prDisp0Val3_result);
display0_mtDisp0Val4_execute <= (prDisp0Val4_result);
display0_mtDisp0Val5_execute <= (prDisp0Val5_result);
display0_mtDisp0Val6_execute <= (prDisp0Val6_result);
display0_mtDisp0Val7_execute <= (prDisp0Val7_result);
display0_mtDisp0Val8_execute <= (prDisp0Val8_result);
display0_mtDisp0Val9_execute <= (prDisp0Val9_result);
display1_mtDisp1Val0_execute <= (prDisp1Val0_result);
display1_mtDisp1Val1_execute <= (prDisp1Val1_result);
display1_mtDisp1Val2_execute <= (prDisp1Val2_result);
display1_mtDisp1Val3_execute <= (prDisp1Val3_result);
display1_mtDisp1Val4_execute <= (prDisp1Val4_result);
display1_mtDisp1Val5_execute <= (prDisp1Val5_result);
display1_mtDisp1Val6_execute <= (prDisp1Val6_result);
display1_mtDisp1Val7_execute <= (prDisp1Val7_result);
display1_mtDisp1Val8_execute <= (prDisp1Val8_result);
display1_mtDisp1Val9_execute <= (prDisp1Val9_result);
display2_mtDisp2Val0_execute <= (prDisp2Val0_result);
display2_mtDisp2Val1_execute <= (prDisp2Val1_result);
display2_mtDisp2Val2_execute <= (prDisp2Val2_result);
display2_mtDisp2Val3_execute <= (prDisp2Val3_result);
display2_mtDisp2Val4_execute <= (prDisp2Val4_result);
display2_mtDisp2Val5_execute <= (prDisp2Val5_result);
display2_mtDisp2Val6_execute <= (prDisp2Val6_result);
```

```
 display2_mtDisp2Val7_execute <= (prDisp2Val7_result);
 display2_mtDisp2Val8_execute <= (prDisp2Val8_result);
 display2_mtDisp2Val9_execute <= (prDisp2Val9_result);
 contador_mtIncCentena_execute <= (prDezenaMaior_result);
 contador_mtIncDezena_execute <= (prUnidadeMaior_result);
 contador_mtIncUnid_execute <= (prContClockMaior_result);
 contador_mtZeraCentena_execute <= (prCentenaMaior_result);
 contador_mtZera_execute <= (prContClockMaior_result);
 contador_mtZeraDezena_execute <= (prDezenaMaior_result);
 contador_mtZeraUnidade_execute <= (prUnidadeMaior_result);
 contador_mtZera_input_a <= "00000000000000000000000000000";
 contador_mtZera_input_b <= "00000000000000000000000000000";
 contador_mtIncrementaCont_input_a <=
contador_atContClock_value;
 contador_mtIncrementaCont_input_b <=
"00000000000000000000000000001";
 contador_mtIncUnid_input_a <= contador_atUnidade_value;
 contador_mtIncUnid_input_b <= "00001";
 contador_mtIncDezena_input_a <= contador_atDezena_value;
 contador_mtIncDezena_input_b <= "00001";
 contador_mtIncCentena_input_a <= contador_atCentena_value;
 contador_mtIncCentena_input_b <= "00001";
 contador_mtZeraUnidade_input_a <= "00000";
 contador_mtZeraUnidade_input_b <= "00000";
 contador_mtZeraDezena_input_a <= "00000";
 contador_mtZeraDezena_input_b <= "00000";
 contador_mtZeraCentena_input_a <= "00000";
 contador_mtZeraCentena_input_b <= "00000";
 display0_mtDisp0Val0_input_a <= "1000000";
 display0_mtDisp0Val0_input_b <= "0000000";
 display0_mtDisp0Val1_input_a <= "1111001";
 display0_mtDisp0Val1_input_b <= "0000000";
 display0_mtDisp0Val2_input_a <= "0100100";
 display0_mtDisp0Val2_input_b <= "0000000";
 display0_mtDisp0Val3_input_a <= "0110000";
 display0_mtDisp0Val3_input_b <= "0000000";
 display0_mtDisp0Val4_input_a <= "0011001";
 display0_mtDisp0Val4_input_b <= "0000000";
 display0_mtDisp0Val5_input_a <= "0010010";
 display0_mtDisp0Val5_input_b <= "0000000";
 display0_mtDisp0Val6_input_a <= "0000010";
 display0_mtDisp0Val6_input_b <= "0000000";
 display0_mtDisp0Val7_input_a <= "1111000";
 display0_mtDisp0Val7_input_b <= "0000000";
 display0_mtDisp0Val8_input_a <= "0000000";
 display0_mtDisp0Val8_input_b <= "0000000";
 display0_mtDisp0Val9_input_a <= "0010000";
 display0_mtDisp0Val9_input_b <= "0000000";
 display1_mtDisp1Val0_input_a <= "1000000";
 display1_mtDisp1Val0_input_b <= "0000000";
 display1_mtDisp1Val1_input_a <= "1111001";
```

```
display1_mtDisp1Val1_input_b <= "0000000";
display1_mtDisp1Val2_input_a <= "0100100";
display1_mtDisp1Val2_input_b <= "0000000";
display1_mtDisp1Val3_input_a <= "0110000";
display1_mtDisp1Val3_input_b <= "0000000";
display1_mtDisp1Val4_input_a <= "0011001";
display1_mtDisp1Val4_input_b <= "0000000";
display1_mtDisp1Val5_input_a <= "0010010";
display1_mtDisp1Val5_input_b <= "0000000";
display1_mtDisp1Val6_input_a <= "0000010";
display1_mtDisp1Val6_input_b <= "0000000";
display1_mtDisp1Val7_input_a <= "1111000";
display1_mtDisp1Val7_input_b <= "0000000";
display1_mtDisp1Val8_input_a <= "0000000";
display1_mtDisp1Val8_input_b <= "0000000";
display1_mtDisp1Val9_input_a <= "0010000";
display1_mtDisp1Val9_input_b <= "0000000";
display2_mtDisp2Val0_input_a <= "1000000";
display2_mtDisp2Val0_input_b <= "0000000";
display2_mtDisp2Val1_input_a <= "1111001";
display2_mtDisp2Val1_input_b <= "0000000";
display2_mtDisp2Val2_input_a <= "0100100";
display2_mtDisp2Val2_input_b <= "0000000";
display2_mtDisp2Val3_input_a <= "0110000";
display2_mtDisp2Val3_input_b <= "0000000";
display2_mtDisp2Val4_input_a <= "0011001";
display2_mtDisp2Val4_input_b <= "0000000";
display2_mtDisp2Val5_input_a <= "0010010";
display2_mtDisp2Val5_input_b <= "0000000";
display2_mtDisp2Val6_input_a <= "0000010";
display2_mtDisp2Val6_input_b <= "0000000";
display2_mtDisp2Val7_input_a <= "1111000";
display2_mtDisp2Val7_input_b <= "0000000";
display2_mtDisp2Val8_input_a <= "0000000";
display2_mtDisp2Val8_input_b <= "0000000";
display2_mtDisp2Val9_input_a <= "0010000";
display2_mtDisp2Val9_input_b <= "0000000";
END ContadorDecimal_arq;
```

Apêndice L – calculaMedia.pon

Código fonte em LingPON-HD 1.0 para o experimento do cálculo de média.

```
fbe fbeCalculaMedia
 attributes
  integer atDataIn 0 // entrada de dados
  integer atTemp 0 // armazena temporariamente o resultado
  integer atRes 0 // saída que apresenta a média
  integer atAddr 0 // endereço para a leitura dos dados de
entrada
  integer atStep 0 // controla as etapas de funcionamento do
circuito
  boolean atCE false // sinal de leitura dos dados de entrada
  boolean atResVal false // saída que indica que o resultado é
válido
  boolean atReset false // entrada de reset
 end_attributes
 methods
  method mtResetRes(atRes = 0)
  method mtResetStep(atStep = 0)
  method mtIncStep(atStep = atStep + 1)
  method mtResVal_0(atResVal = false)
  method mtResVal_1(atResVal = true)
  method mtAddr_0(atAddr = 0)
  method mtAddrInc(atAddr = atAddr + 1)
  method mtCE_0(atCE = false)
  method mtCE_1(atCE = true)
  method mtAtualRes(atRes = atTemp / 10) // atualiza Resultado
  method mtTempDataIn(atTemp = atDataIn)
  method mtAcumulaTemp(atTemp = atTemp + atDataIn)
 end_methods
end_fbe

inst
 fbeCalculaMedia CM
end_inst

strategy
 no_one
end_strategy

// Reset
rule rlReset
 condition
  subcondition SReset
   premise prReset CM.atReset == true
  end_subcondition
 end_condition
 action
  instigation inResetRes CM.mtResetRes();
```

```
 instigation inResetAddr CM.mtAddr_0();
 instigation inResetStep CM.mtResetStep();
 instigation inResetResetCE CM.mtCE_0();
 instigation inResetResVal CM.mtResVal_0();
end_action
end_rule

// Etapa 0
rule rlEtapa0
condition
 subcondition SEtapa0
  premise prEtapa0 CM.atStep == 0
 end_subcondition
end_condition
action
 instigation inEtapa0Addr_0 CM.mtAddr_0();
 instigation inEtapa0CE_1 CM.mtCE_1();
 instigation inEtapa0AtualRes CM.mtAtualRes();
 instigation inEtapa0ResVal_1 CM.mtResVal_1();
 instigation inEtapa0IncStep CM.mtIncStep();
end_action
end_rule

// Etapa 1
rule rlEtapa1
condition
 subcondition SEtapa1
  premise prEtapa1 CM.atStep == 1
 end_subcondition
end_condition
action
 instigation inEtapa1CE_0 CM.mtCE_0();
 instigation inEtapa1TempDataIn CM.mtTempDataIn();
 instigation inEtapa1ResVal_0 CM.mtResVal_0();
 instigation inEtapa1IncStep CM.mtIncStep();
end_action
end_rule

// Etapa 2
rule rlEtapa2
condition
 subcondition SEtapa2
  premise prEtapa2 CM.atStep == 2
 end_subcondition
end_condition
action
 instigation inEtapa2AddrInc CM.mtAddrInc();
 instigation inEtapa2CE_1 CM.mtCE_1();
 instigation inEtapa2IncStep CM.mtIncStep();
end_action
end_rule
```

```
// Etapa 3
rule rlEtapa3
 condition
  subcondition SEtapa3
   premise prEtapa3 CM.atStep == 3
  end_subcondition
 end_condition
 action
  instigation inEtapa3CE_0 CM.mtCE_0();
  instigation inEtapa3AcumulaTemp CM.mtAcumulaTemp();
  instigation inEtapa3IncStep CM.mtIncStep();
 end_action
end_rule

// Etapa 4
rule rlEtapa4
 condition
  subcondition SEtapa4
   premise prEtapa4 CM.atStep == 4
  end_subcondition
 end_condition
 action
  instigation inEtapa4AddrInc CM.mtAddrInc();
  instigation inEtapa4CE_1 CM.mtCE_1();
  instigation inEtapa4IncStep CM.mtIncStep();
 end_action
end_rule

// Etapa 5
rule rlEtapa5
 condition
  subcondition SEtapa5
   premise prEtapa5 CM.atStep == 5
  end_subcondition
 end_condition
 action
  instigation inEtapa5CE_0 CM.mtCE_0();
  instigation inEtapa5AcumulaTemp CM.mtAcumulaTemp();
  instigation inEtapa5IncStep CM.mtIncStep();
 end_action
end_rule

// Etapa 6
rule rlEtapa6
 condition
  subcondition SEtapa6
   premise prEtapa6 CM.atStep == 6
  end_subcondition
 end_condition
 action
```

```
 instigation inEtapa6AddrInc CM.mtAddrInc();
 instigation inEtapa6CE_1 CM.mtCE_1();
 instigation inEtapa6IncStep CM.mtIncStep();
end_action
end_rule

// Etapa 7
rule rlEtapa7
 condition
 subcondition SEtapa7
  premise prEtapa7 CM.atStep == 7
 end_subcondition
end_condition
action
 instigation inEtapa7CE_0 CM.mtCE_0();
 instigation inEtapa7AcumulaTemp CM.mtAcumulaTemp();
 instigation inEtapa7IncStep CM.mtIncStep();
end_action
end_rule

// Etapa 8
rule rlEtapa8
 condition
 subcondition SEtapa8
  premise prEtapa8 CM.atStep == 8
 end_subcondition
end_condition
action
 instigation inEtapa8AddrInc CM.mtAddrInc();
 instigation inEtapa8CE_1 CM.mtCE_1();
 instigation inEtapa8IncStep CM.mtIncStep();
end_action
end_rule

// Etapa 9
rule rlEtapa9
 condition
 subcondition SEtapa9
  premise prEtapa9 CM.atStep == 9
 end_subcondition
end_condition
action
 instigation inEtapa9CE_0 CM.mtCE_0();
 instigation inEtapa9AcumulaTemp CM.mtAcumulaTemp();
 instigation inEtapa9IncStep CM.mtIncStep();
end_action
end_rule

// Etapa 10
rule rlEtapa10
 condition
```

```
 subcondition SEtapa10
  premise prEtapa10 CM.atStep == 10
 end_subcondition
end_condition
action
 instigation inEtapa10AddrInc CM.mtAddrInc();
 instigation inEtapa10CE_1 CM.mtCE_1();
 instigation inEtapa10IncStep CM.mtIncStep();
end_action
end_rule

// Etapa 11
rule rlEtapa11
condition
 subcondition SEtapa11
  premise prEtapa11 CM.atStep == 11
 end_subcondition
end_condition
action
 instigation inEtapa11CE_0 CM.mtCE_0();
 instigation inEtapa11AcumulaTemp CM.mtAcumulaTemp();
 instigation inEtapa11IncStep CM.mtIncStep();
end_action
end_rule

// Etapa 12
rule rlEtapa12
condition
 subcondition SEtapa12
  premise prEtapa12 CM.atStep == 12
 end_subcondition
end_condition
action
 instigation inEtapa12AddrInc CM.mtAddrInc();
 instigation inEtapa12CE_1 CM.mtCE_1();
 instigation inEtapa12IncStep CM.mtIncStep();
end_action
end_rule

// Etapa 13
rule rlEtapa13
condition
 subcondition SEtapa13
  premise prEtapa13 CM.atStep == 13
 end_subcondition
end_condition
action
 instigation inEtapa13CE_0 CM.mtCE_0();
 instigation inEtapa13AcumulaTemp CM.mtAcumulaTemp();
 instigation inEtapa13IncStep CM.mtIncStep();
end_action
```

```
end_rule

// Etapa 14
rule rlEtapa14
 condition
  subcondition SEtapa14
   premise prEtapa14 CM.atStep == 14
  end_subcondition
 end_condition
 action
  instigation inEtapa14AddrInc CM.mtAddrInc();
  instigation inEtapa14CE_1 CM.mtCE_1();
  instigation inEtapa14IncStep CM.mtIncStep();
 end_action
end_rule

// Etapa 15
rule rlEtapa15
 condition
  subcondition SEtapa15
   premise prEtapa15 CM.atStep == 15
  end_subcondition
 end_condition
 action
  instigation inEtapa15CE_0 CM.mtCE_0();
  instigation inEtapa15AcumulaTemp CM.mtAcumulaTemp();
  instigation inEtapa15IncStep CM.mtIncStep();
 end_action
end_rule

// Etapa 16
rule rlEtapa16
 condition
  subcondition SEtapa16
   premise prEtapa16 CM.atStep == 16
  end_subcondition
 end_condition
 action
  instigation inEtapa16AddrInc CM.mtAddrInc();
  instigation inEtapa16CE_1 CM.mtCE_1();
  instigation inEtapa16IncStep CM.mtIncStep();
 end_action
end_rule

// Etapa 17
rule rlEtapa17
 condition
  subcondition SEtapa17
   premise prEtapa17 CM.atStep == 17
  end_subcondition
 end_condition
```

```
 action
  instigation inEtapa17CE_0 CM.mtCE_0();
  instigation inEtapa17AcumulaTemp CM.mtAcumulaTemp();
  instigation inEtapa17IncStep CM.mtIncStep();
 end_action
end_rule

// Etapa 18
rule rlEtapa18
 condition
  subcondition SEtapa18
   premise prEtapa18 CM.atStep == 18
  end_subcondition
 end_condition
 action
  instigation inEtapa18AddrInc CM.mtAddrInc();
  instigation inEtapa18CE_1 CM.mtCE_1();
  instigation inEtapa18IncStep CM.mtIncStep();
 end_action
end_rule

// Etapa 19
rule rlEtapa19
 condition
  subcondition SEtapa19
   premise prEtapa19 CM.atStep == 19
  end_subcondition
 end_condition
 action
  instigation inEtapa19CE_0 CM.mtCE_0();
  instigation inEtapa19AcumulaTemp CM.mtAcumulaTemp();
  instigation inEtapa19ResetStep CM.mtResetStep();
 end_action
end_rule

main {
 entity calculaMedia

 in CM.atDataIn
 out CM.atRes
 out CM.atAddr
 out CM.atCE
 out CM.atResVal
 in CM.atReset

 NBits CM.atDataIn 16
 NBits CM.atTemp 16
 NBits CM.atRes 16
 NBits CM.atAddr 4
 NBits CM.atStep 6
 }
```

Apêndice M – calculaMedia.vhd

Código fonte em VHDL para o experimento do cálculo de média gerado automaticamente pelo Compilador PON-HD 1.0.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.data_type_pkg.all;

ENTITY calculaMedia IS
PORT(
 in_CM_atDataIn :IN STD_LOGIC_VECTOR(16-1 downto 0);
 in_set_CM_atDataIn :IN STD_LOGIC;
 out_CM_atRes :OUT STD_LOGIC_VECTOR(16-1 downto 0);
 out_CM_atAddr :OUT STD_LOGIC_VECTOR(4-1 downto 0);
 out_CM_atCE :OUT STD_LOGIC;
 out_CM_atResVal :OUT STD_LOGIC;
 in_CM_atReset :IN STD_LOGIC;
 in_set_CM_atReset :IN STD_LOGIC;
 clock :IN STD_LOGIC
 );
END calculaMedia;

ARCHITECTURE calculaMedia_arq OF calculaMedia IS
----------------- COMPONENTS ---------------------
COMPONENT NOP_attribute
 GENERIC (
 N_bits: INTEGER;
 N_new_values: INTEGER;
 initial_value: STD_LOGIC_VECTOR
 );
 PORT(
 att_clock :IN STD_LOGIC;
 att_new_value :IN data(0 TO N_new_values-1)(N_bits-1 downto
0);
 att_set_value :IN STD_LOGIC_VECTOR(N_new_values-1 downto 0);
 att_value :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0)
 );
END COMPONENT;
COMPONENT NOP_premise
 GENERIC (
 N_bits: INTEGER:=1;
 dataType: INTEGER:=0;
 operation: INTEGER:=1 --EQUAL 1; DIFFERENT 2; LESS_THAN 3;
GREATER_THAN 4; LESS_EQUAL 5; GREATER_EQUAL 6;
 );
 PORT(
 att_value_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 att_value_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 pre_result :OUT STD_LOGIC
 );
```

```
END COMPONENT;
COMPONENT NOP_method
 GENERIC (
  N_bits: INTEGER:=1;
  dataType: INTEGER:=1;
 operation: INTEGER:=1
 );
 PORT(
 in_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 in_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 execute :IN STD_LOGIC;
 result :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0);
 notify :OUT STD_LOGIC
 );
END COMPONENT;
----------------- SIGNALS --------------------
 SIGNAL CM_atDataIn_new_value: data(0 TO 1-1)(16-1 downto 0);
 SIGNAL CM_atTemp_new_value: data(0 TO 2-1)(16-1 downto 0);
 SIGNAL CM_atRes_new_value: data(0 TO 2-1)(16-1 downto 0);
 SIGNAL CM_atAddr_new_value: data(0 TO 2-1)(4-1 downto 0);
 SIGNAL CM_atStep_new_value: data(0 TO 2-1)(6-1 downto 0);
 SIGNAL CM_atCE_new_value: data(0 TO 2-1)(0 downto 0);
 SIGNAL CM_atResVal_new_value: data(0 TO 2-1)(0 downto 0);
 SIGNAL CM_atReset_new_value: data(0 TO 1-1)(0 downto 0);

 SIGNAL CM_atDataIn_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL CM_atTemp_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL CM_atRes_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL CM_atAddr_value: STD_LOGIC_VECTOR(4-1 downto 0);
 SIGNAL CM_atStep_value: STD_LOGIC_VECTOR(6-1 downto 0);
 SIGNAL CM_atCE_value: STD_LOGIC_VECTOR(0 downto 0);
 SIGNAL CM_atResVal_value: STD_LOGIC_VECTOR(0 downto 0);
 SIGNAL CM_atReset_value: STD_LOGIC_VECTOR(0 downto 0);

 SIGNAL CM_atDataIn_set: STD_LOGIC_VECTOR(1-1 downto 0);
 SIGNAL CM_atTemp_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL CM_atRes_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL CM_atAddr_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL CM_atStep_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL CM_atCE_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL CM_atResVal_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL CM_atReset_set: STD_LOGIC_VECTOR(1-1 downto 0);

 SIGNAL prEtapa0_result: STD_LOGIC;
 SIGNAL prEtapa1_result: STD_LOGIC;
 SIGNAL prEtapa10_result: STD_LOGIC;
 SIGNAL prEtapa11_result: STD_LOGIC;
 SIGNAL prEtapa12_result: STD_LOGIC;
 SIGNAL prEtapa13_result: STD_LOGIC;
 SIGNAL prEtapa14_result: STD_LOGIC;
 SIGNAL prEtapa15_result: STD_LOGIC;
```

```
SIGNAL prEtapa16_result: STD_LOGIC;
SIGNAL prEtapa17_result: STD_LOGIC;
SIGNAL prEtapa18_result: STD_LOGIC;
SIGNAL prEtapa19_result: STD_LOGIC;
SIGNAL prEtapa2_result: STD_LOGIC;
SIGNAL prEtapa3_result: STD_LOGIC;
SIGNAL prEtapa4_result: STD_LOGIC;
SIGNAL prEtapa5_result: STD_LOGIC;
SIGNAL prEtapa6_result: STD_LOGIC;
SIGNAL prEtapa7_result: STD_LOGIC;
SIGNAL prEtapa8_result: STD_LOGIC;
SIGNAL prEtapa9_result: STD_LOGIC;
SIGNAL prReset_result: STD_LOGIC;

SIGNAL prEtapa0_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa1_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa10_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa11_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa12_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa13_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa14_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa15_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa16_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa17_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa18_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa19_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa2_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa3_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa4_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa5_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa6_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa7_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa8_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa9_imput_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prReset_imput_a: STD_LOGIC_VECTOR(0 downto 0);

SIGNAL prEtapa0_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa1_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa10_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa11_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa12_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa13_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa14_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa15_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa16_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa17_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa18_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa19_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa2_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa3_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa4_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
```

```
SIGNAL prEtapa5_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa6_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa7_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa8_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prEtapa9_imput_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL prReset_imput_b: STD_LOGIC_VECTOR(0 downto 0);

SIGNAL CM_mtResetRes_input_a: STD_LOGIC_VECTOR(16-1 downto 0);
SIGNAL CM_mtResetStep_input_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL CM_mtIncStep_input_a: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL CM_mtResVal_0_input_a: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtResVal_1_input_a: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtAddr_0_input_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL CM_mtAddrInc_input_a: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL CM_mtCE_0_input_a: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtCE_1_input_a: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtAtualRes_input_a: STD_LOGIC_VECTOR(16-1 downto 0);
SIGNAL CM_mtTempDataIn_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
SIGNAL CM_mtAcumulaTemp_input_a: STD_LOGIC_VECTOR(16-1 downto
0);

SIGNAL CM_mtResetRes_input_b: STD_LOGIC_VECTOR(16-1 downto 0);
SIGNAL CM_mtResetStep_input_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL CM_mtIncStep_input_b: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL CM_mtResVal_0_input_b: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtResVal_1_input_b: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtAddr_0_input_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL CM_mtAddrInc_input_b: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL CM_mtCE_0_input_b: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtCE_1_input_b: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtAtualRes_input_b: STD_LOGIC_VECTOR(16-1 downto 0);
SIGNAL CM_mtTempDataIn_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
SIGNAL CM_mtAcumulaTemp_input_b: STD_LOGIC_VECTOR(16-1 downto
0);

SIGNAL CM_mtResetRes_output: STD_LOGIC_VECTOR(16-1 downto 0);
SIGNAL CM_mtResetStep_output: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL CM_mtIncStep_output: STD_LOGIC_VECTOR(6-1 downto 0);
SIGNAL CM_mtResVal_0_output: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtResVal_1_output: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtAddr_0_output: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL CM_mtAddrInc_output: STD_LOGIC_VECTOR(4-1 downto 0);
SIGNAL CM_mtCE_0_output: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtCE_1_output: STD_LOGIC_VECTOR(0 downto 0);
SIGNAL CM_mtAtualRes_output: STD_LOGIC_VECTOR(16-1 downto 0);
SIGNAL CM_mtTempDataIn_output: STD_LOGIC_VECTOR(16-1 downto
0);
SIGNAL CM_mtAcumulaTemp_output: STD_LOGIC_VECTOR(16-1 downto
0);
```

```
SIGNAL CM_mtResetRes_execute: STD_LOGIC;
SIGNAL CM_mtResetStep_execute: STD_LOGIC;
SIGNAL CM_mtIncStep_execute: STD_LOGIC;
SIGNAL CM_mtResVal_0_execute: STD_LOGIC;
SIGNAL CM_mtResVal_1_execute: STD_LOGIC;
SIGNAL CM_mtAddr_0_execute: STD_LOGIC;
SIGNAL CM_mtAddrInc_execute: STD_LOGIC;
SIGNAL CM_mtCE_0_execute: STD_LOGIC;
SIGNAL CM_mtCE_1_execute: STD_LOGIC;
SIGNAL CM_mtAtualRes_execute: STD_LOGIC;
SIGNAL CM_mtTempDataIn_execute: STD_LOGIC;
SIGNAL CM_mtAcumulaTemp_execute: STD_LOGIC;

SIGNAL CM_mtResetRes_notify: STD_LOGIC;
SIGNAL CM_mtResetStep_notify: STD_LOGIC;
SIGNAL CM_mtIncStep_notify: STD_LOGIC;
SIGNAL CM_mtResVal_0_notify: STD_LOGIC;
SIGNAL CM_mtResVal_1_notify: STD_LOGIC;
SIGNAL CM_mtAddr_0_notify: STD_LOGIC;
SIGNAL CM_mtAddrInc_notify: STD_LOGIC;
SIGNAL CM_mtCE_0_notify: STD_LOGIC;
SIGNAL CM_mtCE_1_notify: STD_LOGIC;
SIGNAL CM_mtAtualRes_notify: STD_LOGIC;
SIGNAL CM_mtTempDataIn_notify: STD_LOGIC;
SIGNAL CM_mtAcumulaTemp_notify: STD_LOGIC;
-----------------------------------------------
BEGIN
---------- PON ELEMENTS INSTANCES -------------
CM_atDataIn: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 1, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
CM_atDataIn_new_value, att_set_value => CM_atDataIn_set,
att_value => CM_atDataIn_value);
CM_atTemp: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value => CM_atTemp_new_value,
att_set_value => CM_atTemp_set, att_value => CM_atTemp_value);
CM_atRes: NOP_attribute GENERIC MAP(N_bits => 16, N_new_values
=> 2, initial_value => "0000000000000000") PORT MAP(att_clock
=> clock, att_new_value => CM_atRes_new_value, att_set_value
=> CM_atRes_set, att_value => CM_atRes_value);
CM_atAddr: NOP_attribute GENERIC MAP(N_bits => 4, N_new_values
=> 2, initial_value => "0000") PORT MAP(att_clock => clock,
att_new_value => CM_atAddr_new_value, att_set_value =>
CM_atAddr_set, att_value => CM_atAddr_value);
CM_atStep: NOP_attribute GENERIC MAP(N_bits => 6, N_new_values
=> 2, initial_value => "000000") PORT MAP(att_clock => clock,
att_new_value => CM_atStep_new_value, att_set_value =>
CM_atStep_set, att_value => CM_atStep_value);
```

```
CM_atCE: NOP_attribute GENERIC MAP(N_bits => 1, N_new_values
=> 2, initial_value => "0") PORT MAP(att_clock => clock,
att_new_value => CM_atCE_new_value, att_set_value =>
CM_atCE_set, att_value => CM_atCE_value);
CM_atResVal: NOP_attribute GENERIC MAP(N_bits => 1,
N_new_values => 2, initial_value => "0") PORT MAP(att_clock =>
clock, att_new_value => CM_atResVal_new_value, att_set_value
=> CM_atResVal_set, att_value => CM_atResVal_value);
CM_atReset: NOP_attribute GENERIC MAP(N_bits => 1,
N_new_values => 1, initial_value => "0") PORT MAP(att_clock =>
clock, att_new_value => CM_atReset_new_value, att_set_value =>
CM_atReset_set, att_value => CM_atReset_value);

prEtapa0_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa0_imput_a, att_value_b => prEtapa0_imput_b, pre_result
=> prEtapa0_result);
prEtapa1_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa1_imput_a, att_value_b => prEtapa1_imput_b, pre_result
=> prEtapa1_result);
prEtapa10_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa10_imput_a, att_value_b => prEtapa10_imput_b,
pre_result => prEtapa10_result);
prEtapa11_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa11_imput_a, att_value_b => prEtapa11_imput_b,
pre_result => prEtapa11_result);
prEtapa12_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa12_imput_a, att_value_b => prEtapa12_imput_b,
pre_result => prEtapa12_result);
prEtapa13_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa13_imput_a, att_value_b => prEtapa13_imput_b,
pre_result => prEtapa13_result);
prEtapa14_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa14_imput_a, att_value_b => prEtapa14_imput_b,
pre_result => prEtapa14_result);
prEtapa15_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa15_imput_a, att_value_b => prEtapa15_imput_b,
pre_result => prEtapa15_result);
prEtapa16_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa16_imput_a, att_value_b => prEtapa16_imput_b,
pre_result => prEtapa16_result);
prEtapa17_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
```

```
prEtapa17_imput_a, att_value_b => prEtapa17_imput_b,
pre_result => prEtapa17_result);
 prEtapa18_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa18_imput_a, att_value_b => prEtapa18_imput_b,
pre_result => prEtapa18_result);
 prEtapa19_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa19_imput_a, att_value_b => prEtapa19_imput_b,
pre_result => prEtapa19_result);
 prEtapa2_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa2_imput_a, att_value_b => prEtapa2_imput_b, pre_result
=> prEtapa2_result);
 prEtapa3_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa3_imput_a, att_value_b => prEtapa3_imput_b, pre_result
=> prEtapa3_result);
 prEtapa4_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa4_imput_a, att_value_b => prEtapa4_imput_b, pre_result
=> prEtapa4_result);
 prEtapa5_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa5_imput_a, att_value_b => prEtapa5_imput_b, pre_result
=> prEtapa5_result);
 prEtapa6_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa6_imput_a, att_value_b => prEtapa6_imput_b, pre_result
=> prEtapa6_result);
 prEtapa7_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa7_imput_a, att_value_b => prEtapa7_imput_b, pre_result
=> prEtapa7_result);
 prEtapa8_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa8_imput_a, att_value_b => prEtapa8_imput_b, pre_result
=> prEtapa8_result);
 prEtapa9_EQUAL: NOP_premise GENERIC MAP(N_bits => 6, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prEtapa9_imput_a, att_value_b => prEtapa9_imput_b, pre_result
=> prEtapa9_result);
 prReset_EQUAL: NOP_premise GENERIC MAP(N_bits => 1, dataType
=> 0, operation => 1) PORT MAP(att_value_a => prReset_imput_a,
att_value_b => prReset_imput_b, pre_result => prReset_result);

 CM_mtResetRes: NOP_method GENERIC MAP(N_bits => 16, dataType
=> 1, operation => 0) PORT MAP(in_a => CM_mtResetRes_input_a,
in_b => CM_mtResetRes_input_b, execute =>
CM_mtResetRes_execute, result => CM_mtResetRes_output, notify
=> CM_mtResetRes_notify);
```

```
CM_mtResetStep: NOP_method GENERIC MAP(N_bits => 6, dataType
=> 1, operation => 0) PORT MAP(in_a => CM_mtResetStep_input_a,
in_b => CM_mtResetStep_input_b, execute =>
CM_mtResetStep_execute, result => CM_mtResetStep_output,
notify => CM_mtResetStep_notify);
CM_mtIncStep: NOP_method GENERIC MAP(N_bits => 6, dataType =>
1, operation => 1) PORT MAP(in_a => CM_mtIncStep_input_a, in_b
=> CM_mtIncStep_input_b, execute => CM_mtIncStep_execute,
result => CM_mtIncStep_output, notify => CM_mtIncStep_notify);
CM_mtResVal_0: NOP_method GENERIC MAP(N_bits => 1, dataType =>
1, operation => 0) PORT MAP(in_a => CM_mtResVal_0_input_a,
in_b => CM_mtResVal_0_input_b, execute =>
CM_mtResVal_0_execute, result => CM_mtResVal_0_output, notify
=> CM_mtResVal_0_notify);
CM_mtResVal_1: NOP_method GENERIC MAP(N_bits => 1, dataType =>
1, operation => 0) PORT MAP(in_a => CM_mtResVal_1_input_a,
in_b => CM_mtResVal_1_input_b, execute =>
CM_mtResVal_1_execute, result => CM_mtResVal_1_output, notify
=> CM_mtResVal_1_notify);
CM_mtAddr_0: NOP_method GENERIC MAP(N_bits => 4, dataType =>
1, operation => 0) PORT MAP(in_a => CM_mtAddr_0_input_a, in_b
=> CM_mtAddr_0_input_b, execute => CM_mtAddr_0_execute, result
=> CM_mtAddr_0_output, notify => CM_mtAddr_0_notify);
CM_mtAddrInc: NOP_method GENERIC MAP(N_bits => 4, dataType =>
1, operation => 1) PORT MAP(in_a => CM_mtAddrInc_input_a, in_b
=> CM_mtAddrInc_input_b, execute => CM_mtAddrInc_execute,
result => CM_mtAddrInc_output, notify => CM_mtAddrInc_notify);
CM_mtCE_0: NOP_method GENERIC MAP(N_bits => 1, dataType => 1,
operation => 0) PORT MAP(in_a => CM_mtCE_0_input_a, in_b =>
CM_mtCE_0_input_b, execute => CM_mtCE_0_execute, result =>
CM_mtCE_0_output, notify => CM_mtCE_0_notify);
CM_mtCE_1: NOP_method GENERIC MAP(N_bits => 1, dataType => 1,
operation => 0) PORT MAP(in_a => CM_mtCE_1_input_a, in_b =>
CM_mtCE_1_input_b, execute => CM_mtCE_1_execute, result =>
CM_mtCE_1_output, notify => CM_mtCE_1_notify);
CM_mtAtualRes: NOP_method GENERIC MAP(N_bits => 16, dataType
=> 1, operation => 4) PORT MAP(in_a => CM_mtAtualRes_input_a,
in_b => CM_mtAtualRes_input_b, execute =>
CM_mtAtualRes_execute, result => CM_mtAtualRes_output, notify
=> CM_mtAtualRes_notify);
CM_mtTempDataIn: NOP_method GENERIC MAP(N_bits => 16, dataType
=> 1, operation => 0) PORT MAP(in_a =>
CM_mtTempDataIn_input_a, in_b => CM_mtTempDataIn_input_b,
execute => CM_mtTempDataIn_execute, result =>
CM_mtTempDataIn_output, notify => CM_mtTempDataIn_notify);
CM_mtAcumulaTemp: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 1) PORT MAP(in_a =>
CM_mtAcumulaTemp_input_a, in_b => CM_mtAcumulaTemp_input_b,
execute => CM_mtAcumulaTemp_execute, result =>
CM_mtAcumulaTemp_output, notify => CM_mtAcumulaTemp_notify);
---------- PON ELEMENTS CONNECTIONS -------------
```

```
---------- attributes inputs -------------
CM_atDataIn_new_value(0) <= in_CM_atDataIn;
CM_atDataIn_set(0) <= in_set_CM_atDataIn;
CM_atTemp_new_value(0) <= CM_mtTempDataIn_output;
CM_atTemp_set(0) <= CM_mtTempDataIn_notify;
CM_atTemp_new_value(1) <= CM_mtAcumulaTemp_output;
CM_atTemp_set(1) <= CM_mtAcumulaTemp_notify;
CM_atRes_new_value(0) <= CM_mtResetRes_output;
CM_atRes_set(0) <= CM_mtResetRes_notify;
CM_atRes_new_value(1) <= CM_mtAtualRes_output;
CM_atRes_set(1) <= CM_mtAtualRes_notify;
CM_atAddr_new_value(0) <= CM_mtAddr_0_output;
CM_atAddr_set(0) <= CM_mtAddr_0_notify;
CM_atAddr_new_value(1) <= CM_mtAddrInc_output;
CM_atAddr_set(1) <= CM_mtAddrInc_notify;
CM_atStep_new_value(0) <= CM_mtResetStep_output;
CM_atStep_set(0) <= CM_mtResetStep_notify;
CM_atStep_new_value(1) <= CM_mtIncStep_output;
CM_atStep_set(1) <= CM_mtIncStep_notify;
CM_atCE_new_value(0) <= CM_mtCE_0_output;
CM_atCE_set(0) <= CM_mtCE_0_notify;
CM_atCE_new_value(1) <= CM_mtCE_1_output;
CM_atCE_set(1) <= CM_mtCE_1_notify;
CM_atResVal_new_value(0) <= CM_mtResVal_0_output;
CM_atResVal_set(0) <= CM_mtResVal_0_notify;
CM_atResVal_new_value(1) <= CM_mtResVal_1_output;
CM_atResVal_set(1) <= CM_mtResVal_1_notify;
CM_atReset_new_value(0)(0) <= in_CM_atReset;
CM_atReset_set(0) <= in_set_CM_atReset;

---------- attributes outputs -------------
out_CM_atRes <= CM_atRes_value;
out_CM_atAddr <= CM_atAddr_value;
out_CM_atCE <= CM_atCE_value(0);
out_CM_atResVal <= CM_atResVal_value(0);

---------- premises inputs -------------
prEtapa0_imput_a <= CM_atStep_value;
prEtapa1_imput_a <= CM_atStep_value;
prEtapa10_imput_a <= CM_atStep_value;
prEtapa11_imput_a <= CM_atStep_value;
prEtapa12_imput_a <= CM_atStep_value;
prEtapa13_imput_a <= CM_atStep_value;
prEtapa14_imput_a <= CM_atStep_value;
prEtapa15_imput_a <= CM_atStep_value;
prEtapa16_imput_a <= CM_atStep_value;
prEtapa17_imput_a <= CM_atStep_value;
prEtapa18_imput_a <= CM_atStep_value;
prEtapa19_imput_a <= CM_atStep_value;
prEtapa2_imput_a <= CM_atStep_value;
prEtapa3_imput_a <= CM_atStep_value;
```

```
prEtapa4_imput_a <= CM_atStep_value;
prEtapa5_imput_a <= CM_atStep_value;
prEtapa6_imput_a <= CM_atStep_value;
prEtapa7_imput_a <= CM_atStep_value;
prEtapa8_imput_a <= CM_atStep_value;
prEtapa9_imput_a <= CM_atStep_value;
prReset_imput_a <= CM_atReset_value;

prEtapa0_imput_b <= "000000";
prEtapa1_imput_b <= "000001";
prEtapa10_imput_b <= "001010";
prEtapa11_imput_b <= "001011";
prEtapa12_imput_b <= "001100";
prEtapa13_imput_b <= "001101";
prEtapa14_imput_b <= "001110";
prEtapa15_imput_b <= "001111";
prEtapa16_imput_b <= "010000";
prEtapa17_imput_b <= "010001";
prEtapa18_imput_b <= "010010";
prEtapa19_imput_b <= "010011";
prEtapa2_imput_b <= "000010";
prEtapa3_imput_b <= "000011";
prEtapa4_imput_b <= "000100";
prEtapa5_imput_b <= "000101";
prEtapa6_imput_b <= "000110";
prEtapa7_imput_b <= "000111";
prEtapa8_imput_b <= "001000";
prEtapa9_imput_b <= "001001";
prReset_imput_b <= "1";

---------- Methods Imputs -------------
CM_mtAddr_0_execute <= (prEtapa0_result) OR (prReset_result);
CM_mtCE_1_execute <= (prEtapa0_result) OR (prEtapa10_result)
OR (prEtapa12_result) OR (prEtapa14_result) OR
(prEtapa16_result) OR (prEtapa18_result) OR (prEtapa2_result)
OR (prEtapa4_result) OR (prEtapa6_result) OR
(prEtapa8_result);
CM_mtAtualRes_execute <= (prEtapa0_result);
CM_mtResVal_1_execute <= (prEtapa0_result);
CM_mtIncStep_execute <= (prEtapa0_result) OR (prEtapa1_result)
OR (prEtapa10_result) OR (prEtapa11_result) OR
(prEtapa12_result) OR (prEtapa13_result) OR (prEtapa14_result)
OR (prEtapa15_result) OR (prEtapa16_result) OR
(prEtapa17_result) OR (prEtapa18_result) OR (prEtapa2_result)
OR (prEtapa3_result) OR (prEtapa4_result) OR (prEtapa5_result)
OR (prEtapa6_result) OR (prEtapa7_result) OR (prEtapa8_result)
OR (prEtapa9_result);
CM_mtCE_0_execute <= (prEtapa1_result) OR (prEtapa11_result)
OR (prEtapa13_result) OR (prEtapa15_result) OR
(prEtapa17_result) OR (prEtapa19_result) OR (prEtapa3_result)
```

```
OR (prEtapa5_result) OR (prEtapa7_result) OR (prEtapa9_result)
OR (prReset_result);
 CM_mtTempDataIn_execute <= (prEtapa1_result);
 CM_mtResVal_0_execute <= (prEtapa1_result) OR
(prReset_result);
 CM_mtAddrInc_execute <= (prEtapa10_result) OR
(prEtapa12_result) OR (prEtapa14_result) OR (prEtapa16_result)
OR (prEtapa18_result) OR (prEtapa2_result) OR
(prEtapa4_result) OR (prEtapa6_result) OR (prEtapa8_result);
 CM_mtAcumulaTemp_execute <= (prEtapa11_result) OR
(prEtapa13_result) OR (prEtapa15_result) OR (prEtapa17_result)
OR (prEtapa19_result) OR (prEtapa3_result) OR
(prEtapa5_result) OR (prEtapa7_result) OR (prEtapa9_result);
 CM_mtResetStep_execute <= (prEtapa19_result) OR
(prReset_result);
 CM_mtResetRes_execute <= (prReset_result);
 CM_mtResetRes_input_a <= "0000000000000000";
 CM_mtResetRes_input_b <= "0000000000000000";
 CM_mtResetStep_input_a <= "000000";
 CM_mtResetStep_input_b <= "000000";
 CM_mtIncStep_input_a <= CM_atStep_value;
 CM_mtIncStep_input_b <= "000001";
 CM_mtResVal_0_input_a <= "0";
 CM_mtResVal_0_input_b <= "0";
 CM_mtResVal_1_input_a <= "1";
 CM_mtResVal_1_input_b <= "0";
 CM_mtAddr_0_input_a <= "0000";
 CM_mtAddr_0_input_b <= "0000";
 CM_mtAddrInc_input_a <= CM_atAddr_value;
 CM_mtAddrInc_input_b <= "0001";
 CM_mtCE_0_input_a <= "0";
 CM_mtCE_0_input_b <= "0";
 CM_mtCE_1_input_a <= "1";
 CM_mtCE_1_input_b <= "0";
 CM_mtAtualRes_input_a <= CM_atTemp_value;
 CM_mtAtualRes_input_b <= "0000000000001010";
 CM_mtTempDataIn_input_a <= CM_atDataIn_value;
 CM_mtTempDataIn_input_b <= "0000000000000000";
 CM_mtAcumulaTemp_input_a <= CM_atTemp_value;
 CM_mtAcumulaTemp_input_b <= CM_atDataIn_value;
END calculaMedia_arq;
```

Apêndice N – calculaMedia.vhd

Código fonte em VHDL para o experimento do cálculo de média gerado automaticamente pela ferramenta Vivado HLS.

```
--
==========================================================
-- RTL generated by Vivado(TM) HLS - High-Level Synthesis from
C, C++ and SystemC
-- Version: 2018.1_AR70908
-- Copyright (C) 1986-2018 Xilinx, Inc. All Rights Reserved.
--
-- ==========================================================

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity calculaMedia is
port (
    ap_clk : IN STD_LOGIC;
    ap_rst : IN STD_LOGIC;
    ap_start : IN STD_LOGIC;
    ap_done : OUT STD_LOGIC;
    ap_idle : OUT STD_LOGIC;
    ap_ready : OUT STD_LOGIC;
    v_address0 : OUT STD_LOGIC_VECTOR (3 downto 0);
    v_ce0 : OUT STD_LOGIC;
    v_q0 : IN STD_LOGIC_VECTOR (15 downto 0);
    res : OUT STD_LOGIC_VECTOR (15 downto 0);
    res_ap_vld : OUT STD_LOGIC );
end;


architecture behav of calculaMedia is
    attribute CORE_GENERATION_INFO : STRING;
    attribute CORE_GENERATION_INFO of behav : architecture is

"calculaMedia,hls_ip_2018_1_AR70908,{HLS_INPUT_TYPE=cxx,HLS_IN
PUT_FLOAT=0,HLS_INPUT_FIXED=1,HLS_INPUT_PART=xa7a12tcsg325-
1q,HLS_INPUT_CLOCK=20.000000,HLS_INPUT_ARCH=others,HLS_SYN_CLO
CK=13.412750,HLS_SYN_LAT=21,HLS_SYN_TPT=none,HLS_SYN_MEM=0,HLS
_SYN_DSP=1,HLS_SYN_FF=27,HLS_SYN_LUT=179}";
    constant ap_const_logic_1 : STD_LOGIC := '1';
    constant ap_const_logic_0 : STD_LOGIC := '0';
    constant ap_ST_fsm_state1 : STD_LOGIC_VECTOR (2 downto 0)
:= "001";
    constant ap_ST_fsm_state2 : STD_LOGIC_VECTOR (2 downto 0)
:= "010";
    constant ap_ST_fsm_state3 : STD_LOGIC_VECTOR (2 downto 0)
:= "100";
```

```
    constant ap_const_lv32_0 : STD_LOGIC_VECTOR (31 downto 0)
:= "00000000000000000000000000000000";
    constant ap_const_lv32_1 : STD_LOGIC_VECTOR (31 downto 0)
:= "00000000000000000000000000000001";
    constant ap_const_lv32_2 : STD_LOGIC_VECTOR (31 downto 0)
:= "00000000000000000000000000000010";
    constant ap_const_lv16_0 : STD_LOGIC_VECTOR (15 downto 0)
:= "0000000000000000";
    constant ap_const_lv4_0 : STD_LOGIC_VECTOR (3 downto 0) :=
"0000";
    constant ap_const_lv1_1 : STD_LOGIC_VECTOR (0 downto 0) :=
"1";
    constant ap_const_lv4_A : STD_LOGIC_VECTOR (3 downto 0) :=
"1010";
    constant ap_const_lv4_1 : STD_LOGIC_VECTOR (3 downto 0) :=
"0001";
    constant ap_const_lv33_0 : STD_LOGIC_VECTOR (32 downto 0)
:= "000000000000000000000000000000000";
    constant ap_const_lv32_F : STD_LOGIC_VECTOR (31 downto 0)
:= "00000000000000000000000000001111";
    constant ap_const_lv32_14 : STD_LOGIC_VECTOR (31 downto 0)
:= "00000000000000000000000000010100";
    constant ap_const_lv32_20 : STD_LOGIC_VECTOR (31 downto 0)
:= "00000000000000000000000000100000";
    constant ap_const_lv32_21 : STD_LOGIC_VECTOR (31 downto 0)
:= "00000000000000000000000000100001";
    constant ap_const_lv34_1999A : STD_LOGIC_VECTOR (33 downto
0) := "0000000000000000011001100110011010";
    constant ap_const_boolean_1 : BOOLEAN := true;

    signal ap_CS_fsm : STD_LOGIC_VECTOR (2 downto 0) := "001";
    attribute fsm_encoding : string;
    attribute fsm_encoding of ap_CS_fsm : signal is "none";
    signal ap_CS_fsm_state1 : STD_LOGIC;
    attribute fsm_encoding of ap_CS_fsm_state1 : signal is
"none";
    signal cont_1_fu_92_p2 : STD_LOGIC_VECTOR (3 downto 0);
    signal cont_1_reg_191 : STD_LOGIC_VECTOR (3 downto 0);
    signal ap_CS_fsm_state2 : STD_LOGIC;
    attribute fsm_encoding of ap_CS_fsm_state2 : signal is
"none";
    signal tmp_1_fu_86_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_6_fu_174_p2 : STD_LOGIC_VECTOR (15 downto 0);
    signal ap_CS_fsm_state3 : STD_LOGIC;
    attribute fsm_encoding of ap_CS_fsm_state3 : signal is
"none";
    signal tmp_reg_63 : STD_LOGIC_VECTOR (15 downto 0);
    signal cont_reg_75 : STD_LOGIC_VECTOR (3 downto 0);
    signal tmp_5_fu_98_p1 : STD_LOGIC_VECTOR (63 downto 0);
    signal mul_fu_180_p2 : STD_LOGIC_VECTOR (33 downto 0);
    signal tmp_8_fu_107_p1 : STD_LOGIC_VECTOR (32 downto 0);
```

```
    signal neg_mul_fu_110_p2 : STD_LOGIC_VECTOR (32 downto 0);
    signal tmp_10_fu_124_p4 : STD_LOGIC_VECTOR (12 downto 0);
    signal tmp_11_fu_138_p4 : STD_LOGIC_VECTOR (13 downto 0);
    signal tmp_9_fu_116_p3 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_3_fu_134_p1 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_4_fu_147_p1 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_7_fu_151_p3 : STD_LOGIC_VECTOR (15 downto 0);
    signal neg_ti_fu_159_p2 : STD_LOGIC_VECTOR (15 downto 0);
    signal mul_fu_180_p0 : STD_LOGIC_VECTOR (17 downto 0);
    signal ap_NS_fsm : STD_LOGIC_VECTOR (2 downto 0);

    component calculaMedia_mul_bkb IS
    generic (
        ID : INTEGER;
        NUM_STAGE : INTEGER;
        din0_WIDTH : INTEGER;
        din1_WIDTH : INTEGER;
        dout_WIDTH : INTEGER );
    port (
        din0 : IN STD_LOGIC_VECTOR (17 downto 0);
        din1 : IN STD_LOGIC_VECTOR (15 downto 0);
        dout : OUT STD_LOGIC_VECTOR (33 downto 0) );
    end component;



begin
    calculaMedia_mul_bkb_U1 : component calculaMedia_mul_bkb
    generic map (
        ID => 1,
        NUM_STAGE => 1,
        din0_WIDTH => 18,
        din1_WIDTH => 16,
        dout_WIDTH => 34)
    port map (
        din0 => mul_fu_180_p0,
        din1 => tmp_reg_63,
        dout => mul_fu_180_p2);




    ap_CS_fsm_assign_proc : process(ap_clk)
    begin
        if (ap_clk'event and ap_clk =  '1') then
            if (ap_rst = '1') then
                ap_CS_fsm <= ap_ST_fsm_state1;
            else
                ap_CS_fsm <= ap_NS_fsm;
            end if;
```

```
        end if;
    end process;


    cont_reg_75_assign_proc : process (ap_clk)
    begin
        if (ap_clk'event and ap_clk = '1') then
            if ((ap_const_logic_1 = ap_CS_fsm_state3)) then
                cont_reg_75 <= cont_1_reg_191;
            elsif (((ap_start = ap_const_logic_1) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
                cont_reg_75 <= ap_const_lv4_0;
            end if;
        end if;
    end process;

    tmp_reg_63_assign_proc : process (ap_clk)
    begin
        if (ap_clk'event and ap_clk = '1') then
            if ((ap_const_logic_1 = ap_CS_fsm_state3)) then
                tmp_reg_63 <= tmp_6_fu_174_p2;
            elsif (((ap_start = ap_const_logic_1) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
                tmp_reg_63 <= ap_const_lv16_0;
            end if;
        end if;
    end process;
    process (ap_clk)
    begin
        if (ap_clk'event and ap_clk = '1') then
            if ((ap_const_logic_1 = ap_CS_fsm_state2)) then
                cont_1_reg_191 <= cont_1_fu_92_p2;
            end if;
        end if;
    end process;

    ap_NS_fsm_assign_proc : process (ap_start, ap_CS_fsm,
ap_CS_fsm_state1, ap_CS_fsm_state2, tmp_1_fu_86_p2)
    begin
        case ap_CS_fsm is
            when ap_ST_fsm_state1 =>
                if (((ap_start = ap_const_logic_1) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
                    ap_NS_fsm <= ap_ST_fsm_state2;
                else
                    ap_NS_fsm <= ap_ST_fsm_state1;
                end if;
            when ap_ST_fsm_state2 =>
                if (((tmp_1_fu_86_p2 = ap_const_lv1_1) and
(ap_const_logic_1 = ap_CS_fsm_state2))) then
                    ap_NS_fsm <= ap_ST_fsm_state1;
```

```
                else
                    ap_NS_fsm <= ap_ST_fsm_state3;
                end if;
            when ap_ST_fsm_state3 =>
                ap_NS_fsm <= ap_ST_fsm_state2;
            when others =>
                ap_NS_fsm <= "XXX";
        end case;
    end process;
    ap_CS_fsm_state1 <= ap_CS_fsm(0);
    ap_CS_fsm_state2 <= ap_CS_fsm(1);
    ap_CS_fsm_state3 <= ap_CS_fsm(2);

    ap_done_assign_proc : process(ap_CS_fsm_state2,
tmp_1_fu_86_p2)
    begin
        if (((tmp_1_fu_86_p2 = ap_const_lv1_1) and
(ap_const_logic_1 = ap_CS_fsm_state2))) then
            ap_done <= ap_const_logic_1;
        else
            ap_done <= ap_const_logic_0;
        end if;
    end process;

    ap_idle_assign_proc : process(ap_start, ap_CS_fsm_state1)
    begin
        if (((ap_start = ap_const_logic_0) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
            ap_idle <= ap_const_logic_1;
        else
            ap_idle <= ap_const_logic_0;
        end if;
    end process;

    ap_ready_assign_proc : process(ap_CS_fsm_state2,
tmp_1_fu_86_p2)
    begin
        if (((tmp_1_fu_86_p2 = ap_const_lv1_1) and
(ap_const_logic_1 = ap_CS_fsm_state2))) then
            ap_ready <= ap_const_logic_1;
        else
            ap_ready <= ap_const_logic_0;
        end if;
    end process;
    cont_1_fu_92_p2 <= std_logic_vector(unsigned(cont_reg_75)
+ unsigned(ap_const_lv4_1));
    mul_fu_180_p0 <= ap_const_lv34_1999A(18 - 1 downto 0);
    neg_mul_fu_110_p2 <=
std_logic_vector(unsigned(ap_const_lv33_0) -
unsigned(tmp_8_fu_107_p1));
```

```
    neg_ti_fu_159_p2 <=
std_logic_vector(unsigned(ap_const_lv16_0) -
unsigned(tmp_7_fu_151_p3));
    res <=
        neg_ti_fu_159_p2 when (tmp_9_fu_116_p3(0) = '1') else
        tmp_4_fu_147_p1;
    res_ap_vld_assign_proc : process(ap_CS_fsm_state2,
tmp_1_fu_86_p2)
    begin
        if (((tmp_1_fu_86_p2 = ap_const_lv1_1) and
(ap_const_logic_1 = ap_CS_fsm_state2))) then
            res_ap_vld <= ap_const_logic_1;
        else
            res_ap_vld <= ap_const_logic_0;
        end if;
    end process;

    tmp_10_fu_124_p4 <= neg_mul_fu_110_p2(32 downto 20);
    tmp_11_fu_138_p4 <= mul_fu_180_p2(33 downto 20);
    tmp_1_fu_86_p2 <= "1" when (cont_reg_75 = ap_const_lv4_A)
else "0";
        tmp_3_fu_134_p1 <=
std_logic_vector(IEEE.numeric_std.resize(signed(tmp_10_fu_124_
p4),16));

        tmp_4_fu_147_p1 <=
std_logic_vector(IEEE.numeric_std.resize(signed(tmp_11_fu_138_
p4),16));
    tmp_5_fu_98_p1 <=
std_logic_vector(IEEE.numeric_std.resize(unsigned(cont_reg_75)
,64));
    tmp_6_fu_174_p2 <= std_logic_vector(unsigned(v_q0) +
unsigned(tmp_reg_63));
    tmp_7_fu_151_p3 <=
        tmp_3_fu_134_p1 when (tmp_9_fu_116_p3(0) = '1') else
        tmp_4_fu_147_p1;
    tmp_8_fu_107_p1 <= mul_fu_180_p2(33 - 1 downto 0);
    tmp_9_fu_116_p3 <= tmp_reg_63(15 downto 15);
    v_address0 <= tmp_5_fu_98_p1(4 - 1 downto 0);

    v_ce0_assign_proc : process(ap_CS_fsm_state2)
    begin
        if ((ap_const_logic_1 = ap_CS_fsm_state2)) then
            v_ce0 <= ap_const_logic_1;
        else
            v_ce0 <= ap_const_logic_0;
        end if;
    end process;

end behav;
```

Apêndice O – Código do ordenador de dados utilizando PON-HD 1.0

Código fonte em VHDL para o experimento do ordenador de dados gerado manualmente utilizando o *framework* PON-HD 1.0.

```
-- Ordenador de dados PON-HD 1.0
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE work.data_type_pkg.all;

ENTITY NopSort2 IS
GENERIC (N: INTEGER := 16);
PORT(
 in_atData1 :IN STD_LOGIC_VECTOR(15 downto 0);
 in_set_atData1 :IN STD_LOGIC;
 out_atData1 :OUT STD_LOGIC_VECTOR(15 downto 0);
 out_finished :OUT STD_LOGIC;
 clock :IN STD_LOGIC
 );
END NopSort2;

ARCHITECTURE NopSort2_arq OF NopSort2 IS
----------------- COMPONENTS --------------------
COMPONENT NOP_attribute
 GENERIC (
  N_bits: INTEGER;
  N_new_values: INTEGER;
  initial_value: STD_LOGIC_VECTOR
  );
 PORT(
  att_clock :IN STD_LOGIC;
  att_new_value :IN data(0 TO N_new_values-1)(N_bits-1 downto 0);
  att_set_value :IN STD_LOGIC_VECTOR(N_new_values-1 downto 0);
  att_value :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0)
  );
END COMPONENT;
COMPONENT NOP_premise
 GENERIC (
  N_bits: INTEGER;
  opperation: INTEGER
  );
 PORT(
  att_value_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
  att_value_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
  pre_result :OUT STD_LOGIC
  );
END COMPONENT;
COMPONENT NOP_method
 GENERIC (
  N_bits: INTEGER;
```

```
 operation: INTEGER
 );
 PORT(
 in_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 in_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
 execute :IN STD_LOGIC;
 result :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0);
 notify :OUT STD_LOGIC
 );
END COMPONENT;
----------------- SIGNALS --------------------
 TYPE datax IS ARRAY (0 TO N-1) OF INTEGER;
 CONSTANT              init:              datax              :=
(15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0); -- 16
 TYPE sets IS ARRAY (0 TO N-1) OF STD_LOGIC_VECTOR(1 downto 0);
 SIGNAL att_set: sets;
 SIGNAL attOdd_set: STD_LOGIC_VECTOR(1 downto 0);
 SIGNAL att_new_value: dataArray(0 TO N-1);
 SIGNAL attOdd_new_value: data(0 TO 1)(0 downto 0);
 TYPE values IS ARRAY (0 TO N-1) OF STD_LOGIC_VECTOR(15 downto
0);
 SIGNAL att_value: values;
 SIGNAL attOdd_value: STD_LOGIC_VECTOR(0 downto 0);
 TYPE preRes IS ARRAY (0 TO N-2) OF STD_LOGIC;
 SIGNAL pre_result: preRes;
 SIGNAL preOdd_result: STD_LOGIC;
 TYPE ruleRes IS ARRAY (0 TO N-2) OF STD_LOGIC;
 SIGNAL rule_result: ruleRes;
 TYPE endArray IS ARRAY (0 TO N-2) OF STD_LOGIC;
 SIGNAL ended: endArray;

BEGIN
Generate_Attributes:
 for I in 0 to N-1 generate
 att:                   NOP_attribute                    GENERIC
MAP(16,2,conv_std_logic_vector(init(I),16))                PORT
MAP(clock,att_new_value(I),att_set(I),att_value(I));
 end generate Generate_Attributes;

Generate_premises_methods:
 for J in 0 to N-2 generate
 pre:       NOP_premise       GENERIC       MAP(16,4)       PORT
MAP(att_value(J),att_value(J+1),pre_result(j));
 met0:       NOP_method       GENERIC       MAP(16,0)       PORT
MAP(att_value(J),"0000000000000000",rule_result(J),att_new_val
ue(J+1)(0),att_set(J+1)(0));
 met1:       NOP_method       GENERIC       MAP(16,0)       PORT
MAP(att_value(J+1),"0000000000000000",rule_result(J),att_new_v
alue(J)(1),att_set(J)(1));
 end generate Generate_premises_methods;
```

```
Generate_even_rules:
 for K in 0 to (N-2)/2 generate
  rule_result(K*2) <= pre_result(K*2) AND (NOT attOdd_value(0));
 end generate Generate_even_rules;

Generate_odd_rules:
 for L in 0 to ((N-2)/2)-1 generate
  rule_result((L*2)+1)      <=      pre_result((L*2)+1)      AND
attOdd_value(0);
 end generate Generate_odd_rules;

 ended(0) <= NOT pre_result(0);
Generate_ended:
 for M in 1 to N-2 generate
  ended(M) <= ended(M-1) AND (NOT pre_result(M));
 end generate Generate_ended;

 -- to generate the odd signal
 attOdd:     NOP_attribute     GENERIC     MAP(1,2,"0")     PORT
MAP(clock,attOdd_new_value,attOdd_set,attOdd_value);
 preOdd:       NOP_premise       GENERIC       MAP(1,1)       PORT
MAP(attOdd_value,"0",preOdd_result);
 metOdd0:  NOP_method  GENERIC  MAP(1,0)  PORT  MAP("0","0",NOT
preOdd_result,attOdd_new_value(0),attOdd_set(0));
 metOdd1:       NOP_method       GENERIC       MAP(1,0)       PORT
MAP("1","0",preOdd_result,attOdd_new_value(1),attOdd_set(1));

 -- the i/o signals
 att_new_value(0)(0) <= in_atData1;
 att_set(0)(0) <= in_set_atData1;
 out_atData1 <= att_value(0);

 -- to signalize the end of sort
 out_finished <= ended(N-2);

 -- unused attribute imput
 att_set(N-1)(1) <= '0';
 att_new_value(N-1)(1) <= "0000000000000000";

END NopSort2_arq;
```

Apêndice P – Código do ordenador de dados em VHDL

Código fonte em VHDL para o experimento do ordenador de dados gerado manualmente sem a utilização do *framework* PON-HD 1.0.

```
-- Ordenador de dados VHDL
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;

ENTITY NopSort4 IS
GENERIC (N: INTEGER := 16);
PORT(
 in_atData1 :IN STD_LOGIC_VECTOR(15 downto 0);
 in_set_atData1 :IN STD_LOGIC;
 out_atData1 :OUT STD_LOGIC_VECTOR(15 downto 0);
 out_finished :OUT STD_LOGIC;
 clock :IN STD_LOGIC
 );
END NopSort4;

ARCHITECTURE NopSort4_arq OF NopSort4 IS
------------------ SIGNALS ---------------------
 TYPE tdata IS ARRAY (0 TO N-1) OF INTEGER RANGE -32768 TO 32767;
 SIGNAL data: tdata := (15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0);
-- 16
 TYPE stdarray IS ARRAY (0 TO N-2) OF STD_LOGIC;
 SIGNAL result: stdarray;
 SIGNAL compare: stdarray;
 SIGNAL ended: stdarray;
 SIGNAL odd: STD_LOGIC;

BEGIN
Generate_compare:
 for I in 0 to N-2 generate
  compare(I)<='1' WHEN (data(I) > data(I+1)) ELSE
      '0';
 end generate Generate_compare;

Generate_even:
 for K in 0 to (N-2)/2 generate
  result(K*2) <= compare(K*2) AND (NOT odd);
 end generate Generate_even;

Generate_odd:
 for L in 0 to ((N-2)/2)-1 generate
  result((L*2)+1) <= compare((L*2)+1) AND odd;
 end generate Generate_odd;

 PROCESS (clock)
 BEGIN
```

```
 IF (clock'EVENT AND clock = '1') THEN
  IF (odd = '1') THEN
   odd<='0';
  ELSE
   odd<='1';
  END IF;
  IF (in_set_atData1 = '1') THEN
   data(0)<=to_integer(signed(in_atData1));
  ELSE
   if(result(0)='1') THEN
    data(1) <= data(0);
    data(0) <= data(1);
   END IF;
  END IF;
  FOR J IN 1 to N-2 LOOP
   if(result(J)='1') THEN
    data(J+1) <= data(J);
    data(J) <= data(J+1);
   END IF;
  END LOOP;
 END IF;
END PROCESS;

 ended(0) <= NOT result(0);
Generate_ended:
 for K in 1 to N-2 generate
  ended(K) <= ended(K-1) AND (NOT result(K));
 end generate Generate_ended;
 out_finished <= ended(N-2);

 out_atData1 <= std_logic_vector(to_signed(data(0),16));

END NopSort4_arq;
```

Apêndice Q – OrdenadorLingPON.pon

Código fonte em LingPON-HD 1.0 para o experimento do ordenador de dados.

```
// ordenador de dados para 16 elementos inversamente ordenados
fbe fbeOrdenador
 attributes
  integer atData1 15
  integer atData2 14
  integer atData3 13
  integer atData4 12
  integer atData5 11
  integer atData6 10
  integer atData7 9
  integer atData8 8
  integer atData9 7
  integer atData10 6
  integer atData11 5
  integer atData12 4
  integer atData13 3
  integer atData14 2
  integer atData15 1
  integer atData16 0

  boolean finished false

  boolean atOdd false
 end_attributes
 methods
  method mtSwap12(atData1 = atData2)
  method mtSwap21(atData2 = atData1)
  method mtSwap23(atData2 = atData3)
  method mtSwap32(atData3 = atData2)
  method mtSwap34(atData3 = atData4)
  method mtSwap43(atData4 = atData3)
  method mtSwap45(atData4 = atData5)
  method mtSwap54(atData5 = atData4)
  method mtSwap56(atData5 = atData6)
  method mtSwap65(atData6 = atData5)
  method mtSwap67(atData6 = atData7)
  method mtSwap76(atData7 = atData6)
  method mtSwap78(atData7 = atData8)
  method mtSwap87(atData8 = atData7)
  method mtSwap89(atData8 = atData9)
  method mtSwap98(atData9 = atData8)
  method mtSwap910(atData9 = atData10)
  method mtSwap109(atData10 = atData9)
  method mtSwap1011(atData10 = atData11)
  method mtSwap1110(atData11 = atData10)
  method mtSwap1112(atData11 = atData12)
  method mtSwap1211(atData12 = atData11)
```

```
 method mtSwap1213(atData12 = atData13)
 method mtSwap1312(atData13 = atData12)
 method mtSwap1314(atData13 = atData14)
 method mtSwap1413(atData14 = atData13)
 method mtSwap1415(atData14 = atData15)
 method mtSwap1514(atData15 = atData14)
 method mtSwap1516(atData15 = atData16)
 method mtSwap1615(atData16 = atData15)

 method mtFinished(finished = true)

 method mtOdd(atOdd = true)
 method mtNOdd(atOdd = false)
end_methods
end_fbe

inst
 fbeOrdenador Ordenador
end_inst

strategy
 no_one
end_strategy

rule RlSwap12
 condition
  subcondition SubSwap12
   premise pr1g2 Ordenador.atData1 > Ordenador.atData2 and
   premise prOddI Ordenador.atOdd == false
  end_subcondition
 end_condition
 action
  instigation inSwap12 Ordenador.mtSwap12();
  instigation inSwap21 Ordenador.mtSwap21();
 end_action
end_rule

rule RlSwap23
 condition
  subcondition SubSwap23
   premise pr2g3 Ordenador.atData2 > Ordenador.atData3 and
   premise prOdd Ordenador.atOdd == true
  end_subcondition
 end_condition
 action
  instigation inSwap23 Ordenador.mtSwap23();
  instigation inSwap32 Ordenador.mtSwap32();
 end_action
end_rule

rule RlSwap34
```

```
 condition
  subcondition SubSwap34
   premise pr3g4 Ordenador.atData3 > Ordenador.atData4 and
   premise prOddI Ordenador.atOdd == false
  end_subcondition
 end_condition
 action
  instigation inSwap34 Ordenador.mtSwap34();
  instigation inSwap43 Ordenador.mtSwap43();

 end_action
end_rule

rule RlSwap45
 condition
  subcondition SubSwap45
   premise pr4g5 Ordenador.atData4 > Ordenador.atData5 and
   premise prOdd Ordenador.atOdd == true
  end_subcondition
 end_condition
 action
  instigation inSwap45 Ordenador.mtSwap45();
  instigation inSwap54 Ordenador.mtSwap54();
 end_action
end_rule

rule RlSwap56
 condition
  subcondition SubSwap56
   premise pr5g6 Ordenador.atData5 > Ordenador.atData6 and
   premise prOddI Ordenador.atOdd == false
  end_subcondition
 end_condition
 action
  instigation inSwap56 Ordenador.mtSwap56();
  instigation inSwap65 Ordenador.mtSwap65();
 end_action
end_rule

rule RlSwap67
 condition
  subcondition SubSwap67
   premise pr6g7 Ordenador.atData6 > Ordenador.atData7 and
   premise prOdd Ordenador.atOdd == true
  end_subcondition
 end_condition
 action
  instigation inSwap67 Ordenador.mtSwap67();
  instigation inSwap76 Ordenador.mtSwap76();
 end_action
end_rule
```

```
rule RlSwap78
 condition
  subcondition SubSwap78
   premise pr7g8 Ordenador.atData7 > Ordenador.atData8 and
   premise prOddI Ordenador.atOdd == false
  end_subcondition
 end_condition
 action
  instigation inSwap78 Ordenador.mtSwap78();
  instigation inSwap87 Ordenador.mtSwap87();
 end_action
end_rule

rule RlSwap89
 condition
  subcondition SubSwap89
   premise pr8g9 Ordenador.atData8 > Ordenador.atData9 and
   premise prOdd Ordenador.atOdd == true
  end_subcondition
 end_condition
 action
  instigation inSwap89 Ordenador.mtSwap89();
  instigation inSwap98 Ordenador.mtSwap98();
 end_action
end_rule

rule RlSwap910
 condition
  subcondition SubSwap910
   premise pr9g10 Ordenador.atData9 > Ordenador.atData10 and
   premise prOddI Ordenador.atOdd == false
  end_subcondition
 end_condition
 action
  instigation inSwap910 Ordenador.mtSwap910();
  instigation inSwap109 Ordenador.mtSwap109();
 end_action
end_rule

rule RlSwap1011
 condition
  subcondition SubSwap1011
   premise pr10g11 Ordenador.atData10 > Ordenador.atData11 and
   premise prOdd Ordenador.atOdd == true
  end_subcondition
 end_condition
 action
  instigation inSwap1011 Ordenador.mtSwap1011();
  instigation inSwap1110 Ordenador.mtSwap1110();
 end_action
```

```
end_rule

rule RlSwap1112
 condition
  subcondition SubSwap1112
   premise pr11g12 Ordenador.atData11 > Ordenador.atData12 and
   premise prOddI Ordenador.atOdd == false
  end_subcondition
 end_condition
 action
  instigation inSwap1112 Ordenador.mtSwap1112();
  instigation inSwap1211 Ordenador.mtSwap1211();
 end_action
end_rule

rule RlSwap1213
 condition
  subcondition SubSwap1213
   premise pr12g13 Ordenador.atData12 > Ordenador.atData13 and
   premise prOdd Ordenador.atOdd == true
  end_subcondition
 end_condition
 action
  instigation inSwap1213 Ordenador.mtSwap1213();
  instigation inSwap1312 Ordenador.mtSwap1312();
 end_action
end_rule

rule RlSwap1314
 condition
  subcondition SubSwap1314
   premise pr13g14 Ordenador.atData13 > Ordenador.atData14 and
   premise prOddI Ordenador.atOdd == false
  end_subcondition
 end_condition
 action
  instigation inSwap1314 Ordenador.mtSwap1314();
  instigation inSwap1413 Ordenador.mtSwap1413();
 end_action
end_rule

rule RlSwap1415
 condition
  subcondition SubSwap1415
   premise pr14g15 Ordenador.atData14 > Ordenador.atData15 and
   premise prOdd Ordenador.atOdd == true
  end_subcondition
 end_condition
 action
  instigation inSwap1415 Ordenador.mtSwap1415();
  instigation inSwap1514 Ordenador.mtSwap1514();
```

```
 end_action
end_rule

rule RlSwap1516
 condition
  subcondition SubSwap1516
   premise pr15g16 Ordenador.atData15 > Ordenador.atData16 and
   premise prOddI Ordenador.atOdd == false
  end_subcondition
 end_condition
 action
  instigation inSwap1516 Ordenador.mtSwap1516();
  instigation inSwap1615 Ordenador.mtSwap1615();
 end_action
end_rule

rule RlOdd
 condition
  subcondition SubOdd
   premise prOdd Ordenador.atOdd == true
  end_subcondition
 end_condition
 action
  instigation inNOdd Ordenador.mtNOdd();
 end_action
end_rule

rule RlNOdd
 condition
  subcondition SubNOdd
   premise prOddI Ordenador.atOdd == false
  end_subcondition
 end_condition
 action
  instigation inOdd Ordenador.mtOdd();
 end_action
end_rule

rule Rlfinished
 condition
  subcondition Subfinished
   premise pr1ng2 Ordenador.atData1 <= Ordenador.atData2 and
   premise pr2ng3 Ordenador.atData2 <= Ordenador.atData3 and
   premise pr3ng4 Ordenador.atData3 <= Ordenador.atData4 and
   premise pr4ng5 Ordenador.atData4 <= Ordenador.atData5 and
   premise pr5ng6 Ordenador.atData5 <= Ordenador.atData6 and
   premise pr6ng7 Ordenador.atData6 <= Ordenador.atData7 and
   premise pr7ng8 Ordenador.atData7 <= Ordenador.atData8 and
   premise pr8ng9 Ordenador.atData8 <= Ordenador.atData9 and
   premise pr9ng10 Ordenador.atData9 <= Ordenador.atData10 and
   premise pr10ng11 Ordenador.atData10 <= Ordenador.atData11 and
```

```
 premise pr11ng12 Ordenador.atData11 <= Ordenador.atData12 and
 premise pr12ng13 Ordenador.atData12 <= Ordenador.atData13 and
 premise pr13ng14 Ordenador.atData13 <= Ordenador.atData14 and
 premise pr14ng15 Ordenador.atData14 <= Ordenador.atData15 and
 premise pr15ng16 Ordenador.atData15 <= Ordenador.atData16
 end_subcondition
end_condition
action
 instigation inFinished Ordenador.mtFinished();
end_action
end_rule

main {
 entity OrdenadorLingPON

 in Ordenador.atData1
 out Ordenador.atData1
 out Ordenador.finished

 NBits Ordenador.atData1 16
 NBits Ordenador.atData2 16
 NBits Ordenador.atData3 16
 NBits Ordenador.atData4 16
 NBits Ordenador.atData5 16
 NBits Ordenador.atData6 16
 NBits Ordenador.atData7 16
 NBits Ordenador.atData8 16
 NBits Ordenador.atData9 16
 NBits Ordenador.atData10 16
 NBits Ordenador.atData11 16
 NBits Ordenador.atData12 16
 NBits Ordenador.atData13 16
 NBits Ordenador.atData14 16
 NBits Ordenador.atData15 16
 NBits Ordenador.atData16 16
 }
```

Apêndice R – controladorHexapod.pon

Código fonte em LingPON-HD 1.0 para o experimento do controlador de robô hexápode.

```
fbe f_controlador
 attributes
 boolean txempty false
 boolean ativo false
 boolean sensorEsq false
 boolean sensorDir false
 boolean liga false
 boolean desliga false
 integer direcao 0 // 0 para frente, 1 para direita e 2 para
esquerda
 integer contador 0
 integer dado 13
 integer passo 1 // etapas do passo 0 - 3
 integer tempo 0
 end_attributes
 methods
 method mtPara(dado = 13)
 method mtLiga(ativo = true)
 method mtDesliga(ativo = false)
 method mtAnda(direcao = 0)
 method mtViraDireita(direcao = 1)
 method mtZerraTempo(tempo = 0)
 method mtViraEsquerda(direcao = 2)
 method mtIncTempo(tempo = tempo + 1)
 method mtZerraPasso(passo = 0)
 method mtIncPasso(passo = passo + 1)
 method mtZerraContador(contador = 0)
 method mtIncContador(contador = contador + 1)
 method mtSeleciona0(dado = motores.motor0)
 method mtSeleciona1(dado = motores.motor1)
 method mtSeleciona2(dado = motores.motor2)
 method mtSeleciona3(dado = motores.motor3)
 method mtSeleciona4(dado = motores.motor4)
 method mtSeleciona5(dado = motores.motor5)
 method mtSeleciona6(dado = motores.motor6)
 method mtSeleciona7(dado = motores.motor7)
 method mtSeleciona8(dado = motores.motor8)
 method mtSeleciona9(dado = motores.motor9)
 method mtSeleciona10(dado = motores.motor10)
 method mtSeleciona11(dado = motores.motor11)
 method mtSeleciona12(dado = motores.motor12)
 method mtSeleciona13(dado = motores.motor13)
 method mtSeleciona14(dado = motores.motor14)
 method mtSeleciona15(dado = motores.motor15)
 method mtSeleciona16(dado = motores.motor16)
 method mtSeleciona17(dado = motores.motor17)
```

```
 method mtSeleciona18(dado = 13)
 end_methods
end_fbe

fbe f_motores
 attributes
 integer motor0 16
 integer motor1 -19
 integer motor2 60

 integer motor3 0
 integer motor4 -19
 integer motor5 60

 integer motor6 -16
 integer motor7 -19
 integer motor8 60

 integer motor9 16
 integer motor10 -19
 integer motor11 60

 integer motor12 0
 integer motor13 -19
 integer motor14 60

 integer motor15 -16
 integer motor16 -19
 integer motor17 60
 end_attributes
 methods
 // passo 0
 method mtMotor0Passo0(motor0 = 16)
 method mtMotor1Passo0(motor1 = -12)

 method mtMotor3Passo0(motor3 = 0)
 method mtMotor4Passo0(motor4 = -19)

 method mtMotor6Passo0(motor6 = -16)
 method mtMotor7Passo0(motor7 = -12)

 method mtMotor9Passo0(motor9 = 16)
 method mtMotor10Passo0(motor10 = -19)

 method mtMotor12Passo0(motor12 = 0)
 method mtMotor13Passo0(motor13 = -12)

 method mtMotor15Passo0(motor15 = -16)
 method mtMotor16Passo0(motor16 = -19)

 // passo 1 esquerda
```

```
method mtMotor0Passo1Esq(motor0 = 11)
method mtMotor1Passo1Esq(motor1 = -12)

method mtMotor3Passo3Esq(motor3 = -5)
method mtMotor4Passo1Esq(motor4 = -19)

method mtMotor6Passo1Esq(motor6 = -21)
method mtMotor7Passo1Esq(motor7 = -12)

// frente

method mtMotor0Passo1(motor0 = 21)
method mtMotor1Passo1(motor1 = -12)

method mtMotor3Passo3(motor3 = 5)
method mtMotor4Passo1(motor4 = -19)

method mtMotor6Passo1(motor6 = -11)
method mtMotor7Passo1(motor7 = -12)

method mtMotor9Passo1(motor9 = 21)
method mtMotor10Passo1(motor10 = -19)

method mtMotor12Passo3(motor12 = 5)
method mtMotor13Passo1(motor13 = -12)

method mtMotor15Passo1(motor15 = -11)
method mtMotor16Passo1(motor16 = -19)

// direita
method mtMotor9Passo1Dir(motor9 = 11)
method mtMotor10Passo1Dir(motor10 = -19)

method mtMotor12Passo3Dir(motor12 = -5)
method mtMotor13Passo1Dir(motor13 = -12)

method mtMotor15Passo1Dir(motor15 = -21)
method mtMotor16Passo1Dir(motor16 = -19)

// passo 2
method mtMotor0Passo2(motor0 = 16)
method mtMotor1Passo2(motor1 = -19)

method mtMotor3Passo2(motor3 = 0)
method mtMotor4Passo2(motor4 = -12)

method mtMotor6Passo2(motor6 = -16)
method mtMotor7Passo2(motor7 = -19)

method mtMotor9Passo2(motor9 = 16)
```

```
 method mtMotor10Passo2(motor10 = -12)

 method mtMotor12Passo2(motor12 = 0)
 method mtMotor13Passo2(motor13 = -19)

 method mtMotor15Passo2(motor15 = -16)
 method mtMotor16Passo2(motor16 = -12)

 // passo 3 esquerda
 method mtMotor0Passo3Esq(motor0 = 21)
 method mtMotor1Passo3Esq(motor1 = -19)

 method mtMotor3Passo1Esq(motor3 = 5)
 method mtMotor4Passo3Esq(motor4 = -12)

 method mtMotor6Passo3Esq(motor6 = -11)
 method mtMotor7Passo3Esq(motor7 = -19)

 // frente
 method mtMotor0Passo3(motor0 = 11)
 method mtMotor1Passo3(motor1 = -19)

 method mtMotor3Passo1(motor3 = -5)
 method mtMotor4Passo3(motor4 = -12)

 method mtMotor6Passo3(motor6 = -21)
 method mtMotor7Passo3(motor7 = -19)

 method mtMotor9Passo3(motor9 = 11)
 method mtMotor10Passo3(motor10 = -12)

 method mtMotor12Passo1(motor12 = -5)
 method mtMotor13Passo3(motor13 = -19)

 method mtMotor15Passo3(motor15 = -21)
 method mtMotor16Passo3(motor16 = -12)

 // direita
 method mtMotor9Passo3Dir(motor9 = 21)
 method mtMotor10Passo3Dir(motor10 = -12)

 method mtMotor12Passo1Dir(motor12 = 5)
 method mtMotor13Passo3Dir(motor13 = -19)

 method mtMotor15Passo3Dir(motor15 = -11)
 method mtMotor16Passo3Dir(motor16 = -12)
 end_methods
end_fbe

inst
 f_controlador cn
```

```
 f_motores motores
end_inst

strategy
 no_one
end_strategy

rule RlParado
 condition
  subcondition SubParado
   premise prParado cn.ativo == false
  end_subcondition
 end_condition
 action
  instigation inParado cn.mtPara();
 end_action
end_rule

rule RlLiga
 condition
  subcondition SubLiga
   premise prLiga cn.liga == false and
   premise prDesl cn.desliga == true
  end_subcondition
 end_condition
 action
  instigation inLiga cn.mtLiga();
 end_action
end_rule

rule RlDesliga
 condition
  subcondition SubDesliga
   premise prDesliga cn.desliga == false
  end_subcondition
 end_condition
 action
  instigation inDesliga cn.mtDesliga();
 end_action
end_rule

rule RlAnda
 condition
  subcondition SubAnda
   premise prAndaSd cn.sensorDir == false and
   premise prAndaSe cn.sensorEsq == false
  end_subcondition
 end_condition
 action
  instigation inAnda cn.mtAnda();
 end_action
```

```
end_rule

rule RlViraEsquerda
 condition
  subcondition SubViEs
   premise prViEsSd cn.sensorDir == false and
   premise prViEsSe cn.sensorEsq == true
  end_subcondition
 end_condition
 action
  instigation inViraEsquerda cn.mtViraEsquerda();
 end_action
end_rule

rule RlViraDireita
 condition
  subcondition SubViDi
   premise prViDiSd cn.sensorDir == true
  end_subcondition
 end_condition
 action
  instigation inViraDireita cn.mtViraDireita();
 end_action
end_rule

rule RlPasso0
 condition
  subcondition SubPasso0
   premise prPasso0 cn.passo == 0
  end_subcondition
 end_condition
 action
  instigation inPasso0_0 motores.mtMotor0Passo0();
  instigation inPasso0_1 motores.mtMotor1Passo0();

  instigation inPasso0_3 motores.mtMotor3Passo0();
  instigation inPasso0_4 motores.mtMotor4Passo0();

  instigation inPasso0_6 motores.mtMotor6Passo0();
  instigation inPasso0_7 motores.mtMotor7Passo0();

  instigation inPasso0_9 motores.mtMotor9Passo0();
  instigation inPasso0_10 motores.mtMotor10Passo0();

  instigation inPasso0_12 motores.mtMotor12Passo0();
  instigation inPasso0_13 motores.mtMotor13Passo0();

  instigation inPasso0_15 motores.mtMotor15Passo0();
  instigation inPasso0_16 motores.mtMotor16Passo0();
 end_action
end_rule
```

```
rule RlPasso1frente
 condition
  subcondition SubPasso1frente
   premise prPasso1f cn.passo == 1 and
   premise prDirF1cn.direcao == 0
  end_subcondition
 end_condition
 action
  instigation inPasso1_0f motores.mtMotor0Passo1();
  instigation inPasso1_1f motores.mtMotor1Passo1();

  instigation inPasso1_3f motores.mtMotor3Passo1();
  instigation inPasso1_4f motores.mtMotor4Passo1();

  instigation inPasso1_6f motores.mtMotor6Passo1();
  instigation inPasso1_7f motores.mtMotor7Passo1();

  instigation inPasso1_9f motores.mtMotor9Passo1();
  instigation inPasso1_10f motores.mtMotor10Passo1();

  instigation inPasso1_12f motores.mtMotor12Passo1();
  instigation inPasso1_13f motores.mtMotor13Passo1();

  instigation inPasso1_15f motores.mtMotor15Passo1();
  instigation inPasso1_16f motores.mtMotor16Passo1();
 end_action
end_rule

rule RlPasso1esquerda
 condition
  subcondition SubPasso1esquerda
   premise prPasso1e cn.passo == 1 and
   premise prDirE1cn.direcao == 2
  end_subcondition
 end_condition
 action
  instigation inPasso1_0e motores.mtMotor0Passo1Esq();
  instigation inPasso1_1e motores.mtMotor1Passo1Esq();

  instigation inPasso1_3e motores.mtMotor3Passo1Esq();
  instigation inPasso1_4e motores.mtMotor4Passo1Esq();

  instigation inPasso1_6e motores.mtMotor6Passo1Esq();
  instigation inPasso1_7e motores.mtMotor7Passo1Esq();

  instigation inPasso1_9e motores.mtMotor9Passo1();
  instigation inPasso1_10e motores.mtMotor10Passo1();

  instigation inPasso1_12e motores.mtMotor12Passo1();
  instigation inPasso1_13e motores.mtMotor13Passo1();
```

```
 instigation inPasso1_15e motores.mtMotor15Passo1();
 instigation inPasso1_16e motores.mtMotor16Passo1();
end_action
end_rule

rule RlPasso1direita
condition
 subcondition SubPasso1direita
  premise prPasso1d cn.passo == 1 and
  premise prDirD1cn.direcao == 0
 end_subcondition
end_condition
action
 instigation inPasso1_0d motores.mtMotor0Passo1();
 instigation inPasso1_1d motores.mtMotor1Passo1();

 instigation inPasso1_3d motores.mtMotor3Passo1();
 instigation inPasso1_4d motores.mtMotor4Passo1();

 instigation inPasso1_6d motores.mtMotor6Passo1();
 instigation inPasso1_7d motores.mtMotor7Passo1();

 instigation inPasso1_9d motores.mtMotor9Passo1Dir();
 instigation inPasso1_10d motores.mtMotor10Passo1Dir();

 instigation inPasso1_12d motores.mtMotor12Passo1Dir();
 instigation inPasso1_13d motores.mtMotor13Passo1Dir();

 instigation inPasso1_15d motores.mtMotor15Passo1Dir();
 instigation inPasso1_16d motores.mtMotor16Passo1Dir();
end_action
end_rule

rule RlPasso2
condition
 subcondition SubPasso2
  premise prPasso2 cn.passo == 2
 end_subcondition
end_condition
action
 instigation inPasso2_0 motores.mtMotor0Passo2();
 instigation inPasso2_1 motores.mtMotor1Passo2();

 instigation inPasso2_3 motores.mtMotor3Passo2();
 instigation inPasso2_4 motores.mtMotor4Passo2();

 instigation inPasso2_6 motores.mtMotor6Passo2();
 instigation inPasso2_7 motores.mtMotor7Passo2();

 instigation inPasso2_9 motores.mtMotor9Passo2();
```

```
 instigation inPasso2_10 motores.mtMotor10Passo2();

 instigation inPasso2_12 motores.mtMotor12Passo2();
 instigation inPasso2_13 motores.mtMotor13Passo2();

 instigation inPasso2_15 motores.mtMotor15Passo2();
 instigation inPasso2_16 motores.mtMotor16Passo2();
 end_action
end_rule

rule RlPasso3frente
 condition
 subcondition SubPasso3frente
  premise prPasso3f cn.passo == 3 and
  premise prDirF2cn.direcao == 0
 end_subcondition
 end_condition
 action
 instigation inPasso3_0f motores.mtMotor0Passo3();
 instigation inPasso3_1f motores.mtMotor1Passo3();

 instigation inPasso3_3f motores.mtMotor3Passo3();
 instigation inPasso3_4f motores.mtMotor4Passo3();

 instigation inPasso3_6f motores.mtMotor6Passo3();
 instigation inPasso3_7f motores.mtMotor7Passo3();

 instigation inPasso3_9f motores.mtMotor9Passo3();
 instigation inPasso3_10f motores.mtMotor10Passo3();

 instigation inPasso3_12f motores.mtMotor12Passo3();
 instigation inPasso3_13f motores.mtMotor13Passo3();

 instigation inPasso3_15f motores.mtMotor15Passo3();
 instigation inPasso3_16f motores.mtMotor16Passo3();
 end_action
end_rule

rule RlPasso3esquerda
    condition
 subcondition SubPasso3esquerda
  premise prPasso3e cn.passo == 3 and
  premise prDirE2cn.direcao == 2
 end_subcondition
 end_condition
 action
 instigation inPasso3_0e motores.mtMotor0Passo3Esq();
 instigation inPasso3_1e motores.mtMotor1Passo3Esq();

 instigation inPasso3_3e motores.mtMotor3Passo3Esq();
 instigation inPasso3_4e motores.mtMotor4Passo3Esq();
```

```
 instigation inPasso3_6e motores.mtMotor6Passo3Esq();
 instigation inPasso3_7e motores.mtMotor7Passo3Esq();

 instigation inPasso3_9e motores.mtMotor9Passo3();
 instigation inPasso3_10e motores.mtMotor10Passo3();

 instigation inPasso3_12e motores.mtMotor12Passo3();
 instigation inPasso3_13e motores.mtMotor13Passo3();

 instigation inPasso3_15e motores.mtMotor15Passo3();
 instigation inPasso3_16e motores.mtMotor16Passo3();
 end_action
end_rule

rule RlPasso3direita
 condition
 subcondition SubPasso3direita
  premise prPasso3d cn.passo == 3 and
  premise prDirD2cn.direcao == 1
 end_subcondition
 end_condition
 action
 instigation inPasso3_0d motores.mtMotor0Passo3();
 instigation inPasso3_1d motores.mtMotor1Passo3();

 instigation inPasso3_3d motores.mtMotor3Passo3();
 instigation inPasso3_4d motores.mtMotor4Passo3();

 instigation inPasso3_6d motores.mtMotor6Passo3();
 instigation inPasso3_7d motores.mtMotor7Passo3();

 instigation inPasso3_9d motores.mtMotor9Passo3Dir();
 instigation inPasso3_10d motores.mtMotor10Passo3Dir();

 instigation inPasso3_12d motores.mtMotor12Passo3Dir();
 instigation inPasso3_13d motores.mtMotor13Passo3Dir();

 instigation inPasso3_15d motores.mtMotor15Passo3Dir();
 instigation inPasso3_16d motores.mtMotor16Passo3Dir();
 end_action
end_rule

rule RlZerraPasso
 condition
 subcondition SubZerPasso
  premise prZePs cn.passo == 4
 end_subcondition
 end_condition
 action
 instigation inZerraPasso cn.mtZerraPasso();
```

```
 end_action
end_rule

rule RlZerraTempo
 condition
  subcondition SubZerTmp
   premise prValorTempo cn.tempo == 5
  end_subcondition
 end_condition
 action
  instigation inZerraTempo cn.mtZerraTempo();
  instigation inIncPasso cn.mtIncPasso();
 end_action
end_rule

rule RlZerraContador
 condition
  subcondition SubZerCont
   premise prTxOk1 cn.txempty == true and
   premise prContando1 cn.ativo == true and
   premise prValorContagem cn.contador == 18
  end_subcondition
 end_condition
 action
  instigation inZerraContador cn.mtZerraContador();
  instigation inIncTempo cn.mtIncTempo();
 end_action
end_rule

rule RlIncrementaContador
 condition
  subcondition SubIncCont
   premise prTxOk2 cn.txempty == true and
   premise prContando2 cn.ativo == true and
   premise prValorfinal cn.contador != 18
  end_subcondition
 end_condition
 action
  instigation inIncrementaContador cn.mtIncContador();
 end_action
end_rule

rule RlSaidaSerial0
 condition
  subcondition Sub0
   premise prDefineSaida0 cn.contador == 0
  end_subcondition
 end_condition
 action
  instigation inSeleciona0 cn.mtSeleciona0();
 end_action
```

```
end_rule

rule RlSaidaSerial1
 condition
  subcondition Sub1
   premise prDefineSaida1 cn.contador == 1
  end_subcondition
 end_condition
 action
  instigation inSeleciona1 cn.mtSeleciona1();
 end_action
end_rule

rule RlSaidaSerial2
 condition
  subcondition Sub2
   premise prDefineSaida2 cn.contador == 2
  end_subcondition
 end_condition
 action
  instigation inSeleciona2 cn.mtSeleciona2();
 end_action
end_rule

rule RlSaidaSerial3
 condition
  subcondition Sub3
   premise prDefineSaida3 cn.contador == 3
 end_subcondition
 end_condition
 action
  instigation inSeleciona3 cn.mtSeleciona3();
 end_action
end_rule

rule RlSaidaSerial4
 condition
  subcondition Sub4
   premise prDefineSaida4 cn.contador == 4
  end_subcondition
 end_condition
 action
  instigation inSeleciona4 cn.mtSeleciona4();
 end_action
end_rule

rule RlSaidaSerial5
 condition
  subcondition Sub5
   premise prDefineSaida5 cn.contador == 5
  end_subcondition
```

```
 end_condition
 action
  instigation inSeleciona5 cn.mtSeleciona5();
 end_action
end_rule

rule RlSaidaSerial6
 condition
  subcondition Sub6
   premise prDefineSaida6 cn.contador == 6
  end_subcondition
 end_condition
 action
  instigation inSeleciona6 cn.mtSeleciona6();
 end_action
end_rule

rule RlSaidaSerial7
 condition
  subcondition Sub7
   premise prDefineSaida7 cn.contador == 7
  end_subcondition
 end_condition
 action
  instigation inSeleciona7 cn.mtSeleciona7();
 end_action
end_rule

rule RlSaidaSerial8
 condition
  subcondition Sub8
   premise prDefineSaida8 cn.contador == 8
  end_subcondition
 end_condition
 action
  instigation inSeleciona8 cn.mtSeleciona8();
 end_action
end_rule

rule RlSaidaSerial9
 condition
  subcondition Sub9
   premise prDefineSaida9 cn.contador == 9
  end_subcondition
 end_condition
 action
  instigation inSeleciona9 cn.mtSeleciona9();
 end_action
end_rule

rule RlSaidaSerial10
```

```
condition
 subcondition Sub10
  premise prDefineSaida10 cn.contador == 10
 end_subcondition
end_condition
action
 instigation inSeleciona10 cn.mtSeleciona10();
end_action
end_rule

rule RlSaidaSerial11
condition
 subcondition Sub11
  premise prDefineSaida11 cn.contador == 11
 end_subcondition
end_condition
action
 instigation inSeleciona11 cn.mtSeleciona11();
end_action
end_rule

rule RlSaidaSerial12
    condition
 subcondition Sub12
  premise prDefineSaida12 cn.contador == 12
 end_subcondition
end_condition
action
 instigation inSeleciona12 cn.mtSeleciona12();
end_action
end_rule

rule RlSaidaSerial13
condition
 subcondition Sub13
  premise prDefineSaida13 cn.contador == 13
 end_subcondition
end_condition
action
 instigation inSeleciona13 cn.mtSeleciona13();
end_action
end_rule

rule RlSaidaSerial14
condition
 subcondition Sub14
  premise prDefineSaida14 cn.contador == 14
 end_subcondition
end_condition
action
 instigation inSeleciona14 cn.mtSeleciona14();
```

```
 end_action
end_rule

rule RlSaidaSerial15
 condition
  subcondition Sub15
   premise prDefineSaida15 cn.contador == 15
  end_subcondition
 end_condition
 action
  instigation inSeleciona15 cn.mtSeleciona15();
 end_action
end_rule

rule RlSaidaSerial16
 condition
  subcondition Sub16
   premise prDefineSaida16 cn.contador == 16
  end_subcondition
 end_condition
 action
  instigation inSeleciona16 cn.mtSeleciona16();
 end_action
end_rule

rule RlSaidaSerial17
 condition
  subcondition Sub17
   premise prDefineSaida17 cn.contador == 17
  end_subcondition
 end_condition
 action
  instigation inSeleciona17 cn.mtSeleciona17();
 end_action
end_rule

rule RlSaidaSerial18
 condition
  subcondition Sub18
   premise prDefineSaida18 cn.contador == 18
  end_subcondition
 end_condition
 action
  instigation inSeleciona18 cn.mtSeleciona18();
 end_action
end_rule

main {
 entity controladorHexapod
 in cn.sensorEsq
 in cn.sensorDir
```

```
in cn.liga
in cn.desliga
in cn.txempty
out cn.contador
out cn.dado
}
```

Apêndice S – controladorPID.pon

Código fonte em LingPON-HD 1.0 para o experimento do controlador PID.

```
fbe f_controladorPID
 attributes
  boolean atReset 0
  integer atErro 0
  integer atRealimentacao 0
  integer atSinalControle 0
  integer atErroP 0
  integer atErroIntTmp 0
  integer atErroD 0
  integer atErroI 0
  integer atErroAnterior 0
  integer atKp 0
  integer atKi 0
  integer atKd 0
  integer atTermo_P 0
  integer atTermo_I 0
  integer atTermo_D 0
  integer atTemp 0
  integer atReferencia 0
  integer atSaidaTmp 0
 end_attributes
 methods
  method mtResetSaida(atSinalControle = 0)
  method mtResetErroAnterior(atErroAnterior = 0)
  method mtResetErro(atErro = 0)
  method mtResetErroP(atErroP = 0)
  method mtResetErroIntTmp(atErroIntTmp = 0)
  method mtResetErroD(atErroD = 0)
  method mtResetErroI(atErroI = 0)
  method mtResetTermo_P(atTermo_P = 0)
  method mtResetTermo_I(atTermo_I = 0)
  method mtResetTermo_D(atTermo_D = 0)
  method mtResetTemp(atTemp = 0)
  method mtResetSaidaTmp(atSaidaTmp = 0)
  method mtMaiorMax(atErroIntTmp = 10000)
  method mtMenorMin(atErroIntTmp = -10000)
  method mtSaidaMaiorMax(atSinalControle = 10000)
  method mtSaidaMenorMin(atSinalControle = -10000)
  method mtCalculaErro(atErro = atReferencia - atRealimentacao)
  method mtGravaErroP(atErroP = atErro)
  method mtErroIntTmp(atErroIntTmp = atErroI + atErro)
  method mtErroD(atErroD = atErro - atErroAnterior)
  method mtArmazenaErro(atErroAnterior = atErro)
  method mtCalculaP(atTermo_P = atKp * atErroP)
  method mtCalculaErroI(atErroI = atErroIntTmp)
  method mtCalculaD(atTermo_D = atKd * atErroD)
  method mtCalculaI(atTermo_I = atKi * atErroI)
```

```
 method mtTmp(atTemp = atTermo_P + atTermo_D)
 method mtSaidaTmp(atSaidaTmp = atTemp + atTermo_I)
 method mtSaida(atSinalControle = atSaidaTmp)
end_methods
end_fbe

inst
 f_controladorPID cPID
end_inst

strategy
 no_one
end_strategy

// verifica a condição de reset
rule RlReset
 condition
  subcondition SubReset
   premise prReset cPID.atReset == true
  end_subcondition
 end_condition
 action
  instigation inResetSaida cPID.mtResetSaida();
  instigation inResetErroAnterior cPID.mtResetErroAnterior();
  instigation inResetErro cPID.mtResetErro();
  instigation inResetErroP cPID.mtResetErroP();
  instigation inResetErroIntTmp cPID.mtResetErroIntTmp();
  instigation inResetErroD cPID.mtResetErroD();
  instigation inResetErroI cPID.mtResetErroI();
  instigation inResetTermo_P cPID.mtResetTermo_P();
  instigation inResetTermo_I cPID.mtResetTermo_I();
  instigation inResetTermo_D cPID.mtResetTermo_D();
  instigation inResetTemp cPID.mtResetTemp();
  instigation inResetSaidaTmp cPID.mtResetSaidaTmp();
 end_action
end_rule

//verifica limites do integrador
rule RlMaiorMax
 condition
  subcondition SubMaiorMax
   premise prMaiorMax cPID.atErroIntTmp > 10000
  end_subcondition
 end_condition
 action
  instigation inMaiorMax cPID.mtMaiorMax();
 end_action
end_rule

rule RlMenorMin
 condition
```

```
 subcondition SubMenorMin
  premise prMenorMin cPID.atErroIntTmp < -10000
 end_subcondition
end_condition
action
 instigation inMenorMin cPID.mtMenorMin();
end_action
end_rule

// Verifica os limites da saída
rule RlSaidaMaiorMax
condition
 subcondition SubSaidaMaiorMax
  premise prSaidaMaiorMax cPID.atSaidaTmp > 10000
 end_subcondition
end_condition
action
 instigation inSaidaMaiorMax cPID.mtSaidaMaiorMax();
end_action
end_rule

rule RlSaidaMenorMin
condition
 subcondition SubSaidaMenorMin
  premise prSaidaMenorMin cPID.atSaidaTmp < -10000
 end_subcondition
end_condition
action
 instigation inSaidaMenorMin cPID.mtSaidaMenorMin();
end_action
end_rule

rule RlCalculaPID
condition
 subcondition SubCalculaPID
  premise prResetI cPID.atReset == false
 end_subcondition
end_condition
action
 instigation inCalculaErro cPID.mtCalculaErro();
 instigation inGravaErroP cPID.mtGravaErroP();
 instigation inErroIntTmp cPID.mtErroIntTmp();
 instigation inCalculaErroD cPID.mtErroD();
 instigation inArmazenaErro cPID.mtArmazenaErro();
 instigation inCalculaP cPID.mtCalculaP();
 instigation inCalculaErroI cPID.mtCalculaErroI();
 instigation inCalculaD cPID.mtCalculaD();
 instigation inCalculaI cPID.mtCalculaI();
 instigation inTmp cPID.mtTmp();
 instigation inSaidaTmp cPID.mtSaidaTmp();
 instigation inSaida cPID.mtSaida();
```

```
 end_action
end_rule

main {
 entity controladorPID

 in cPID.atReset
 in cPID.atReferencia
 in cPID.atRealimentacao
 in cPID.atKp
 in cPID.atKi
 in cPID.atKd

 out cPID.atSinalControle

 NBits cPID.atErroI 16
 NBits cPID.atErroAnterior 16
 NBits cPID.atErroD 16
 NBits cPID.atErro 16
 NBits cPID.atTermo_P 16
 NBits cPID.atTermo_I 16
 NBits cPID.atTermo_D 16
 NBits cPID.atReferencia 16
 NBits cPID.atRealimentacao 16
 NBits cPID.atSinalControle 16
 NBits cPID.atKp 16
 NBits cPID.atKi 16
 NBits cPID.atKd 16
 NBits cPID.atSaidaTmp 16
 NBits cPID.atErroP 16
 NBits cPID.atErroIntTmp 16
 NBits cPID.atTemp 16
 }
```

Apêndice T – controladorPID.vhd

Código fonte em VHDL para o experimento do controlador PID gerado automaticamente pelo Compilador PON-HD 1.0.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.data_type_pkg.all;

ENTITY controladorPID IS
PORT(
 in_cPID_atReset :IN STD_LOGIC;
 in_set_cPID_atReset :IN STD_LOGIC;
 in_cPID_atReferencia :IN STD_LOGIC_VECTOR(16-1 downto 0);
 in_set_cPID_atReferencia :IN STD_LOGIC;
 in_cPID_atRealimentacao :IN STD_LOGIC_VECTOR(16-1 downto 0);
 in_set_cPID_atRealimentacao :IN STD_LOGIC;
 in_cPID_atKp :IN STD_LOGIC_VECTOR(16-1 downto 0);
 in_set_cPID_atKp :IN STD_LOGIC;
 in_cPID_atKi :IN STD_LOGIC_VECTOR(16-1 downto 0);
 in_set_cPID_atKi :IN STD_LOGIC;
 in_cPID_atKd :IN STD_LOGIC_VECTOR(16-1 downto 0);
 in_set_cPID_atKd :IN STD_LOGIC;
 out_cPID_atSinalControle :OUT STD_LOGIC_VECTOR(16-1 downto 0);
 clock :IN STD_LOGIC
 );
END controladorPID;

ARCHITECTURE controladorPID_arq OF controladorPID IS
----------------- COMPONENTS ---------------------
COMPONENT NOP_attribute
 GENERIC (
 N_bits: INTEGER;
 N_new_values: INTEGER;
 initial_value: STD_LOGIC_VECTOR
 );
 PORT(
 att_clock :IN STD_LOGIC;
 att_new_value :IN data(0 TO N_new_values-1)(N_bits-1 downto
0);
 att_set_value :IN STD_LOGIC_VECTOR(N_new_values-1 downto 0);
 att_value :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0)
 );
END COMPONENT;
COMPONENT NOP_premise
 GENERIC (
 N_bits: INTEGER:=1;
 dataType: INTEGER:=0;
 operation: INTEGER:=1 --EQUAL 1; DIFFERENT 2; LESS_THAN 3;
GREATER_THAN 4; LESS_EQUAL 5; GREATER_EQUAL 6;
 );
```

```
 PORT(
  att_value_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
  att_value_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
  pre_result :OUT STD_LOGIC
  );
END COMPONENT;
COMPONENT NOP_method
 GENERIC (
   N_bits: INTEGER:=1;
   dataType: INTEGER:=1;
 operation: INTEGER:=1
 );
 PORT(
  in_a :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
  in_b :IN STD_LOGIC_VECTOR(N_bits-1 downto 0);
  execute :IN STD_LOGIC;
  result :OUT STD_LOGIC_VECTOR(N_bits-1 downto 0);
  notify :OUT STD_LOGIC
 );
END COMPONENT;
----------------- SIGNALS ---------------------
 SIGNAL cPID_atReset_new_value: data(0 TO 1-1)(0 downto 0);
 SIGNAL cPID_atErro_new_value: data(0 TO 2-1)( 16-1 downto 0);
 SIGNAL cPID_atRealimentacao_new_value: data(0 TO 1-1)(16-1
downto 0);
 SIGNAL cPID_atSinalControle_new_value: data(0 TO 4-1)(16-1
downto 0);
 SIGNAL cPID_atErroP_new_value: data(0 TO 2-1)(16-1 downto 0);
 SIGNAL cPID_atErroIntTmp_new_value: data(0 TO 4-1)(16-1 downto
0);
 SIGNAL cPID_atErroD_new_value: data(0 TO 2-1)(16-1 downto 0);
 SIGNAL cPID_atErroI_new_value: data(0 TO 2-1)(16-1 downto 0);
 SIGNAL cPID_atErroAnterior_new_value: data(0 TO 2-1)(16-1
downto 0);
 SIGNAL cPID_atKp_new_value: data(0 TO 1-1)(16-1 downto 0);
 SIGNAL cPID_atKi_new_value: data(0 TO 1-1)(16-1 downto 0);
 SIGNAL cPID_atKd_new_value: data(0 TO 1-1)(16-1 downto 0);
 SIGNAL cPID_atTermo_P_new_value: data(0 TO 2-1)(16-1 downto
0);
 SIGNAL cPID_atTermo_I_new_value: data(0 TO 2-1)(16-1 downto
0);
 SIGNAL cPID_atTermo_D_new_value: data(0 TO 2-1)(16-1 downto
0);
 SIGNAL cPID_atTemp_new_value: data(0 TO 2-1)(16-1 downto 0);
 SIGNAL cPID_atReferencia_new_value: data(0 TO 1-1)(16-1 downto
0);
 SIGNAL cPID_atSaidaTmp_new_value: data(0 TO 2-1)(16-1 downto
0);

 SIGNAL cPID_atReset_value: STD_LOGIC_VECTOR(0 downto 0);
 SIGNAL cPID_atErro_value: STD_LOGIC_VECTOR( 16-1 downto 0);
```

```
 SIGNAL cPID_atRealimentacao_value: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_atSinalControle_value: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_atErroP_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_atErroIntTmp_value: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_atErroD_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_atErroI_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_atErroAnterior_value: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_atKp_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_atKi_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_atKd_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_atTermo_P_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_atTermo_I_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_atTermo_D_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_atTemp_value: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_atReferencia_value: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_atSaidaTmp_value: STD_LOGIC_VECTOR(16-1 downto 0);

 SIGNAL cPID_atReset_set: STD_LOGIC_VECTOR(1-1 downto 0);
 SIGNAL cPID_atErro_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL cPID_atRealimentacao_set: STD_LOGIC_VECTOR(1-1 downto
0);
 SIGNAL cPID_atSinalControle_set: STD_LOGIC_VECTOR(4-1 downto
0);
 SIGNAL cPID_atErroP_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL cPID_atErroIntTmp_set: STD_LOGIC_VECTOR(4-1 downto 0);
 SIGNAL cPID_atErroD_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL cPID_atErroI_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL cPID_atErroAnterior_set: STD_LOGIC_VECTOR(2-1 downto
0);
 SIGNAL cPID_atKp_set: STD_LOGIC_VECTOR(1-1 downto 0);
 SIGNAL cPID_atKi_set: STD_LOGIC_VECTOR(1-1 downto 0);
 SIGNAL cPID_atKd_set: STD_LOGIC_VECTOR(1-1 downto 0);
 SIGNAL cPID_atTermo_P_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL cPID_atTermo_I_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL cPID_atTermo_D_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL cPID_atTemp_set: STD_LOGIC_VECTOR(2-1 downto 0);
 SIGNAL cPID_atReferencia_set: STD_LOGIC_VECTOR(1-1 downto 0);
 SIGNAL cPID_atSaidaTmp_set: STD_LOGIC_VECTOR(2-1 downto 0);

 SIGNAL prMaiorMax_result: STD_LOGIC;
 SIGNAL prMenorMin_result: STD_LOGIC;
 SIGNAL prReset_result: STD_LOGIC;
 SIGNAL prResetI_result: STD_LOGIC;
 SIGNAL prSaidaMaiorMax_result: STD_LOGIC;
 SIGNAL prSaidaMenorMin_result: STD_LOGIC;
```

```
 SIGNAL prMaiorMax_imput_a: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL prMenorMin_imput_a: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL prReset_imput_a: STD_LOGIC_VECTOR(0 downto 0);
 SIGNAL prResetI_imput_a: STD_LOGIC_VECTOR(0 downto 0);
 SIGNAL prSaidaMaiorMax_imput_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL prSaidaMenorMin_imput_a: STD_LOGIC_VECTOR(16-1 downto
0);

 SIGNAL prMaiorMax_imput_b: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL prMenorMin_imput_b: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL prReset_imput_b: STD_LOGIC_VECTOR(0 downto 0);
 SIGNAL prResetI_imput_b: STD_LOGIC_VECTOR(0 downto 0);
 SIGNAL prSaidaMaiorMax_imput_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL prSaidaMenorMin_imput_b: STD_LOGIC_VECTOR(16-1 downto
0);

 SIGNAL cPID_mtResetSaida_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetErroAnterior_input_a: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetErro_input_a: STD_LOGIC_VECTOR( 16-1 downto
0);
 SIGNAL cPID_mtResetErroP_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetErroIntTmp_input_a: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetErroD_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetErroI_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetTermo_P_input_a: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetTermo_I_input_a: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetTermo_D_input_a: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetTemp_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetSaidaTmp_input_a: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtMaiorMax_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtMenorMin_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtSaidaMaiorMax_input_a: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtSaidaMenorMin_input_a: STD_LOGIC_VECTOR(16-1
downto 0);
```

```
 SIGNAL cPID_mtCalculaErro_input_a: STD_LOGIC_VECTOR( 16-1
downto 0);
 SIGNAL cPID_mtGravaErroP_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtErroIntTmp_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtErroD_input_a: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_mtArmazenaErro_input_a: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtCalculaP_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtCalculaErroI_input_a: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtCalculaD_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtCalculaI_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtTmp_input_a: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_mtSaidaTmp_input_a: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtSaida_input_a: STD_LOGIC_VECTOR(16-1 downto 0);

 SIGNAL cPID_mtResetSaida_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetErroAnterior_input_b: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetErro_input_b: STD_LOGIC_VECTOR( 16-1 downto
0);
 SIGNAL cPID_mtResetErroP_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetErroIntTmp_input_b: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetErroD_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetErroI_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetTermo_P_input_b: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetTermo_I_input_b: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetTermo_D_input_b: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetTemp_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetSaidaTmp_input_b: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtMaiorMax_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtMenorMin_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
```

```
 SIGNAL cPID_mtSaidaMaiorMax_input_b: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtSaidaMenorMin_input_b: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtCalculaErro_input_b: STD_LOGIC_VECTOR( 16-1
downto 0);
 SIGNAL cPID_mtGravaErroP_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtErroIntTmp_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtErroD_input_b: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_mtArmazenaErro_input_b: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtCalculaP_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtCalculaErroI_input_b: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtCalculaD_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtCalculaI_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtTmp_input_b: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_mtSaidaTmp_input_b: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtSaida_input_b: STD_LOGIC_VECTOR(16-1 downto 0);

 SIGNAL cPID_mtResetSaida_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetErroAnterior_output: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetErro_output: STD_LOGIC_VECTOR( 16-1 downto
0);
 SIGNAL cPID_mtResetErroP_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetErroIntTmp_output: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetErroD_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetErroI_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetTermo_P_output: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetTermo_I_output: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetTermo_D_output: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtResetTemp_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtResetSaidaTmp_output: STD_LOGIC_VECTOR(16-1
downto 0);
```

```
 SIGNAL cPID_mtMaiorMax_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtMenorMin_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtSaidaMaiorMax_output: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtSaidaMenorMin_output: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtCalculaErro_output: STD_LOGIC_VECTOR( 16-1
downto 0);
 SIGNAL cPID_mtGravaErroP_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtErroIntTmp_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtErroD_output: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_mtArmazenaErro_output: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtCalculaP_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtCalculaErroI_output: STD_LOGIC_VECTOR(16-1
downto 0);
 SIGNAL cPID_mtCalculaD_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtCalculaI_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtTmp_output: STD_LOGIC_VECTOR(16-1 downto 0);
 SIGNAL cPID_mtSaidaTmp_output: STD_LOGIC_VECTOR(16-1 downto
0);
 SIGNAL cPID_mtSaida_output: STD_LOGIC_VECTOR(16-1 downto 0);

 SIGNAL cPID_mtResetSaida_execute: STD_LOGIC;
 SIGNAL cPID_mtResetErroAnterior_execute: STD_LOGIC;
 SIGNAL cPID_mtResetErro_execute: STD_LOGIC;
 SIGNAL cPID_mtResetErroP_execute: STD_LOGIC;
 SIGNAL cPID_mtResetErroIntTmp_execute: STD_LOGIC;
 SIGNAL cPID_mtResetErroD_execute: STD_LOGIC;
 SIGNAL cPID_mtResetErroI_execute: STD_LOGIC;
 SIGNAL cPID_mtResetTermo_P_execute: STD_LOGIC;
 SIGNAL cPID_mtResetTermo_I_execute: STD_LOGIC;
 SIGNAL cPID_mtResetTermo_D_execute: STD_LOGIC;
 SIGNAL cPID_mtResetTemp_execute: STD_LOGIC;
 SIGNAL cPID_mtResetSaidaTmp_execute: STD_LOGIC;
 SIGNAL cPID_mtMaiorMax_execute: STD_LOGIC;
 SIGNAL cPID_mtMenorMin_execute: STD_LOGIC;
 SIGNAL cPID_mtSaidaMaiorMax_execute: STD_LOGIC;
 SIGNAL cPID_mtSaidaMenorMin_execute: STD_LOGIC;
 SIGNAL cPID_mtCalculaErro_execute: STD_LOGIC;
 SIGNAL cPID_mtGravaErroP_execute: STD_LOGIC;
 SIGNAL cPID_mtErroIntTmp_execute: STD_LOGIC;
 SIGNAL cPID_mtErroD_execute: STD_LOGIC;
 SIGNAL cPID_mtArmazenaErro_execute: STD_LOGIC;
```

```
SIGNAL cPID_mtCalculaP_execute: STD_LOGIC;
SIGNAL cPID_mtCalculaErroI_execute: STD_LOGIC;
SIGNAL cPID_mtCalculaD_execute: STD_LOGIC;
SIGNAL cPID_mtCalculaI_execute: STD_LOGIC;
SIGNAL cPID_mtTmp_execute: STD_LOGIC;
SIGNAL cPID_mtSaidaTmp_execute: STD_LOGIC;
SIGNAL cPID_mtSaida_execute: STD_LOGIC;

SIGNAL cPID_mtResetSaida_notify: STD_LOGIC;
SIGNAL cPID_mtResetErroAnterior_notify: STD_LOGIC;
SIGNAL cPID_mtResetErro_notify: STD_LOGIC;
SIGNAL cPID_mtResetErroP_notify: STD_LOGIC;
SIGNAL cPID_mtResetErroIntTmp_notify: STD_LOGIC;
SIGNAL cPID_mtResetErroD_notify: STD_LOGIC;
SIGNAL cPID_mtResetErroI_notify: STD_LOGIC;
SIGNAL cPID_mtResetTermo_P_notify: STD_LOGIC;
SIGNAL cPID_mtResetTermo_I_notify: STD_LOGIC;
SIGNAL cPID_mtResetTermo_D_notify: STD_LOGIC;
SIGNAL cPID_mtResetTemp_notify: STD_LOGIC;
SIGNAL cPID_mtResetSaidaTmp_notify: STD_LOGIC;
SIGNAL cPID_mtMaiorMax_notify: STD_LOGIC;
SIGNAL cPID_mtMenorMin_notify: STD_LOGIC;
SIGNAL cPID_mtSaidaMaiorMax_notify: STD_LOGIC;
SIGNAL cPID_mtSaidaMenorMin_notify: STD_LOGIC;
SIGNAL cPID_mtCalculaErro_notify: STD_LOGIC;
SIGNAL cPID_mtGravaErroP_notify: STD_LOGIC;
SIGNAL cPID_mtErroIntTmp_notify: STD_LOGIC;
SIGNAL cPID_mtErroD_notify: STD_LOGIC;
SIGNAL cPID_mtArmazenaErro_notify: STD_LOGIC;
SIGNAL cPID_mtCalculaP_notify: STD_LOGIC;
SIGNAL cPID_mtCalculaErroI_notify: STD_LOGIC;
SIGNAL cPID_mtCalculaD_notify: STD_LOGIC;
SIGNAL cPID_mtCalculaI_notify: STD_LOGIC;
SIGNAL cPID_mtTmp_notify: STD_LOGIC;
SIGNAL cPID_mtSaidaTmp_notify: STD_LOGIC;
SIGNAL cPID_mtSaida_notify: STD_LOGIC;
-----------------------------------------------
BEGIN
---------- PON ELEMENTS INSTANCES -------------
cPID_atReset: NOP_attribute GENERIC MAP(N_bits => 1,
N_new_values => 1, initial_value => "0") PORT MAP(att_clock =>
clock, att_new_value => cPID_atReset_new_value, att_set_value
=> cPID_atReset_set, att_value => cPID_atReset_value);
cPID_atErro: NOP_attribute GENERIC MAP(N_bits =>  16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atErro_new_value, att_set_value => cPID_atErro_set,
att_value => cPID_atErro_value);
cPID_atRealimentacao: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 1, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
```

```
cPID_atRealimentacao_new_value, att_set_value =>
cPID_atRealimentacao_set, att_value =>
cPID_atRealimentacao_value);
 cPID_atSinalControle: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 4, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atSinalControle_new_value, att_set_value =>
cPID_atSinalControle_set, att_value =>
cPID_atSinalControle_value);
 cPID_atErroP: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atErroP_new_value, att_set_value => cPID_atErroP_set,
att_value => cPID_atErroP_value);
 cPID_atErroIntTmp: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 4, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atErroIntTmp_new_value, att_set_value =>
cPID_atErroIntTmp_set, att_value => cPID_atErroIntTmp_value);
 cPID_atErroD: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atErroD_new_value, att_set_value => cPID_atErroD_set,
att_value => cPID_atErroD_value);
 cPID_atErroI: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atErroI_new_value, att_set_value => cPID_atErroI_set,
att_value => cPID_atErroI_value);
 cPID_atErroAnterior: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atErroAnterior_new_value, att_set_value =>
cPID_atErroAnterior_set, att_value =>
cPID_atErroAnterior_value);
 cPID_atKp: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 1, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value => cPID_atKp_new_value,
att_set_value => cPID_atKp_set, att_value => cPID_atKp_value);
 cPID_atKi: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 1, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value => cPID_atKi_new_value,
att_set_value => cPID_atKi_set, att_value => cPID_atKi_value);
 cPID_atKd: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 1, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value => cPID_atKd_new_value,
att_set_value => cPID_atKd_set, att_value => cPID_atKd_value);
 cPID_atTermo_P: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
```

```
cPID_atTermo_P_new_value, att_set_value => cPID_atTermo_P_set,
att_value => cPID_atTermo_P_value);
 cPID_atTermo_I: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atTermo_I_new_value, att_set_value => cPID_atTermo_I_set,
att_value => cPID_atTermo_I_value);
 cPID_atTermo_D: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atTermo_D_new_value, att_set_value => cPID_atTermo_D_set,
att_value => cPID_atTermo_D_value);
 cPID_atTemp: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atTemp_new_value, att_set_value => cPID_atTemp_set,
att_value => cPID_atTemp_value);
 cPID_atReferencia: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 1, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atReferencia_new_value, att_set_value =>
cPID_atReferencia_set, att_value => cPID_atReferencia_value);
 cPID_atSaidaTmp: NOP_attribute GENERIC MAP(N_bits => 16,
N_new_values => 2, initial_value => "0000000000000000") PORT
MAP(att_clock => clock, att_new_value =>
cPID_atSaidaTmp_new_value, att_set_value =>
cPID_atSaidaTmp_set, att_value => cPID_atSaidaTmp_value);

 prMaiorMax_GREATER: NOP_premise GENERIC MAP(N_bits => 16,
dataType => 0, operation => 4) PORT MAP(att_value_a =>
prMaiorMax_imput_a, att_value_b => prMaiorMax_imput_b,
pre_result => prMaiorMax_result);
 prMenorMin_LESS: NOP_premise GENERIC MAP(N_bits => 16,
dataType => 0, operation => 3) PORT MAP(att_value_a =>
prMenorMin_imput_a, att_value_b => prMenorMin_imput_b,
pre_result => prMenorMin_result);
 prReset_EQUAL: NOP_premise GENERIC MAP(N_bits => 1, dataType
=> 0, operation => 1) PORT MAP(att_value_a => prReset_imput_a,
att_value_b => prReset_imput_b, pre_result => prReset_result);
 prResetI_EQUAL: NOP_premise GENERIC MAP(N_bits => 1, dataType
=> 0, operation => 1) PORT MAP(att_value_a =>
prResetI_imput_a, att_value_b => prResetI_imput_b, pre_result
=> prResetI_result);
 prSaidaMaiorMax_GREATER: NOP_premise GENERIC MAP(N_bits => 16,
dataType => 0, operation => 4) PORT MAP(att_value_a =>
prSaidaMaiorMax_imput_a, att_value_b =>
prSaidaMaiorMax_imput_b, pre_result =>
prSaidaMaiorMax_result);
 prSaidaMenorMin_LESS: NOP_premise GENERIC MAP(N_bits => 16,
dataType => 0, operation => 3) PORT MAP(att_value_a =>
prSaidaMenorMin_imput_a, att_value_b =>
```

```
prSaidaMenorMin_imput_b, pre_result =>
prSaidaMenorMin_result);

 cPID_mtResetSaida: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetSaida_input_a, in_b => cPID_mtResetSaida_input_b,
execute => cPID_mtResetSaida_execute, result =>
cPID_mtResetSaida_output, notify => cPID_mtResetSaida_notify);
 cPID_mtResetErroAnterior: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetErroAnterior_input_a, in_b =>
cPID_mtResetErroAnterior_input_b, execute =>
cPID_mtResetErroAnterior_execute, result =>
cPID_mtResetErroAnterior_output, notify =>
cPID_mtResetErroAnterior_notify);
 cPID_mtResetErro: NOP_method GENERIC MAP(N_bits =>  16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetErro_input_a, in_b => cPID_mtResetErro_input_b,
execute => cPID_mtResetErro_execute, result =>
cPID_mtResetErro_output, notify => cPID_mtResetErro_notify);
 cPID_mtResetErroP: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetErroP_input_a, in_b => cPID_mtResetErroP_input_b,
execute => cPID_mtResetErroP_execute, result =>
cPID_mtResetErroP_output, notify => cPID_mtResetErroP_notify);
 cPID_mtResetErroIntTmp: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetErroIntTmp_input_a, in_b =>
cPID_mtResetErroIntTmp_input_b, execute =>
cPID_mtResetErroIntTmp_execute, result =>
cPID_mtResetErroIntTmp_output, notify =>
cPID_mtResetErroIntTmp_notify);
 cPID_mtResetErroD: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetErroD_input_a, in_b => cPID_mtResetErroD_input_b,
execute => cPID_mtResetErroD_execute, result =>
cPID_mtResetErroD_output, notify => cPID_mtResetErroD_notify);
 cPID_mtResetErroI: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetErroI_input_a, in_b => cPID_mtResetErroI_input_b,
execute => cPID_mtResetErroI_execute, result =>
cPID_mtResetErroI_output, notify => cPID_mtResetErroI_notify);
 cPID_mtResetTermo_P: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetTermo_P_input_a, in_b =>
cPID_mtResetTermo_P_input_b, execute =>
cPID_mtResetTermo_P_execute, result =>
cPID_mtResetTermo_P_output, notify =>
cPID_mtResetTermo_P_notify);
 cPID_mtResetTermo_I: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
```

```
cPID_mtResetTermo_I_input_a, in_b =>
cPID_mtResetTermo_I_input_b, execute =>
cPID_mtResetTermo_I_execute, result =>
cPID_mtResetTermo_I_output, notify =>
cPID_mtResetTermo_I_notify);
 cPID_mtResetTermo_D: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetTermo_D_input_a, in_b =>
cPID_mtResetTermo_D_input_b, execute =>
cPID_mtResetTermo_D_execute, result =>
cPID_mtResetTermo_D_output, notify =>
cPID_mtResetTermo_D_notify);
 cPID_mtResetTemp: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetTemp_input_a, in_b => cPID_mtResetTemp_input_b,
execute => cPID_mtResetTemp_execute, result =>
cPID_mtResetTemp_output, notify => cPID_mtResetTemp_notify);
 cPID_mtResetSaidaTmp: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtResetSaidaTmp_input_a, in_b =>
cPID_mtResetSaidaTmp_input_b, execute =>
cPID_mtResetSaidaTmp_execute, result =>
cPID_mtResetSaidaTmp_output, notify =>
cPID_mtResetSaidaTmp_notify);
 cPID_mtMaiorMax: NOP_method GENERIC MAP(N_bits => 16, dataType
=> 1, operation => 0) PORT MAP(in_a =>
cPID_mtMaiorMax_input_a, in_b => cPID_mtMaiorMax_input_b,
execute => cPID_mtMaiorMax_execute, result =>
cPID_mtMaiorMax_output, notify => cPID_mtMaiorMax_notify);
 cPID_mtMenorMin: NOP_method GENERIC MAP(N_bits => 16, dataType
=> 1, operation => 0) PORT MAP(in_a =>
cPID_mtMenorMin_input_a, in_b => cPID_mtMenorMin_input_b,
execute => cPID_mtMenorMin_execute, result =>
cPID_mtMenorMin_output, notify => cPID_mtMenorMin_notify);
 cPID_mtSaidaMaiorMax: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtSaidaMaiorMax_input_a, in_b =>
cPID_mtSaidaMaiorMax_input_b, execute =>
cPID_mtSaidaMaiorMax_execute, result =>
cPID_mtSaidaMaiorMax_output, notify =>
cPID_mtSaidaMaiorMax_notify);
 cPID_mtSaidaMenorMin: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtSaidaMenorMin_input_a, in_b =>
cPID_mtSaidaMenorMin_input_b, execute =>
cPID_mtSaidaMenorMin_execute, result =>
cPID_mtSaidaMenorMin_output, notify =>
cPID_mtSaidaMenorMin_notify);
 cPID_mtCalculaErro: NOP_method GENERIC MAP(N_bits =>  16,
dataType => 1, operation => 2) PORT MAP(in_a =>
cPID_mtCalculaErro_input_a, in_b =>
```

```
cPID_mtCalculaErro_input_b, execute =>
cPID_mtCalculaErro_execute, result =>
cPID_mtCalculaErro_output, notify =>
cPID_mtCalculaErro_notify);
 cPID_mtGravaErroP: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtGravaErroP_input_a, in_b => cPID_mtGravaErroP_input_b,
execute => cPID_mtGravaErroP_execute, result =>
cPID_mtGravaErroP_output, notify => cPID_mtGravaErroP_notify);
 cPID_mtErroIntTmp: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 1) PORT MAP(in_a =>
cPID_mtErroIntTmp_input_a, in_b => cPID_mtErroIntTmp_input_b,
execute => cPID_mtErroIntTmp_execute, result =>
cPID_mtErroIntTmp_output, notify => cPID_mtErroIntTmp_notify);
 cPID_mtErroD: NOP_method GENERIC MAP(N_bits => 16, dataType =>
1, operation => 2) PORT MAP(in_a => cPID_mtErroD_input_a, in_b
=> cPID_mtErroD_input_b, execute => cPID_mtErroD_execute,
result => cPID_mtErroD_output, notify => cPID_mtErroD_notify);
 cPID_mtArmazenaErro: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtArmazenaErro_input_a, in_b =>
cPID_mtArmazenaErro_input_b, execute =>
cPID_mtArmazenaErro_execute, result =>
cPID_mtArmazenaErro_output, notify =>
cPID_mtArmazenaErro_notify);
 cPID_mtCalculaP: NOP_method GENERIC MAP(N_bits => 16, dataType
=> 1, operation => 3) PORT MAP(in_a =>
cPID_mtCalculaP_input_a, in_b => cPID_mtCalculaP_input_b,
execute => cPID_mtCalculaP_execute, result =>
cPID_mtCalculaP_output, notify => cPID_mtCalculaP_notify);
 cPID_mtCalculaErroI: NOP_method GENERIC MAP(N_bits => 16,
dataType => 1, operation => 0) PORT MAP(in_a =>
cPID_mtCalculaErroI_input_a, in_b =>
cPID_mtCalculaErroI_input_b, execute =>
cPID_mtCalculaErroI_execute, result =>
cPID_mtCalculaErroI_output, notify =>
cPID_mtCalculaErroI_notify);
 cPID_mtCalculaD: NOP_method GENERIC MAP(N_bits => 16, dataType
=> 1, operation => 3) PORT MAP(in_a =>
cPID_mtCalculaD_input_a, in_b => cPID_mtCalculaD_input_b,
execute => cPID_mtCalculaD_execute, result =>
cPID_mtCalculaD_output, notify => cPID_mtCalculaD_notify);
 cPID_mtCalculaI: NOP_method GENERIC MAP(N_bits => 16, dataType
=> 1, operation => 3) PORT MAP(in_a =>
cPID_mtCalculaI_input_a, in_b => cPID_mtCalculaI_input_b,
execute => cPID_mtCalculaI_execute, result =>
cPID_mtCalculaI_output, notify => cPID_mtCalculaI_notify);
 cPID_mtTmp: NOP_method GENERIC MAP(N_bits => 16, dataType =>
1, operation => 1) PORT MAP(in_a => cPID_mtTmp_input_a, in_b
=> cPID_mtTmp_input_b, execute => cPID_mtTmp_execute, result
=> cPID_mtTmp_output, notify => cPID_mtTmp_notify);
```

```
cPID_mtSaidaTmp: NOP_method GENERIC MAP(N_bits => 16, dataType
=> 1, operation => 1) PORT MAP(in_a =>
cPID_mtSaidaTmp_input_a, in_b => cPID_mtSaidaTmp_input_b,
execute => cPID_mtSaidaTmp_execute, result =>
cPID_mtSaidaTmp_output, notify => cPID_mtSaidaTmp_notify);
cPID_mtSaida: NOP_method GENERIC MAP(N_bits => 16, dataType =>
1, operation => 0) PORT MAP(in_a => cPID_mtSaida_input_a, in_b
=> cPID_mtSaida_input_b, execute => cPID_mtSaida_execute,
result => cPID_mtSaida_output, notify => cPID_mtSaida_notify);
---------- PON ELEMENTS CONNECTIONS -------------
---------- attributes inputs -------------
cPID_atReset_new_value(0)(0) <= in_cPID_atReset;
cPID_atReset_set(0) <= in_set_cPID_atReset;
cPID_atErro_new_value(0) <= cPID_mtResetErro_output;
cPID_atErro_set(0) <= cPID_mtResetErro_notify;
cPID_atErro_new_value(1) <= cPID_mtCalculaErro_output;
cPID_atErro_set(1) <= cPID_mtCalculaErro_notify;
cPID_atRealimentacao_new_value(0) <= in_cPID_atRealimentacao;
cPID_atRealimentacao_set(0) <= in_set_cPID_atRealimentacao;
cPID_atSinalControle_new_value(0) <= cPID_mtResetSaida_output;
cPID_atSinalControle_set(0) <= cPID_mtResetSaida_notify;
cPID_atSinalControle_new_value(1) <=
cPID_mtSaidaMaiorMax_output;
cPID_atSinalControle_set(1) <= cPID_mtSaidaMaiorMax_notify;
cPID_atSinalControle_new_value(2) <=
cPID_mtSaidaMenorMin_output;
cPID_atSinalControle_set(2) <= cPID_mtSaidaMenorMin_notify;
cPID_atSinalControle_new_value(3) <= cPID_mtSaida_output;
cPID_atSinalControle_set(3) <= cPID_mtSaida_notify;
cPID_atErroP_new_value(0) <= cPID_mtResetErroP_output;
cPID_atErroP_set(0) <= cPID_mtResetErroP_notify;
cPID_atErroP_new_value(1) <= cPID_mtGravaErroP_output;
cPID_atErroP_set(1) <= cPID_mtGravaErroP_notify;
cPID_atErroIntTmp_new_value(0) <=
cPID_mtResetErroIntTmp_output;
cPID_atErroIntTmp_set(0) <= cPID_mtResetErroIntTmp_notify;
cPID_atErroIntTmp_new_value(1) <= cPID_mtMaiorMax_output;
cPID_atErroIntTmp_set(1) <= cPID_mtMaiorMax_notify;
cPID_atErroIntTmp_new_value(2) <= cPID_mtMenorMin_output;
cPID_atErroIntTmp_set(2) <= cPID_mtMenorMin_notify;
cPID_atErroIntTmp_new_value(3) <= cPID_mtErroIntTmp_output;
cPID_atErroIntTmp_set(3) <= cPID_mtErroIntTmp_notify;
cPID_atErroD_new_value(0) <= cPID_mtResetErroD_output;
cPID_atErroD_set(0) <= cPID_mtResetErroD_notify;
cPID_atErroD_new_value(1) <= cPID_mtErroD_output;
cPID_atErroD_set(1) <= cPID_mtErroD_notify;
cPID_atErroI_new_value(0) <= cPID_mtResetErroI_output;
cPID_atErroI_set(0) <= cPID_mtResetErroI_notify;
cPID_atErroI_new_value(1) <= cPID_mtCalculaErroI_output;
cPID_atErroI_set(1) <= cPID_mtCalculaErroI_notify;
```

```
cPID_atErroAnterior_new_value(0) <=
cPID_mtResetErroAnterior_output;
cPID_atErroAnterior_set(0) <= cPID_mtResetErroAnterior_notify;
cPID_atErroAnterior_new_value(1) <=
cPID_mtArmazenaErro_output;
cPID_atErroAnterior_set(1) <= cPID_mtArmazenaErro_notify;
cPID_atKp_new_value(0) <= in_cPID_atKp;
cPID_atKp_set(0) <= in_set_cPID_atKp;
cPID_atKi_new_value(0) <= in_cPID_atKi;
cPID_atKi_set(0) <= in_set_cPID_atKi;
cPID_atKd_new_value(0) <= in_cPID_atKd;
cPID_atKd_set(0) <= in_set_cPID_atKd;
cPID_atTermo_P_new_value(0) <= cPID_mtResetTermo_P_output;
cPID_atTermo_P_set(0) <= cPID_mtResetTermo_P_notify;
cPID_atTermo_P_new_value(1) <= cPID_mtCalculaP_output;
cPID_atTermo_P_set(1) <= cPID_mtCalculaP_notify;
cPID_atTermo_I_new_value(0) <= cPID_mtResetTermo_I_output;
cPID_atTermo_I_set(0) <= cPID_mtResetTermo_I_notify;
cPID_atTermo_I_new_value(1) <= cPID_mtCalculaI_output;
cPID_atTermo_I_set(1) <= cPID_mtCalculaI_notify;
cPID_atTermo_D_new_value(0) <= cPID_mtResetTermo_D_output;
cPID_atTermo_D_set(0) <= cPID_mtResetTermo_D_notify;
cPID_atTermo_D_new_value(1) <= cPID_mtCalculaD_output;
cPID_atTermo_D_set(1) <= cPID_mtCalculaD_notify;
cPID_atTemp_new_value(0) <= cPID_mtResetTemp_output;
cPID_atTemp_set(0) <= cPID_mtResetTemp_notify;
cPID_atTemp_new_value(1) <= cPID_mtTmp_output;
cPID_atTemp_set(1) <= cPID_mtTmp_notify;
cPID_atReferencia_new_value(0) <= in_cPID_atReferencia;
cPID_atReferencia_set(0) <= in_set_cPID_atReferencia;
cPID_atSaidaTmp_new_value(0) <= cPID_mtResetSaidaTmp_output;
cPID_atSaidaTmp_set(0) <= cPID_mtResetSaidaTmp_notify;
cPID_atSaidaTmp_new_value(1) <= cPID_mtSaidaTmp_output;
cPID_atSaidaTmp_set(1) <= cPID_mtSaidaTmp_notify;

---------- attributes outputs -------------
out_cPID_atSinalControle <= cPID_atSinalControle_value;

---------- premises inputs -------------
prMaiorMax_imput_a <= cPID_atErroIntTmp_value;
prMenorMin_imput_a <= cPID_atErroIntTmp_value;
prReset_imput_a <= cPID_atReset_value;
prResetI_imput_a <= cPID_atReset_value;
prSaidaMaiorMax_imput_a <= cPID_atSaidaTmp_value;
prSaidaMenorMin_imput_a <= cPID_atSaidaTmp_value;

prMaiorMax_imput_b <= "0010011100010000";
prMenorMin_imput_b <= "1101100011110000";
prReset_imput_b <= "1";
prResetI_imput_b <= "0";
prSaidaMaiorMax_imput_b <= "0010011100010000";
```

```
prSaidaMenorMin_imput_b <= "1101100011110000";

---------- Methods Imputs -------------
cPID_mtCalculaErro_execute <= (prResetI_result);
cPID_mtGravaErroP_execute <= (prResetI_result);
cPID_mtErroIntTmp_execute <= (prResetI_result);
cPID_mtErroD_execute <= (prResetI_result);
cPID_mtArmazenaErro_execute <= (prResetI_result);
cPID_mtCalculaP_execute <= (prResetI_result);
cPID_mtCalculaErroI_execute <= (prResetI_result);
cPID_mtCalculaD_execute <= (prResetI_result);
cPID_mtCalculaI_execute <= (prResetI_result);
cPID_mtTmp_execute <= (prResetI_result);
cPID_mtSaidaTmp_execute <= (prResetI_result);
cPID_mtSaida_execute <= (prResetI_result);
cPID_mtMaiorMax_execute <= (prMaiorMax_result);
cPID_mtMenorMin_execute <= (prMenorMin_result);
cPID_mtResetSaida_execute <= (prReset_result);
cPID_mtResetErroAnterior_execute <= (prReset_result);
cPID_mtResetErro_execute <= (prReset_result);
cPID_mtResetErroP_execute <= (prReset_result);
cPID_mtResetErroIntTmp_execute <= (prReset_result);
cPID_mtResetErroD_execute <= (prReset_result);
cPID_mtResetErroI_execute <= (prReset_result);
cPID_mtResetTermo_P_execute <= (prReset_result);
cPID_mtResetTermo_I_execute <= (prReset_result);
cPID_mtResetTermo_D_execute <= (prReset_result);
cPID_mtResetTemp_execute <= (prReset_result);
cPID_mtResetSaidaTmp_execute <= (prReset_result);
cPID_mtSaidaMaiorMax_execute <= (prSaidaMaiorMax_result);
cPID_mtSaidaMenorMin_execute <= (prSaidaMenorMin_result);
cPID_mtResetSaida_input_a <= "0000000000000000";
cPID_mtResetSaida_input_b <= "0000000000000000";
cPID_mtResetErroAnterior_input_a <= "0000000000000000";
cPID_mtResetErroAnterior_input_b <= "0000000000000000";
cPID_mtResetErro_input_a <= "0000000000000000";
cPID_mtResetErro_input_b <= "0000000000000000";
cPID_mtResetErroP_input_a <= "0000000000000000";
cPID_mtResetErroP_input_b <= "0000000000000000";
cPID_mtResetErroIntTmp_input_a <= "0000000000000000";
cPID_mtResetErroIntTmp_input_b <= "0000000000000000";
cPID_mtResetErroD_input_a <= "0000000000000000";
cPID_mtResetErroD_input_b <= "0000000000000000";
cPID_mtResetErroI_input_a <= "0000000000000000";
cPID_mtResetErroI_input_b <= "0000000000000000";
cPID_mtResetTermo_P_input_a <= "0000000000000000";
cPID_mtResetTermo_P_input_b <= "0000000000000000";
cPID_mtResetTermo_I_input_a <= "0000000000000000";
cPID_mtResetTermo_I_input_b <= "0000000000000000";
cPID_mtResetTermo_D_input_a <= "0000000000000000";
cPID_mtResetTermo_D_input_b <= "0000000000000000";
```

```
cPID_mtResetTemp_input_a <= "0000000000000000";
cPID_mtResetTemp_input_b <= "0000000000000000";
cPID_mtResetSaidaTmp_input_a <= "0000000000000000";
cPID_mtResetSaidaTmp_input_b <= "0000000000000000";
cPID_mtMaiorMax_input_a <= "0010011100010000";
cPID_mtMaiorMax_input_b <= "0000000000000000";
cPID_mtMenorMin_input_a <= "1101100011110000";
cPID_mtMenorMin_input_b <= "0000000000000000";
cPID_mtSaidaMaiorMax_input_a <= "0010011100010000";
cPID_mtSaidaMaiorMax_input_b <= "0000000000000000";
cPID_mtSaidaMenorMin_input_a <= "1101100011110000";
cPID_mtSaidaMenorMin_input_b <= "0000000000000000";
cPID_mtCalculaErro_input_a <= cPID_atReferencia_value;
cPID_mtCalculaErro_input_b <= cPID_atRealimentacao_value;
cPID_mtGravaErroP_input_a <= cPID_atErro_value;
cPID_mtGravaErroP_input_b <= "0000000000000000";
cPID_mtErroIntTmp_input_a <= cPID_atErroI_value;
cPID_mtErroIntTmp_input_b <= cPID_atErro_value;
cPID_mtErroD_input_a <= cPID_atErro_value;
cPID_mtErroD_input_b <= cPID_atErroAnterior_value;
cPID_mtArmazenaErro_input_a <= cPID_atErro_value;
cPID_mtArmazenaErro_input_b <= "0000000000000000";
cPID_mtCalculaP_input_a <= cPID_atKp_value;
cPID_mtCalculaP_input_b <= cPID_atErroP_value;
cPID_mtCalculaErroI_input_a <= cPID_atErroIntTmp_value;
cPID_mtCalculaErroI_input_b <= "0000000000000000";
cPID_mtCalculaD_input_a <= cPID_atKd_value;
cPID_mtCalculaD_input_b <= cPID_atErroD_value;
cPID_mtCalculaI_input_a <= cPID_atKi_value;
cPID_mtCalculaI_input_b <= cPID_atErroI_value;
cPID_mtTmp_input_a <= cPID_atTermo_P_value;
cPID_mtTmp_input_b <= cPID_atTermo_D_value;
cPID_mtSaidaTmp_input_a <= cPID_atTemp_value;
cPID_mtSaidaTmp_input_b <= cPID_atTermo_I_value;
cPID_mtSaida_input_a <= cPID_atSaidaTmp_value;
cPID_mtSaida_input_b <= "0000000000000000";
END controladorPID_arq;
```

Apêndice U – controladorPID.cpp

Código fonte em C++ do experimento do controlador PID utilizado com a ferramenta Vivado HLS.

```
#include "controladorPID.h"

int16 verificaLimites(int32 valor)
{
 if(valor>valorMaximo) return(valorMaximo);
 if(valor<valorMinimo) return(valorMinimo);
 return(valor);
}

void controladorPID(bool reset, int16 referencia, int16
realimentacao, int16 *sinalControle, int16 kp, int16 ki, int16
kd)
{
 static int16 erroIntegral;
 static int16 erroAnterior;
 int16 erro;
 int16 termo_P;
 int16 termo_I;
 int16 termo_D;
 if(reset==true)
 {
  // reinicia os valores do controlador
  erroIntegral=0;
  erroAnterior=0;
  *sinalControle=0;
 }
 else
 {
  // calcula o erro atual
  erro=referencia-realimentacao;
  // acumula o erro
  erroIntegral=verificaLimites((int32)erroIntegral+(int32)erro);
  // calcula o termo proporcional
  termo_P=verificaLimites((int32)kp*(int32)erro);
  // calcula o termo integral
  termo_I=verificaLimites((int32)ki*(int32)erroIntegral);
  // calcula o termo derivativo
  termo_D=verificaLimites((int32)ki*(int32)(erro-erroAnterior));
  // calcula a saída do controlador
 *sinalControle=verificaLimites((int32)termo_P+(int32)termo_I+(
int32)termo_D);
  // armazena o erro para a próxima iteração
  erroAnterior=erro;
 }
}
```

Apêndice V – controladorPID.vhd

Código fonte em VHDL para o experimento do controlador PID gerado automaticamente pela ferramenta Vivado HLS.

```
--
==========================================================
-- RTL generated by Vivado(TM) HLS - High-Level Synthesis from
C, C++ and SystemC
-- Version: 2018.1_AR70908
-- Copyright (C) 1986-2018 Xilinx, Inc. All Rights Reserved.
--
-- ==========================================================

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity controladorPID is
port (
    ap_clk : IN STD_LOGIC;
    ap_rst : IN STD_LOGIC;
    ap_start : IN STD_LOGIC;
    ap_done : OUT STD_LOGIC;
    ap_idle : OUT STD_LOGIC;
    ap_ready : OUT STD_LOGIC;
    reset : IN STD_LOGIC;
    referencia : IN STD_LOGIC_VECTOR (15 downto 0);
    realimentacao : IN STD_LOGIC_VECTOR (15 downto 0);
    sinalControle : OUT STD_LOGIC_VECTOR (15 downto 0);
    sinalControle_ap_vld : OUT STD_LOGIC;
    kp : IN STD_LOGIC_VECTOR (15 downto 0);
    ki : IN STD_LOGIC_VECTOR (15 downto 0);
    kd : IN STD_LOGIC_VECTOR (15 downto 0) );
end;


architecture behav of controladorPID is
    attribute CORE_GENERATION_INFO : STRING;
    attribute CORE_GENERATION_INFO of behav : architecture is

"controladorPID,hls_ip_2018_1_AR70908,{HLS_INPUT_TYPE=cxx,HLS_
INPUT_FLOAT=0,HLS_INPUT_FIXED=1,HLS_INPUT_PART=xa7a12tcsg325-
1q,HLS_INPUT_CLOCK=20.000000,HLS_INPUT_ARCH=others,HLS_SYN_CLO
CK=15.556000,HLS_SYN_LAT=2,HLS_SYN_TPT=none,HLS_SYN_MEM=0,HLS_
SYN_DSP=3,HLS_SYN_FF=133,HLS_SYN_LUT=585}";
    constant ap_const_logic_1 : STD_LOGIC := '1';
    constant ap_const_logic_0 : STD_LOGIC := '0';
    constant ap_ST_fsm_state1 : STD_LOGIC_VECTOR (2 downto 0)
:= "001";
```

```
    constant ap_ST_fsm_state2 : STD_LOGIC_VECTOR (2 downto 0)
:= "010";
    constant ap_ST_fsm_state3 : STD_LOGIC_VECTOR (2 downto 0)
:= "100";
    constant ap_const_lv32_0 : STD_LOGIC_VECTOR (31 downto 0)
:= "00000000000000000000000000000000";
    constant ap_const_lv16_0 : STD_LOGIC_VECTOR (15 downto 0)
:= "0000000000000000";
    constant ap_const_lv1_0 : STD_LOGIC_VECTOR (0 downto 0) :=
"0";
    constant ap_const_lv32_1 : STD_LOGIC_VECTOR (31 downto 0)
:= "00000000000000000000000000000001";
    constant ap_const_lv1_1 : STD_LOGIC_VECTOR (0 downto 0) :=
"1";
    constant ap_const_lv32_2 : STD_LOGIC_VECTOR (31 downto 0)
:= "00000000000000000000000000000010";
    constant ap_const_lv17_7D00 : STD_LOGIC_VECTOR (16 downto
0) := "00111110100000000";
    constant ap_const_lv17_18300 : STD_LOGIC_VECTOR (16 downto
0) := "11000001100000000";
    constant ap_const_lv16_7D00 : STD_LOGIC_VECTOR (15 downto
0) := "0111110100000000";
    constant ap_const_lv16_8300 : STD_LOGIC_VECTOR (15 downto
0) := "1000001100000000";
    constant ap_const_lv32_7D00 : STD_LOGIC_VECTOR (31 downto
0) := "00000000000000000111110100000000";
    constant ap_const_lv32_FFFF8300 : STD_LOGIC_VECTOR (31
downto 0) := "11111111111111111000001100000000";
    constant ap_const_lv18_7D00 : STD_LOGIC_VECTOR (17 downto
0) := "00111110100000000";
    constant ap_const_lv18_38300 : STD_LOGIC_VECTOR (17 downto
0) := "11100000100000000";
    constant ap_const_boolean_1 : BOOLEAN := true;

    signal ap_CS_fsm : STD_LOGIC_VECTOR (2 downto 0) := "001";
    attribute fsm_encoding : string;
    attribute fsm_encoding of ap_CS_fsm : signal is "none";
    signal ap_CS_fsm_state1 : STD_LOGIC;
    attribute fsm_encoding of ap_CS_fsm_state1 : signal is
"none";
    signal erroIntegral : STD_LOGIC_VECTOR (15 downto 0) :=
"0000000000000000";
    signal erroAnterior : STD_LOGIC_VECTOR (15 downto 0) :=
"0000000000000000";
    signal reset_read_read_fu_72_p2 : STD_LOGIC_VECTOR (0
downto 0);
    signal UnifiedRetVal_i_fu_146_p3 : STD_LOGIC_VECTOR (15
downto 0);
    signal UnifiedRetVal_i_reg_401 : STD_LOGIC_VECTOR (15
downto 0);
    signal termo_P_fu_191_p3 : STD_LOGIC_VECTOR (15 downto 0);
```

```
    signal termo_P_reg_406 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_8_fu_199_p1 : STD_LOGIC_VECTOR (31 downto 0);
    signal tmp_8_reg_412 : STD_LOGIC_VECTOR (31 downto 0);
    signal termo_D_fu_238_p3 : STD_LOGIC_VECTOR (15 downto 0);
    signal termo_D_reg_417 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_i3_fu_332_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_i3_reg_423 : STD_LOGIC_VECTOR (0 downto 0);
    signal ap_CS_fsm_state2 : STD_LOGIC;
    attribute fsm_encoding of ap_CS_fsm_state2 : signal is
"none";
    signal tmp_1_i3_fu_338_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_1_i3_reg_429 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_2_i2_fu_344_p2 : STD_LOGIC_VECTOR (15 downto
0);
    signal tmp_2_i2_reg_434 : STD_LOGIC_VECTOR (15 downto 0);
    signal erro_fu_90_p2 : STD_LOGIC_VECTOR (15 downto 0);
    signal UnifiedRetVal_i3_fu_361_p3 : STD_LOGIC_VECTOR (15
downto 0);
    signal ap_CS_fsm_state3 : STD_LOGIC;
    attribute fsm_encoding of ap_CS_fsm_state3 : signal is
"none";
    signal tmp_cast_fu_96_p0 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_cast_fu_96_p1 : STD_LOGIC_VECTOR (16 downto 0);
    signal tmp_3_cast_fu_104_p1 : STD_LOGIC_VECTOR (16 downto
0);
    signal valor_assign_fu_108_p2 : STD_LOGIC_VECTOR (16
downto 0);
    signal tmp_2_i_fu_126_p1 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_i_fu_114_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_1_i_fu_120_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_fu_140_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal p_i_fu_132_p3 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_2_i_fu_126_p2 : STD_LOGIC_VECTOR (15 downto 0);
    signal valor_assign_1_fu_369_p2 : STD_LOGIC_VECTOR (31
downto 0);
    signal tmp_i1_fu_164_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_1_i2_fu_169_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_1_fu_185_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal p_i4_fu_177_p3 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_10_fu_174_p1 : STD_LOGIC_VECTOR (15 downto 0);
    signal grp_fu_378_p3 : STD_LOGIC_VECTOR (31 downto 0);
    signal tmp_i2_fu_211_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_1_i1_fu_216_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_5_fu_232_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal p_i1_fu_224_p3 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_12_fu_221_p1 : STD_LOGIC_VECTOR (15 downto 0);
    signal valor_assign_2_fu_389_p2 : STD_LOGIC_VECTOR (31
downto 0);
    signal tmp_i6_fu_267_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_1_i7_fu_272_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal tmp_4_fu_288_p2 : STD_LOGIC_VECTOR (0 downto 0);
```

```
    signal p_i9_fu_280_p3 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_11_fu_277_p1 : STD_LOGIC_VECTOR (15 downto 0);
    signal termo_I_fu_294_p3 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_11_cast_fu_302_p1 : STD_LOGIC_VECTOR (16 downto
0);
    signal tmp_13_cast_fu_309_p1 : STD_LOGIC_VECTOR (16 downto
0);
    signal tmp6_fu_316_p2 : STD_LOGIC_VECTOR (16 downto 0);
    signal tmp21_cast_fu_322_p1 : STD_LOGIC_VECTOR (17 downto
0);
    signal tmp_12_cast_fu_305_p1 : STD_LOGIC_VECTOR (17 downto
0);
    signal valor_assign_4_fu_326_p2 : STD_LOGIC_VECTOR (17
downto 0);
    signal tmp_7_fu_312_p2 : STD_LOGIC_VECTOR (15 downto 0);
    signal tmp_s_fu_357_p2 : STD_LOGIC_VECTOR (0 downto 0);
    signal p_i2_fu_350_p3 : STD_LOGIC_VECTOR (15 downto 0);
    signal grp_fu_378_p0 : STD_LOGIC_VECTOR (15 downto 0);
    signal valor_assign_2_fu_389_p1 : STD_LOGIC_VECTOR (15
downto 0);
    signal ap_NS_fsm : STD_LOGIC_VECTOR (2 downto 0);

    component controladorPID_mubkb IS
    generic (
        ID : INTEGER;
        NUM_STAGE : INTEGER;
        din0_WIDTH : INTEGER;
        din1_WIDTH : INTEGER;
        dout_WIDTH : INTEGER );
    port (
        din0 : IN STD_LOGIC_VECTOR (15 downto 0);
        din1 : IN STD_LOGIC_VECTOR (15 downto 0);
        dout : OUT STD_LOGIC_VECTOR (31 downto 0) );
    end component;


    component controladorPID_amcud IS
    generic (
        ID : INTEGER;
        NUM_STAGE : INTEGER;
        din0_WIDTH : INTEGER;
        din1_WIDTH : INTEGER;
        din2_WIDTH : INTEGER;
        dout_WIDTH : INTEGER );
    port (
        din0 : IN STD_LOGIC_VECTOR (15 downto 0);
        din1 : IN STD_LOGIC_VECTOR (15 downto 0);
        din2 : IN STD_LOGIC_VECTOR (15 downto 0);
        dout : OUT STD_LOGIC_VECTOR (31 downto 0) );
    end component;
```

```
begin
    controladorPID_mubkb_U1 : component controladorPID_mubkb
    generic map (
        ID => 1,
        NUM_STAGE => 1,
        din0_WIDTH => 16,
        din1_WIDTH => 16,
        dout_WIDTH => 32)
    port map (
        din0 => kp,
        din1 => erro_fu_90_p2,
        dout => valor_assign_1_fu_369_p2);

    controladorPID_amcud_U2 : component controladorPID_amcud
    generic map (
        ID => 1,
        NUM_STAGE => 1,
        din0_WIDTH => 16,
        din1_WIDTH => 16,
        din2_WIDTH => 16,
        dout_WIDTH => 32)
    port map (
        din0 => grp_fu_378_p0,
        din1 => erroAnterior,
        din2 => ki,
        dout => grp_fu_378_p3);

    controladorPID_mubkb_U3 : component controladorPID_mubkb
    generic map (
        ID => 1,
        NUM_STAGE => 1,
        din0_WIDTH => 16,
        din1_WIDTH => 16,
        dout_WIDTH => 32)
    port map (
        din0 => UnifiedRetVal_i_reg_401,
        din1 => valor_assign_2_fu_389_p1,
        dout => valor_assign_2_fu_389_p2);




    ap_CS_fsm_assign_proc : process(ap_clk)
    begin
        if (ap_clk'event and ap_clk =  '1') then
            if (ap_rst = '1') then
                ap_CS_fsm <= ap_ST_fsm_state1;
            else
```

```
                ap_CS_fsm <= ap_NS_fsm;
            end if;
        end if;
    end process;


    erroAnterior_assign_proc : process (ap_clk)
    begin
        if (ap_clk'event and ap_clk = '1') then
            if (((ap_start = ap_const_logic_1) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
                if ((reset_read_read_fu_72_p2 =
ap_const_lv1_1)) then
                    erroAnterior <= ap_const_lv16_0;
                elsif ((reset_read_read_fu_72_p2 =
ap_const_lv1_0)) then
                    erroAnterior <= erro_fu_90_p2;
                end if;
            end if;
        end if;
    end process;

    erroIntegral_assign_proc : process (ap_clk)
    begin
        if (ap_clk'event and ap_clk = '1') then
            if (((ap_start = ap_const_logic_1) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
                if ((reset_read_read_fu_72_p2 =
ap_const_lv1_1)) then
                    erroIntegral <= ap_const_lv16_0;
                elsif ((reset_read_read_fu_72_p2 =
ap_const_lv1_0)) then
                    erroIntegral <= UnifiedRetVal_i_fu_146_p3;
                end if;
            end if;
        end if;
    end process;
    process (ap_clk)
    begin
        if (ap_clk'event and ap_clk = '1') then
            if (((ap_start = ap_const_logic_1) and
(reset_read_read_fu_72_p2 = ap_const_lv1_0) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
                UnifiedRetVal_i_reg_401 <=
UnifiedRetVal_i_fu_146_p3;
                termo_D_reg_417 <= termo_D_fu_238_p3;
                termo_P_reg_406 <= termo_P_fu_191_p3;
                tmp_8_reg_412 <= tmp_8_fu_199_p1;
            end if;
        end if;
    end process;
```

```
    process (ap_clk)
    begin
        if (ap_clk'event and ap_clk = '1') then
            if ((ap_const_logic_1 = ap_CS_fsm_state2)) then
                tmp_1_i3_reg_429 <= tmp_1_i3_fu_338_p2;
                tmp_2_i2_reg_434 <= tmp_2_i2_fu_344_p2;
                tmp_i3_reg_423 <= tmp_i3_fu_332_p2;
            end if;
        end if;
    end process;

    ap_NS_fsm_assign_proc : process (ap_start, ap_CS_fsm,
ap_CS_fsm_state1, reset_read_read_fu_72_p2)
    begin
        case ap_CS_fsm is
            when ap_ST_fsm_state1 =>
                if (((ap_start = ap_const_logic_1) and
(reset_read_read_fu_72_p2 = ap_const_lv1_1) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
                    ap_NS_fsm <= ap_ST_fsm_state3;
                elsif (((ap_start = ap_const_logic_1) and
(reset_read_read_fu_72_p2 = ap_const_lv1_0) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
                    ap_NS_fsm <= ap_ST_fsm_state2;
                else
                    ap_NS_fsm <= ap_ST_fsm_state1;
                end if;
            when ap_ST_fsm_state2 =>
                ap_NS_fsm <= ap_ST_fsm_state3;
            when ap_ST_fsm_state3 =>
                ap_NS_fsm <= ap_ST_fsm_state1;
            when others =>
                ap_NS_fsm <= "XXX";
        end case;
    end process;
    UnifiedRetVal_i3_fu_361_p3 <=
        p_i2_fu_350_p3 when (tmp_s_fu_357_p2(0) = '1') else
        tmp_2_i2_reg_434;
    UnifiedRetVal_i_fu_146_p3 <=
        p_i_fu_132_p3 when (tmp_fu_140_p2(0) = '1') else
        tmp_2_i_fu_126_p2;
    ap_CS_fsm_state1 <= ap_CS_fsm(0);
    ap_CS_fsm_state2 <= ap_CS_fsm(1);
    ap_CS_fsm_state3 <= ap_CS_fsm(2);

    ap_done_assign_proc : process(ap_CS_fsm_state3)
    begin
        if ((ap_const_logic_1 = ap_CS_fsm_state3)) then
            ap_done <= ap_const_logic_1;
        else
            ap_done <= ap_const_logic_0;
```

```
        end if;
    end process;


    ap_idle_assign_proc : process(ap_start, ap_CS_fsm_state1)
    begin
        if (((ap_start = ap_const_logic_0) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
            ap_idle <= ap_const_logic_1;
        else
            ap_idle <= ap_const_logic_0;
        end if;
    end process;


    ap_ready_assign_proc : process(ap_CS_fsm_state3)
    begin
        if ((ap_const_logic_1 = ap_CS_fsm_state3)) then
            ap_ready <= ap_const_logic_1;
        else
            ap_ready <= ap_const_logic_0;
        end if;
    end process;

    erro_fu_90_p2 <= std_logic_vector(unsigned(referencia) -
unsigned(realimentacao));
    grp_fu_378_p0 <= tmp_3_cast_fu_104_p1(16 - 1 downto 0);
    p_i1_fu_224_p3 <=
        ap_const_lv16_7D00 when (tmp_i2_fu_211_p2(0) = '1')
else
        ap_const_lv16_8300;
    p_i2_fu_350_p3 <=
        ap_const_lv16_7D00 when (tmp_i3_reg_423(0) = '1') else
        ap_const_lv16_8300;
    p_i4_fu_177_p3 <=
        ap_const_lv16_7D00 when (tmp_i1_fu_164_p2(0) = '1')
else
        ap_const_lv16_8300;
    p_i9_fu_280_p3 <=
        ap_const_lv16_7D00 when (tmp_i6_fu_267_p2(0) = '1')
else
        ap_const_lv16_8300;
    p_i_fu_132_p3 <=
        ap_const_lv16_7D00 when (tmp_i_fu_114_p2(0) = '1')
else
        ap_const_lv16_8300;
    reset_read_read_fu_72_p2 <= (0=>reset, others=>'-');

    sinalControle_assign_proc : process(ap_start,
ap_CS_fsm_state1, reset_read_read_fu_72_p2,
UnifiedRetVal_i3_fu_361_p3, ap_CS_fsm_state3)
```

```
    begin
        if (((reset_read_read_fu_72_p2 = ap_const_lv1_0) and
(ap_const_logic_1 = ap_CS_fsm_state3))) then
            sinalControle <= UnifiedRetVal_i3_fu_361_p3;
        elsif (((ap_start = ap_const_logic_1) and
(reset_read_read_fu_72_p2 = ap_const_lv1_1) and
(ap_const_logic_1 = ap_CS_fsm_state1))) then
            sinalControle <= ap_const_lv16_0;
        else
            sinalControle <= "XXXXXXXXXXXXXXXX";
        end if;
    end process;


    sinalControle_ap_vld_assign_proc : process(ap_start,
ap_CS_fsm_state1, reset_read_read_fu_72_p2, ap_CS_fsm_state3)
    begin
        if ((((reset_read_read_fu_72_p2 = ap_const_lv1_0) and
(ap_const_logic_1 = ap_CS_fsm_state3)) or ((ap_start =
ap_const_logic_1) and (reset_read_read_fu_72_p2 =
ap_const_lv1_1) and (ap_const_logic_1 = ap_CS_fsm_state1))))
then
            sinalControle_ap_vld <= ap_const_logic_1;
        else
            sinalControle_ap_vld <= ap_const_logic_0;
        end if;
    end process;

    termo_D_fu_238_p3 <=
        p_i1_fu_224_p3 when (tmp_5_fu_232_p2(0) = '1') else
        tmp_12_fu_221_p1;
    termo_I_fu_294_p3 <=
        p_i9_fu_280_p3 when (tmp_4_fu_288_p2(0) = '1') else
        tmp_11_fu_277_p1;
    termo_P_fu_191_p3 <=
        p_i4_fu_177_p3 when (tmp_1_fu_185_p2(0) = '1') else
        tmp_10_fu_174_p1;
        tmp21_cast_fu_322_p1 <=
std_logic_vector(IEEE.numeric_std.resize(signed(tmp6_fu_316_p2
),18));

    tmp6_fu_316_p2 <=
std_logic_vector(signed(tmp_11_cast_fu_302_p1) +
signed(tmp_13_cast_fu_309_p1));
    tmp_10_fu_174_p1 <= valor_assign_1_fu_369_p2(16 - 1 downto
0);
        tmp_11_cast_fu_302_p1 <=
std_logic_vector(IEEE.numeric_std.resize(signed(termo_P_reg_40
6),17));
```

```
    tmp_11_fu_277_p1 <= valor_assign_2_fu_389_p2(16 - 1 downto
0);
        tmp_12_cast_fu_305_p1 <=
std_logic_vector(IEEE.numeric_std.resize(signed(termo_I_fu_294
_p3),18));

    tmp_12_fu_221_p1 <= grp_fu_378_p3(16 - 1 downto 0);
        tmp_13_cast_fu_309_p1 <=
std_logic_vector(IEEE.numeric_std.resize(signed(termo_D_reg_41
7),17));

    tmp_1_fu_185_p2 <= (tmp_i1_fu_164_p2 or
tmp_1_i2_fu_169_p2);
    tmp_1_i1_fu_216_p2 <= "1" when (signed(grp_fu_378_p3) <
signed(ap_const_lv32_FFFF8300)) else "0";
    tmp_1_i2_fu_169_p2 <= "1" when
(signed(valor_assign_1_fu_369_p2) <
signed(ap_const_lv32_FFFF8300)) else "0";
    tmp_1_i3_fu_338_p2 <= "1" when
(signed(valor_assign_4_fu_326_p2) <
signed(ap_const_lv18_38300)) else "0";
    tmp_1_i7_fu_272_p2 <= "1" when
(signed(valor_assign_2_fu_389_p2) <
signed(ap_const_lv32_FFFF8300)) else "0";
    tmp_1_i_fu_120_p2 <= "1" when
(signed(valor_assign_fu_108_p2) < signed(ap_const_lv17_18300))
else "0";
    tmp_2_i2_fu_344_p2 <=
std_logic_vector(signed(termo_I_fu_294_p3) +
signed(tmp_7_fu_312_p2));
    tmp_2_i_fu_126_p1 <= erroIntegral;
    tmp_2_i_fu_126_p2 <=
std_logic_vector(signed(erro_fu_90_p2) +
signed(tmp_2_i_fu_126_p1));
        tmp_3_cast_fu_104_p1 <=
std_logic_vector(IEEE.numeric_std.resize(signed(erro_fu_90_p2)
,17));

    tmp_4_fu_288_p2 <= (tmp_i6_fu_267_p2 or
tmp_1_i7_fu_272_p2);
    tmp_5_fu_232_p2 <= (tmp_i2_fu_211_p2 or
tmp_1_i1_fu_216_p2);
    tmp_7_fu_312_p2 <=
std_logic_vector(signed(termo_D_reg_417) +
signed(termo_P_reg_406));
        tmp_8_fu_199_p1 <=
std_logic_vector(IEEE.numeric_std.resize(signed(ki),32));

    tmp_cast_fu_96_p0 <= erroIntegral;
```

```
        tmp_cast_fu_96_p1 <=
std_logic_vector(IEEE.numeric_std.resize(signed(tmp_cast_fu_96
_p0),17));

    tmp_fu_140_p2 <= (tmp_i_fu_114_p2 or tmp_1_i_fu_120_p2);
    tmp_i1_fu_164_p2 <= "1" when
(signed(valor_assign_1_fu_369_p2) >
signed(ap_const_lv32_7D00)) else "0";
    tmp_i2_fu_211_p2 <= "1" when (signed(grp_fu_378_p3) >
signed(ap_const_lv32_7D00)) else "0";
    tmp_i3_fu_332_p2 <= "1" when
(signed(valor_assign_4_fu_326_p2) >
signed(ap_const_lv18_7D00)) else "0";
    tmp_i6_fu_267_p2 <= "1" when
(signed(valor_assign_2_fu_389_p2) >
signed(ap_const_lv32_7D00)) else "0";
    tmp_i_fu_114_p2 <= "1" when
(signed(valor_assign_fu_108_p2) > signed(ap_const_lv17_7D00))
else "0";
    tmp_s_fu_357_p2 <= (tmp_i3_reg_423 or tmp_1_i3_reg_429);
    valor_assign_2_fu_389_p1 <= tmp_8_reg_412(16 - 1 downto
0);
    valor_assign_4_fu_326_p2 <=
std_logic_vector(signed(tmp21_cast_fu_322_p1) +
signed(tmp_12_cast_fu_305_p1));
    valor_assign_fu_108_p2 <=
std_logic_vector(signed(tmp_cast_fu_96_p1) +
signed(tmp_3_cast_fu_104_p1));
end behav;
```

ANEXOS

Nesta seção são apresentados os anexos, os quais são constituídos por documentos que não foram criados pelo autor deste trabalho.

No ANEXO A é apresentado o relatório feito por Ricardo Jasinski e entregue ao professor Carlos Raimundo Erig Lima como trabalho de conclusão da disciplina de “LRH0087 - Lógica Reconfigurável Por *Hardware”*.

O ANEXO B, por sua vez apresenta o relatório feito por André Augusto Kaviatkovski e Gabriel Rodrigues Garcia e entregue ao professor Carlos Raimundo Erig Lima como trabalho de conclusão da disciplina de “LRH0087 - Lógica Reconfigurável Por *Hardware*”.

O programa em LingPON-HD 1.0 desenvolvido pelos discentes André Augusto Kaviatkovski e Gabriel Rodrigues Garcia para a disciplina de “LRH0087 - Lógica Reconfigurável Por *Hardware*” é apresentado no ANEXO C.

E por fim, o ANEXO D apresenta uma avaliação realizada pelos discentes André Augusto Kaviatkovski e Gabriel Rodrigues Garcia relatando sua experiência com a utilização do ferramental PON-HD 1.0.

Anexo A – Relatório de Ricardo Jasinski

Framework para Geração de Hardware em VHDL a Partir de Modelos em PON (Paradigma Orientado a Notificações)

Aluno Ricardo Pereira Jasinski
Disciplina Lógica Reconfigurável por Hardware
Professor Carlos R. Erig
Fase 3/2011
Data 22/1/2012

Sumário

1 Introdução

1.1 Objetivo Geral

O objetivo geral deste trabalho é investigar os potenciais benefícios da implementação em hardware dedicado de aplicações concebidas segundo o Paradigma Orientado a Notificações (PON). Como o PON é formado por componentes que operam de maneira concorrente, espera-se que uma implementação em hardware apresente benefícios quando comparada a uma implementação puramente em software, na qual o paralelismo precisa ser simulado, ou é limitado pelo número de núcleos do processador utilizado.

Uma análise detalhada da implementação do PON em hardware irá requerer um bom número de aplicações de exemplo; além disso, como pode haver uma grande variação nas técnicas de projeto de tais aplicações, o resultado final dependeria da proficiência e do esforço dos projetistas envolvidos. Desta forma, uma ferramenta capaz de traduzir automaticamente uma aplicação em PON para uma linguagem de descrição de hardware será útil para avaliar de uma forma mais isenta os eventuais benefícios encontrados. O objetivo principal deste trabalho será o desenvolvimento de uma ferramenta capaz de gerar automaticamente uma implementação em hardware, a partir da descrição em alto nível de uma aplicação em PON.

1.2 Objetivos Específicos

O objetivo geral descrito no item anterior pode ser desmembrado em diversas etapas, cujos objetivos serão:

1. Criar um formato de dados de alto nível, capaz de descrever uma aplicação em PON.
2. Criar uma ferramenta capaz de receber um arquivo de entrada, no formato especificado no item anterior, e gerar o código VHDL correspondente.
3. Validar a ferramenta desenvolvida, gerando, compilando e simulando o código VHDL para algumas aplicações de exemplo.
4. Sintetizar o código VHDL para uma FPGA, realizando medidas preliminares sobre o desempenho do hardware gerado.

1.3 Paradigma Orientado a Notificações (PON)

O Paradigma Orientado a Notificações (PON) é uma alternativa aos atuais paradigmas de programação (e.g., programação imperativa, declarativa ou orientada a objetos), visando a eliminar algumas de suas principais deficiências, como a existência de avaliações causais desnecessárias e fortemente acopladas. Este

objetivo é atingido através da decomposição das aplicações em entidades computacionais (i.e., componentes) que se comunicam através de notificações.

Entre as principais vantagens do PON, pode-se citar:

1. Otimização do cálculo causal (i.e., eliminação de operações lógicas repetidas e desnecessárias).
2. Estilo de programação próximo à programação declarativa, o qual possibilita uma representação mais próxima ao conhecimento humano.
3. Separação entre as entidades modeladas e as regras de negócio, habilitando o reuso de código.
4. Facilidade de mapeamento das aplicações para arquiteturas paralelas.

O PON foi introduzido em (SIMÃO, 2001) e (SIMÃO, 2005) e detalhado em (Banaszewski, 2009). O presente trabalho não apresentará detalhes sobre a estrutura do PON; para tanto, recomenda-se a leitura das referências mencionadas.

1.4 Organização do Documento

Ao invés de apresentar um referencial teórico detalhado, os conceitos essenciais são introduzidos ao longo do documento, à medida que se fizerem necessários. Com isto, espera-se facilitar a leitura do texto em ordem sequencial.

Os componentes do PON são descritos brevemente no item 2.1, juntamente com considerações sobre sua implementação em hardware, incluindo diagramas de circuitos.

2 Desenvolvimento

2.1 Componentes do PON

Uma aplicação em PON é formada a partir da combinação de diversos componentes, os quais cooperam e comunicam-se entre si para implementar as funcionalidades desejadas. Os principais componentes do PON são: elementos da base de fatos (*fact base elements*, ou FBEs), atributos (*attributes*), premissas (*premises*), condições (*conditions*), regras (*rules*), ações (*actions*), instigações (*instigations*) e métodos (*methods*). A Figura 1 demonstra o relacionamento entre os componentes do PON para implementar uma aplicação de exemplo, a qual é descrita em (Banaszewski, 2009).

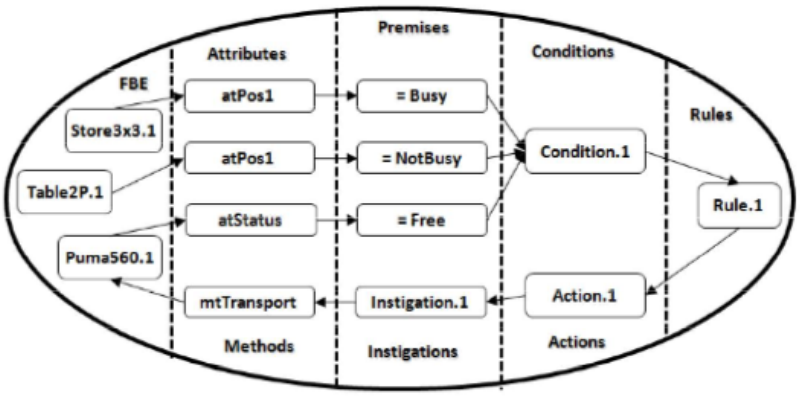

Figura 1. Exemplo da cadeia de notificações formada pelos componentes do PON. Fonte: (Banaszewski, 2009).

Uma breve explicação sobre cada componente é apresentada a seguir, limitando-se às características relevantes a este trabalho. A explanação teórica usada a seguir é um resumo da descrição encontrada em (Banaszewski, 2009). Para cada componente, são também apresentadas considerações relativas à sua implementação em hardware.

O framework do PON é contempla diversos componentes e subcomponentes, sendo que uma aplicação normalmente utiliza apenas uma parte destes. Desta forma, implementar todos os componentes no início do desenvolvimento da ferramenta proposta não permitiria obter resultados rápidos. Além disso, a própria criação da ferramenta constitui um objeto de investigação. Por esses motivos, optou-se por selecionar algumas aplicações de exemplo, e desenvolver o mínimo de código necessário para gerar automaticamente o código VHDL correspondente.

2.1.1 Elementos da Base de Fatos (*Fact Base Elements*)

Em PON, um elemento da base de fatos descreve estados e serviços de entidades (reais ou computacionais) de um dado problema. No contexto deste trabalho, os FBEs consistem em simples *containers* para atributos e métodos relacionados. Desta forma, um FBE não acrescenta nenhuma funcionalidade ao sistema; apenas fornece acessos às funcionalidades de seus métodos e atributos.

Quanto à implementação em hardware, por não introduzirem funcionalidades adicionais, os FBEs tornam-se dispensáveis. Embora uma descrição hierárquica seja útil para os projetistas, as ferramentas de síntese normalmente "planificam" os circuitos gerados, de modo que as fronteiras entre os módulos são perdidas. Consequentemente, não há nenhuma vantagem em se implementar os FBEs em hardware, e a ferramenta proposta irá trabalhar diretamente com os seus objetos constituintes (i.e., atributos e métodos).

Uma observação importante é que, embora no hardware gerado não haja menção aos FBEs, na descrição textual usada como entrada para a ferramenta eles são utilizados. Desta forma, os FBEs e servem como uma maneira de organizar a descrição da aplicação, além de aumentar a compatibilidade com aplicações em PON já existentes.

2.1.2 Atributos (*Attributes*)

Os atributos do PON têm como responsabilidade reagir a mudanças em seus estados, notificando as premissas a eles conectadas quando houver alguma alteração. De uma maneira geral, pode-se dizer que o conjunto de atributos de um sistema compõe o seu estado.

Na cadeia de notifições do PON, um atributo normalmente comunica-se com uma premissa, a qual precisa receber como entrada o valor do atributo, e também uma indicação quando o valor do atributo for alterado. Esta indicação permite a otimização do cálculo causal no PON, pois as expressões são reavaliadas apenas quando os seus valores de entrada forem alterados. Desta forma, conclui-se que uma implementação em hardware do módulo atributo deverá possuir pelo menos duas saídas: o valor do atributo (o qual estará presente em um port chamado *o_value*) e uma flag de notificação (em um port chamado *o_changed*).

Tanto na implementação em hardware (VHDL) como em software (C++), o módulo atributo deve ser especializado de acordo com o tipo de dados que contém. Por exemplo, em VHDL, é necessário implementar as entidades *boolean_attribute*, *integer_attribute*, *float_attribute*, etc. A Figura 2 mostra o diagrama gerado automaticamente pelo software Quartus II (ferramenta *RTL Viewer*) a partir da descrição em VHDL para um *attribute* do tipo *boolean*.

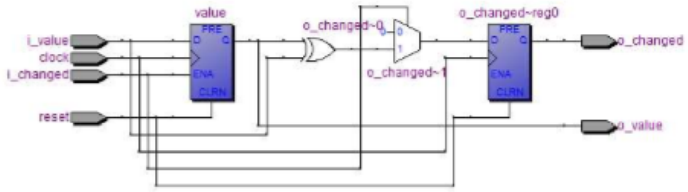

Figura 2. Diagrama do componente *boolean_attribute*.

A Figura 3 mostra o diagrama do componente *integer_attribute*, cuja única diferença em relação ao atributo *boolean* é o tipo de dados armazenado. No exemplo da figura, o valor inteiro é armazenado em um registrador de 32 bits.

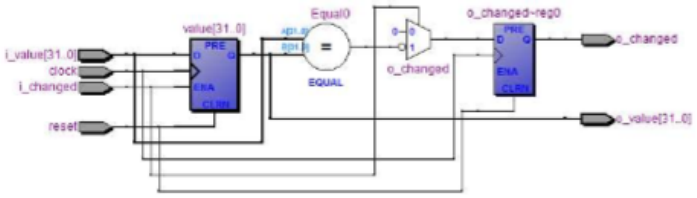

Figura 3. Diagrama do componente *integer_attribute*.

2.1.3 Premissas (*Premises*)

Cada premissa consiste em um teste lógico sobre o valor de um ou dois atributos. Quanto à origem dos valores, as premissas podem ser classificadas em:

1. Comparação do valor de um atributo com uma constante;
2. Comparação entre os valores de dois atributos.

Quanto à operação relacional realizada (*relational operator*, em inglês), uma premissa pode ser configurada para realizar qualquer uma das seguintes comparações:

Tabela 1. Operações relacionais realizadas por uma premissa.

Símbolo do operador (VHDL)	Comparação (operador relacional)	Valor do tipo enumerado (VHDL)
==	igual a	EQUAL_TO
/=	diferente de	NOT_EQUAL_TO
>	maior que	GREATER_THAN
<	menor que	LESS_THAN
>=	maior ou igual a	GREATER_THAN_OR_EQUAL_TO
<=	menor ou igual a	LESS_THAN_OR_EQUAL_TO

Assim como os atributos, as premissas precisam repassar duas informações aos módulos subsequentes: o valor de saída da comparação realizada, e uma notificação quando este valor for alterado. Também como os atributos, os módulos que implementam as premissas precisam ser especializados de acordo com o tipo de dados sobre o qual a premissa opera, e também de acordo com a origem dos valores. A Tabela 2 lista os tipos de premissas especializadas suportados pela ferramenta desenvolvida.

Tabela 2. Módulos especializados para a implementação das premissas.

Nome do módulo especializado	Tipo de dados	Origem dos valores
boolean_premise_compare_with_constant	boolean	1 atributo, 1 constante
integer_premise_compare_with_constant	integer	1 atributo, 1 constante
boolean_premise_compare_attributes	boolean	2 atributos
integer_premise_compare_attributes	integer	2 atributos

A Figura 4 mostra o diagrama para o módulo *boolean_premise_compare_with_constant*, cuja função é comparar o valor de um atributo booleano com um valor constante (*true* ou *false*). A saída *o_changed* é conectada a uma *condition*, para que esta atualize seu valor de saída quando houver uma alteração no valor de qualquer premissa associada.

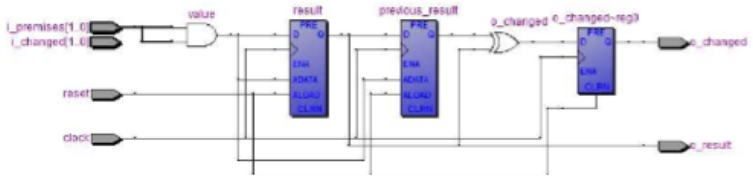

Figura 4. Diagrama do componente *boolean_premise_compare_with_constant*.

A Figura 5 mostra o diagrama para o módulo *integer_premise_compare_with_constant*. No exemplo da figura, o módulo está configurado para fornecer uma saída *true* quando o valor de entrada for igual a 2.

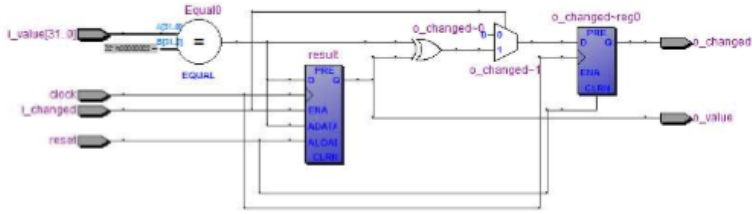

Figura 5. Diagrama do componente *integer_premise_compare_with_constant*.

2.1.4 Condições (*Conditions*)

Uma condição consiste em uma expressão lógica cujas entradas são os valores de saída de uma ou mais premissas. As condições podem ser configuradas quanto à conexão lógica a ser implementada (i.e., conjunção ou disjunção lógica).

Assim como os atributos e as premissas, as condições também precisam repassar aos módulos subsequentes o valor de saída da operação realizada, além de uma notificação quando este valor for alterado. A Figura 6 mostra o diagrama para um componente *condition*, cuja finalidade é realizar a operação lógica AND sobre o valor das premissas de entrada, e notificar em sua saída *o_changed* quando o resultado da expressão lógica for alterado.

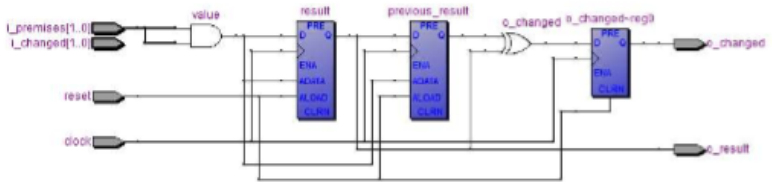

Figura 6. Diagrama do componente *condition*.

O componente *condition* possui dois parâmetros de configuração (*generics*, em VHDL) que podem ser usados em sua instanciação. O parâmetro *ALWAYS_NOTIFY_ON_INPUT_CHANGED* indica se as notificações recebidas na entrada *i_changed* devem ser repassadas adiante mesmo que não ocorra alteração no valor da expressão lógica. O parâmetro *LOGICAL_OPERATION* seleciona o tipo de conjução lógica realizada sobre as premissas (*CONJUNCTION* ou *DISJUNCTION*). O circuito da Figura 6 foi gerado com o parâmetro *ALWAYS_NOTIFY_ON_INPUT_CHANGED* igual a *false*, e portanto neste circuito as entradas *i_changed* estão desconectadas. A Figura 7 mostra uma *condition* configurada com este parâmetro igual a *true*; nesta configuração, se qualquer premissa sofrer uma alteração, a saída *o_changed* será verdadeira por um ciclo de clock, mesmo que o resultado final da operação não seja alterado.

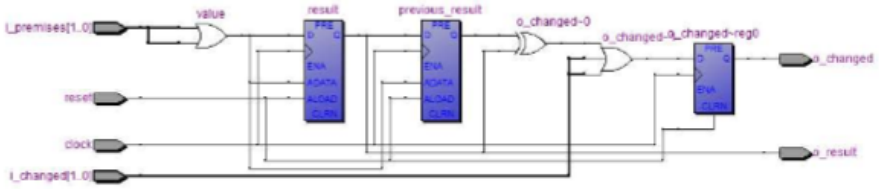

Figura 7. Diagrama do componente *condition*, configurado para propagar notificações.

2.1.5 Regras (*Rules*)

A finalidade de uma regra é descrever um aspecto específico da lógica de negócios da aplicação implementada. Em termos de componentes do PON, uma regra simplesmente conecta uma condição a uma ação, a qual será executada quando a condição for verdadeira.

Em implementações mais recentes do PON em software, as regras não são executadas imediatamente após suas condições se tornarem verdadeiras. Ao invés disso, as regras são apenas aprovadas para execução, e sua execução é controlada por um objeto *Scheduler*, o qual pode considerar outros fatores, como prioridades relativas entre as regras ou estratégias de resolução de conflitos. A versão implementada em hardware, por ora, não contempla uma estratégia de resolução de conflitos, assumindo que o conjunto de regras fornecido é isento de conflitos e é seguro "por *design*". Aplicações com esta característica podem ser implementadas como descrito em (Banaszewski, 2009). Além disso, foi elaborado um sistema simples de prioridade, no qual a primeira regra (mais próxima ao início do arquivo YAML) tem prioridade, caso várias regras tentem sobrescrever o valor de um mesmo atributo.

Desta forma, o hardware correspondente a uma regra é trivial; trata-se de um simples fio conectando a condição à ação. Por este motivo, não é necessário implementar um módulo de hardware para a regra, bastando realizar a conexão entre os sinais correspondentes na entidade top-level.

2.1.6 Ações (*Actions*)

Em PON, uma ação corresponde a um conjunto de operações as quais devem ser realizadas quando uma regra for aprovada para execução. Estas operações são encapsuladas em outro tipo de componente, as instigações (*instigations*). Assim, uma ação é basicamente um *container* para um conjunto de instigações.

Uma ação possui também um parâmetro configurável, o qual indica se as instigações devem ser executadas simultaneamente (em paralelo) ou uma após a outra (em série). Para reduzir a complexidade da primeira versão da ferramenta proposta, apenas a versão paralela foi implementada. Entretanto, a modificação necessária para executar as *instigations* em paralelo é simples, bastando conectar o sinal *output_ready* de uma instigation ao sinal *instigate_request* da próxima *instigation*.

Assim como a regra, o hardware correspondente a uma ação é trivial (um simples fio conectando uma condição a todas as instigações associadas). Assim, não é necessário implementar um módulo em VHDL para uma ação, bastando realizar a conexão entre os sinais correspondentes na entidade top-level.

2.1.7 Instigações (*Instigations*)

Na implementação em software (C++) do PON, uma instigação pode possuir uma de duas funções: redefinir o valor de um atributo ou iniciar a execução de um método (o qual consiste em uma função em C++ encapsulada por um componente do PON). Pelos motivos que serão apresentados no próximo item, a versão inicial da ferramenta não possui suporte a métodos. Assim, o único comportamento possível para uma instigação é redefinir o valor de um atributo.

O valor de origem de uma instigação pode ser constante ou proveniente de um atributo. Em ambos os casos, o comportamento do componente é o mesmo, sendo a única diferença a conexão dos ports de entrada. O port *i_value* pode estar conectado à saída de um atributo ou um valor literal constante; já o port *i_instigate_req* está sempre conectado à saída de uma ação. A Figura 8 mostra o hardware inferido para uma instigação, demonstrando que realmente não há consumo de recursos lógicos em sua implementação.

Figura 8. Diagrama do componente *instigation_set_integer_attribute_value*.

2.1.8 Métodos (*Methods*)

Como mencionado no item anterior, uma das funções da instigação é iniciar a execução de um método (ou seja, de uma função em C++ encapsulada por um componente do PON). Esta funcionalidade possibilita a integração de uma aplicação em PON com código imperativo tradicional. De fato, nas aplicações PON usadas como exemplo, a maior parte do código responsável por implementar as funcionalidades e requisitos da aplicação encontra-se dentro de métodos, escritos em código imperativo/orientado a objeto.

Isto só é possível na implementação em software porque o código da aplicação e o código do PON possuem a mesma natureza (i.e., trata-se de código compilado e executado em uma mesma máquina). Assim, é possível um atributo PON solicitar a execução de um método, fornecer-lhe valores como parâmetros, e (opcionalmente) aguardar o término de sua execução. Também é possível que o código do método comunique-se com outros componentes do PON, consultando-os e alterando seus estados.

Desta forma, o programador possui liberdade para realizar praticamente qualquer operação dentro de um método, independentemente de sua complexidade ou duração. Na implementação atual em C++, um método não possui valor de retorno; porém, isto é contornado devido à facilidade de interagir com os

componentes do PON dentro do próprio método (por exemplo, redefinindo o valor de um atributo ao término de sua execução).

Por outro lado, contraste-se essa liberdade de implementação com a inclusão de código arbitrário dentro de um circuito descrito em VHDL (ou em qualquer outra linguagem de descrição de hardware, ou mesmo em uma implementação em diagramas esquemáticos). Se este código arbitrário requerer acesso a outros componentes, isso só será possível através de sinais e ports específicos para tal fim. Se este código é iterativo, e requer vários ciclos de clock, um sinal *output_ready* precisará ser gerado e fornecido para o restante da aplicação.

Nada disso é impossível; porém, irá exigir do projetista um conhecimento detalhado da maneira pela qual o hardware do framework é implementado, o que invalidaria a premissa inicial da ferramenta de tradução de uma descrição em PON para uma descrição em VHDL. Por estas razões, a versão inicial da ferramenta desenvolvida não implementa o componente método, deixando este problema aberto para futuras discussões.

2.2 Sistema (Entidade *Top-Level*)

Como apresentado no item 2.1, os componentes do PON podem ser descritos usando hardware parametrizável (*generics*, em VHDL). Desta forma, uma ferramenta para gerar código VHDL para aplicações PON não precisaria necessariamente gerar o código de cada componente; bastaria instanciá-los em uma entidade de nível hierárquico superior, configurando-os como necessário para a aplicação.

Desta forma, o hardware produzido pela ferramenta consiste em uma série de arquivos em linguagem VHDL, dos quais um único arquivo corresponde à entidade de nível superior, e é gerado automaticamente pela ferramenta. A entidade top-level também é responsável por fornecer os sinais de clock e reset utilizados pelos componentes. Quando o sinal de reset é ativado, cada atributo, premissa e condição devem calcular o seu valor inicial de saída, a partir das entradas atuais.

Os demais são módulos VHDL pré-escritos e testados, os quais são instanciados pela entidade superior. Para sintetizar o hardware gerado pela ferramenta, basta compilar todos os arquivos de código VHDL dos componentes do PON, juntamente com o arquivo de saída gerado pela ferramenta. Este procedimento é descrito no item 3.3 - Síntese em FPGA.

2.3 Desenvolvimento do Código VHDL

Cada componente do PON utilizado pelas aplicações de exemplo precisou ser criado e testado individualmente, através de *testbenches*. Os testbenches são os equivalentes em hardware dos testes unitários e testes de integração empregados no desenvolvimento de software; tais testes são chamados genericamente de testes de desenvolvedor (McConnel, 2004). Os principais objetivos dos testes de desenvolvedor são:

1) Testar partes específicas do código, em isolamento ou em conjunto com o restante da aplicação.
2) Fornecer uma garantia ao desenvolvedor, permitindo afirmar que todo o código continua funcionando, após sofrer alguma alteração.

A experiência prática sugere que desenvolver um programa de computador acaompanhado do código de testes correspondente demanda menos esforço que tentar desenvolver apenas o programa, sem os testes. Assim, o desenvolvimento de cada componente do PON em VHDL foi acompanhado da elaboração de um testbench, o qual exercita e verifica as funcionalidades básicas do componente. Da mesma forma, cada uma das aplicações de exemplo é acompanhada por um testbench, o que permite comprovar que o código

gerado automaticamente realmente implementa as funcionalidades esperadas. A localização dos arquivos de código VHDL e dos respectivos testbenches é apresentada no item 2.7.1, *Organização do Código-Fonte*.

2.4 Ferramentas Computacionais Utilizadas

A proposta deste trabalho é ambiciosa em relação à amplitude dos objetivos e ao prazo de desenvolvimento limitado. Desta forma, foi fundamental escolher ferramentas computacionais (principalmente, linguagens de programação e metodologias de desenvolvimento) que possibilitassem uma alta produtividade. Entre as decisões que podem ser consideradas essenciais para a realização do trabalho, pode-se citar:

1) O uso de uma linguagem interpretada, dinamicamente tipada e extensível (no caso, Ruby);
2) O uso de práticas de desenvolvimento orientado a testes (*test-driven development*, ou TDD);
3) O uso de ambientes integrados de desenvolvimento;
4) Uso em profusão de scripts para automatizar os procedimentos mais frequentes.

As principais ferramentas utilizadas (e suas respectivas versões) são:

- Controle de versão: Subversion (SVN), versão 1.7.1.
- Linguagem de programação (software): Ruby, versão 1.9.2.
- Linguagem de descrição de hardware: VHDL, versão 2008.
- Ambiente integrado de desenvolvimento (IDE) para Software: Aptana RadRails 3.
- Ambiente integrado de desenvolvimento (IDE) para Hardware: Sigasi 2.0.
- Simulador VHDL: ModelSim Altera Starter Edition, versão 10.0c

Em aplicações Ruby, funcionalidades adicionais são acrescentadas através de bibliotecas chamadas *gems*, cuja instalação é gerenciada automaticamente pela ferramenta *Bundler*. A versão atual da aplicação utiliza a versão 1.0.21 do Bundler; as demais bibliotecas (*gems*) são instaladas automaticamente, ao executar o comando *bundle install* no diretório raiz da aplicação.

Entre os scripts desenvolvidos, os mais utilizados são:

- **script\pon2vhdl.bat**: executa a ferramenta.
- **script\run_all_tb.bat**: recompila todos os arquivos de código VHDL e executa todos os testbenches encontrados.

Entre as *gems* mais importantes, destaca-se a biblioteca RSpec (Chemlinsky, 2010), que consiste em um framework para o desenvolvimento orientado a testes (*Test-Driven Development*, ou TDD). Um exemplo de uso do RSpec será apresentado no item 2.7.2.

2.5 Descrição Inicial da Ferramenta

A ferramenta proposta consiste em um aplicativo usado em linha de comando, o qual irá obter sua entrada (descrição de uma aplicação em PON) de um arquivo, e gerar os arquivos VHDL correspondentes. Para a ferramenta proposta, uma aplicação de linha de comando é vantajosa pela facilidade de se realizar alterações em sua interface (e.g., criação de novas opções), e também pela facilidade de interagir com outras ferramentas do fluxo de desenvolvimento de hardware para FPGA.

Dessa forma, o uso básico da ferramenta consistirá em:

1) Salvar em um arquivo a descrição da aplicação PON, no formato especificado;
2) Executar a ferramenta, usando a linha de comando;
3) Obter no diretório de saída da aplicação os arquivos de código VHDL gerados.

A partir deste ponto, o usuário poderá testar o código gerado em um simulador VHDL

2.6 Formato de Entrada de Dados

Como descrito no item 3 - Uso da Ferramenta, os arquivos colocados na pasta input são automaticamente reconhecidos como possíveis entradas para a aplicação. Tais arquivos devem estar no formato YAML (YAML, 2009), o qual é descrito como "uma linguagem de serialização de dados, ao mesmo tempo legível por seres humanos e computacionalmente poderosa". Como comparação, uma das alternativas ao YAML é o formato XML (pouco legível por humanos). A Figura 9 ilustra a diferença entre os dois padrões, aplicados a uma aplicação PON:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <action1>
    <instigations>
      <inst1>
        <value type="boolean">false</value>
        <attribute>remoteControl.btsPressed</attribute>
      </inst1>
      <inst2>
        <method>gate.mtOpenGate</method>
      </inst2>
    </instigations>
  </action1>
  <action2>
    <instigations>
      <inst1>
        <value type="boolean">false</value>
        <attribute>remoteControl.btsPressed</attribute>
      </inst1>
      <inst2>
        <method>gate.mtCloseGate</method>
      </inst2>
    </instigations>
  </action2>
</actions>
<fbes>
  <gate>
    <methods>
      <mtOpenGate nil="true"></mtOpenGate>
      <mtCloseGate nil="true"></mtCloseGate>
    </methods>
  </gate>
  <remoteControl>
    <attributes>
      <btsPressed>
        <type>boolean</type>
      </btsPressed>
    </attributes>
  </remoteControl>
</fbes>
<conditions>
  <cond1>
    <premises>
      <prem1>
        <value type="boolean">true</value>
        <attribute>remoteControl.btsPressed</attribute>
        <operation>equal</operation>
      </prem1>
      <prem2>
        <value type="integer">2</value>
        <attribute>gate.GateStatus</attribute>
        <operation>equal</operation>
      </prem2>
    </premises>
    <logic>conjunction</logic>
  </cond1>
  <cond2>
    <premises>
      <prem1>
        <value type="boolean">false</value>
        <attribute>remoteControl.btsPressed</attribute>
        <operation>equal</operation>
      </prem1>
      <prem2>
        <value type="integer">0</value>
        <attribute>gate.GateStatus</attribute>
        <operation>equal</operation>
      </prem2>
    </premises>
    <logic>conjunction</logic>
  </cond2>
</conditions>
<rules>
  <rule1>
    <name>Open gate</name>
    <action>action1</action>
    <condition>cond1</condition>
  </rule1>
  <rule2>
    <name>Close gate</name>
    <action>action2</action>
    <condition>cond2</condition>
  </rule2>
</rules>
<system>
  <inputs>remoteControl.btsPressed</inputs>
  <outputs>Gate.state</outputs>
</system>
```

(a) Em formato XML.

```yaml
system:
  name: wire_application
  inputs: [ attribute_input ]
  outputs: [ attribute_output ]

fbes:
  fbe_wire:
    attributes: [ attribute_input, attribute_output ]

attributes:
  attribute_input:
    vhdl_type: boolean
  attribute_output:
    vhdl_type: boolean

premises:
  premise_input_high:
    attribute: attribute_input
    operation: EQUAL_TO
    value: true
  premise_input_low:
    attribute: attribute_input
    operation: EQUAL_TO
    value: false

conditions:
  condition_any_change:
    logic: disjunction
    premises: [ premise_input_high, premise_input_low ]

actions:
  action_copy_input:
    instigations: [ instigation_copy_input ]

instigations:
  instigation_copy_input:
    source_type: attribute
    source: attribute_input
    target: attribute_output

rules:
  rule_copy_input:
    condition: condition_any_change
    action: action_copy_input
```

(b) Em formato YAML.

Figura 9. Descrições textuais equivalentes para a aplicação *wire*.

Pela maior clareza e melhor relação sinal/ruído (menor número de caracteres por informação), o formato YAML foi selecionado para os arquivos de entrada da aplicação. Em YAML, a indentação define a estrutura hierárquica do documento, evitando a necessidade de tags ou marcações adicionais, o que torna a descrição mais limpa.

Ao invés de tentar escrever uma especificação formal para formato de entrada de dados, a Figura 10 mostra um exemplo de arquivo de entrada para a ferramenta, devidamente comentado para que possa servir como modelo para futuras aplicações. O exemplo da figura descreve a aplicação *electronic_gate*, a qual será detalhada no item 4.1.2 - Aplicação *electronic gate*.

```
system:                          # as informações no bloco system referem-se ao sistema como um todo
  name: electronic_gate                # nome do sistema; corresponde ao nome da entidade VHDL
  inputs: [ attribute_button_pressed ]  # nome das entradas do sistema; devem ser nomes de atributos
  outputs: [ attribute_gate_state ]     # nome das saídas do sistema; devem ser nomes de atributos

fbes:                            # o bloco fbes descreve os elementos da base de fatos
  fbe_gate:                              # nome do primeiro FBE
    attributes: [ attribute_gate_state ]       # atributos do primeiro FBE
  fbe_remote_control:                    # nome do segundo FBE
    attributes: [ attribute_button_pressed ]   # atributos do segundo FBE

attributes:                      # o bloco attributes descreve os atributos do sistema
  attribute_button_pressed:              # nome do primeiro atributo
    vhdl_type: boolean                   # tipo (em VHDL) do primeiro atributo
  attribute_gate_state:                  # nome do segundo atributo
    vhdl_type: integer                   # tipo (em VHDL) do segundo atributo

premises:                        # o bloco premises descreve as premissas do sistema
  premise_button_pressed:                # nome da primeira premissa
    attribute: attribute_button_pressed  # atributo associado à premissa
    operation: EQUAL_TO                  # operador relacional usado pela premissa (Tabela 1)
    value: true                          # valor constante usado na comparação
  premise_gate_closed:                   # nome da segunda premissa
    attribute: attribute_gate_state      # atributo associado à premissa
    operation: EQUAL_TO                  # operador relacional usado pela premissa (Tabela 1)
    value: 0                             # valor constante usado na comparação
  premise_gate_open:                     # nome da terceira premissa
    attribute: attribute_gate_state      # atributo associado à premissa
    operation: EQUAL_TO                  # operador relacional usado pela premissa (Tabela 1)
    value: 2                             # valor constante usado na comparação

conditions:                      # o bloco conditions descreve as condições do sistema
  condition_command_open:                                  # nome da primeira condição
    premises: [ premise_button_pressed, premise_gate_closed ] # premissas associadas à condição
    logic: CONJUNCTION                                     # conexão lógica aplicada às premissas
  condition_command_close:                                 # nome da segunda condição
    premises: [ premise_button_pressed, premise_gate_open ]  # premissas associadas à condição
    logic: CONJUNCTION                                     # conexão lógica aplicada às premissas

rules:                           # o bloco rules descreve as regras do sistema
  rule_open_gate:                        # nome da primeira regra
    condition: condition_command_open    # nome da condição associada à primeira regra
    action: action_open_gate             # nome da ação associada à primeira regra
  rule_close_gate:                       # nome da segunda regra
    condition: condition_command_close   # nome da condição associada à segunda regra
    action: action_close_gate            # nome da ação associada à segunda regra

actions:                         # o bloco actions descreve as ações do sistema
  action_open_gate:                      # nome da primeira ação
    instigations: [ instigation_open_gate ]    # instigações associadas à primeira ação
  action_close_gate:                     # nome da segunda ação
    instigations: [ instigation_close_gate ]   # instigações associadas à segunda ação

instigations:                    # o bloco instigations descreve as instigações do sistema
  instigation_open_gate:                 # nome da segunda instigação
    source_type: constant                # origem do valor (constant ou attribute)
    source_vhdl_type: integer            # tipo (em VHDL) do valor constante
    source_value: 2                      # valor constante
    target: attribute_gate_state         # destino do valor (nome de um atributo)
  instigation_close_gate:                # nome da primeira instigação
    source_type: constant                # origem do valor (constant ou attribute)
    source_vhdl_type: integer            # tipo (em VHDL) do valor constante
    source_value: 0                      # valor constante
    target: attribute_gate_state         # destino do valor (nome de um atributo)
```

Figura 10. Exemplo de descrição de uma aplicação PON no formato compreendido pela ferramenta.

À medida que o framework for se consolidando, uma documentação mais formal será mais útil e poderá substituir o uso de exemplos como o da Figura 10.

2.7 Descrição do Framework

A aplicação desenvolvida consiste em um *framework* para geração de hardware em VHDL, a partir de aplicações descritas em PON. Em aplicações de software, um *framework* diferencia-se de uma simples biblioteca por apresentar certas características, tais como:

1) Em um framework, a sequência principal de execução não é ditada por código da aplicação, e sim pelo próprio framework;
2) Um framework pode ser estendido pelo usuário, ou especializado por código de aplicação descrevendo comportamentos mais específicos;
3) O código do framework, em geral, não pode ser modificado. Os usuários podem estendê-lo, mas não modificar o seu código-fonte.

A ferramenta desenvolvida possui todas as características acima. Desta forma, ao longo do documento, termos como "ferramenta proposta" ou "framework desenvolvido" serão usados indistintamente para se referir ao software resultante deste trabalho.

2.7.1 Organização do Código-Fonte

A organização da aplicação em diretórios é inspirada no framework de desenvolvimento web chamado *Rails*. A existência de uma estrutura padrão facilita o manuseio do código-fonte entre desenvolvedores. A estrutura das aplicações Rails é descrita em (Ruby, 2011).

```
pon2vhdl            -- pasta raiz do projeto
  app               -- código-fonte e arquivos de suporte da aplicação
    config          -- configuração da aplicação Ruby
    framework       -- arquivos de código usados apenas pelo framework
    helpers         -- arquivos de ajuda (Utility pattern)
    models          -- modelos (classes de objetos) da aplicação
    templates       -- arquivos vhdl.erb (código VHDL intercalado com Ruby)
    vhdl            -- código VHDL instanciado pelo arquivo top-level
  doc               -- documentação da aplicação
  input             -- arquivos de entrada (YAML)
  modelsim          -- projetos e arquivos do simulador ModelSim
  output            -- arquivos de saída (código VHDL)
  quartus           -- projetos e arquivos da ferramenta Quartus II
  script            -- scripts para automatizar procedimentos comuns
  spec              -- código-fonte dos testes unitários e de integração
```

Figura 11. Estrutura de diretórios do framework.

Os arquivos na pasta *models* seguem a organização tradicional de aplicações orientadas a objeto (OO). Os arquivos na pasta *templates* possuem uma característica única: são arquivos de código VHDL, intercalados com trechos de código em Ruby. Quando renderizados, o código Ruby é executado, e o resultado é código VHDL pronto para ser compilado e sintetizado. A Figura 12 mostra um *template* para instanciar um atributo booleano. O código Ruby se encontra em vermelho, entre as marcações **<%** e **%>**; o restante do arquivo é transcrito literalmente, e corresponde a código em linguagem VHDL.

```
-- instantiate attribute '<%= attribute.vhdl_name %>'
<%= attribute.vhdl_name %>: entity work.boolean_attribute(behavior)
port map(
  clock => clock,
  reset => reset,
  i_value => <%= attribute.port('i_value').signal_name %>,
  i_changed => <%= attribute.port('i_changed').signal_name %>,
  o_value => <%= attribute.port('o_value').signal_name %>,
  o_changed => <%= attribute.port('o_changed').signal_name %>
);
```

Figura 12. *Template* para instanciar um atributo booleano.

A pasta *input* é o local onde devem ser colocados os arquivos de entrada para a aplicação. Após executar a ferramenta, a pasta *output* conterá o código VHDL gerado.

As pastas *quartus* e *modelsim* contêm dados específicos das respectivas ferramentas de compilação e síntese. Dois projetos do Quartus estão incluídos, e correspondem às aplicações *wire* e *electronic_gate*.

Finalmente, os arquivos na pasta *spec* contêm código de teste, incluindo testes unitários e testes de integração, destinados a garantir o correto funcionamento da aplicação.

2.7.2 Desenvolvimento da Ferramenta

Como apresentado no item 2.1, o framework do PON é composto por diversos componentes e subcomponentes; normalmente, uma aplicação PON utiliza apenas uma parte destes. Desta forma, implementar todos os componentes logo de início não possibilitaria obter resultados rápidos. Além disso, a própria criação da ferramenta constitui um objeto de investigação, de modo que o conhecimento obtido ao longo do processo pode e deve ser usado continuamente, para melhorar a qualidade do código já escrito e também dos próximos módulos a serem desenvolvidos. Por estes motivos, optou-se por uma estratégia incremental, consentindo em selecionar algumas aplicações de exemplo, e desenvolver o mínimo de código necessário para se gerar automaticamente o código VHDL correspondente.

Quando a ferramenta é executada, o processo de geração do código VHDL ocorre em três etapas, as quais serão descritas a seguir. De maneira análoga, o desenvolvimento do software foi realizado em sequência e obedecendo às mesmas etapas.

1) *Extrair informações do arquivo de entrada*: a conversão do arquivo de texto escrito em YAML para um objeto da classe Hash em Ruby é trivial, usando-se a biblioteca padrão. Porém, os componentes do PON identificados devem ser conectados como descrito no arquivo de entrada. Assim, os dois passos principais desta etapa são identificar os componentes na descrição YAML e conectá-los obedecendo ao arquivo de entrada.

2) *Representar a aplicação segundo um modelo orientado a objetos*: numa implementação em software, cada componente do PON possui seus métodos e atributos próprios. Na versão em hardware, também é necessário especializar a instanciação e a conexão dos componentes. Para tanto, foi criada uma classe especializada para cada componente do PON. Todas as classes especializadas derivam da classe base genérica chamada *Component*.

3) *Geração do código VHDL*: cada componente identificado precisa ser instanciado na entidade top-level que descreve o sistema. Isto é feito através de templates, que consistem em trechos de código VHDL (estático) intercalados com código em Ruby (executado no momento da instanciação) (ver exemplo na Figura 12). Além disto, é necessário criar os sinais que serão conectados aos ports de cada componente.

Após a geração do código, um arquivo VHDL é produzido no diretório *output*, com o mesmo nome da aplicação de entrada.

O desenvolvimento da ferramenta foi auxiliado pela criação de testes unitários e de integração. A Figura 13 mostra a execução de uma sequência de testes, a qual exercita o código desenvolvido e testa os valores de saída produzidos, comparando-os com os valores esperados. Como pode ser visto na figura, a ferramenta de testes (RSpec) fornece informações valiosas sobre os testes que falharem, indicando o valor de saída produzido, o valor esperado e o número da linha de código onde ocorreu o erro.

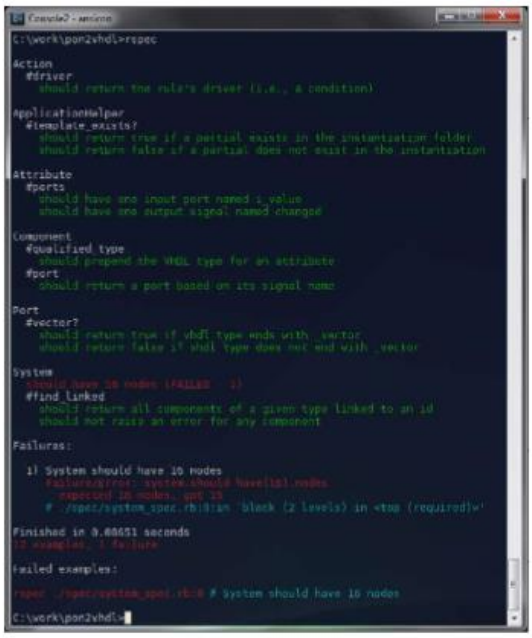

Figura 13. Execução dos testes unitários e de integração.

2.7.3 Estendendo o Framework

Para que a estratégia de desenvolvimento incremental adotada faça sentido, é necessário que haja uma maneira prática de aumentar a funcionalidade da ferramenta. A organização do código como um framework facilita esta atividade. Algumas observações úteis relativas à extensibilidade do framework são apresentadas a seguir:

- **Novos componentes**: os componentes são instanciados na entidade top-level através do método *Renderer#render*, o qual irá buscar no diretório *app/templates/instantiation* o template correspondente ao componente especificado. Assim, para adicionar um novo componente especializado (por exemplo, um atributo de ponto flutuante) bastaria acrescentar o template (*_float_attribute.vhdl.erb*) no diretório *app/templates/instantiation*, e o arquivo vhdl (*float_attribute.vhd*) no diretório *vhdl/synth*. Nenhuma alteração no código-fonte da aplicação é necessária.

- **Entidade top-level**: o código VHDL para a entidade top-level que descreve o sistema encontra-se no template *system.vhdl.erb* (Figura 14). De uma maneira geral, pode-se dizer que a única tarefa da ferramenta desenvolvida é renderizar este template. Qualquer modificação relativa ao código VHDL gerado deverá ser feita neste arquivo.

- **Chaves (Keys) na descrição YAML são convertidas em métodos nos Componentes do PON**: examinando a descrição YAML da Figura 10, observa-se que cada nome de componente é seguido por um mapeamento (*map*, ou *hash*) contendo nome de atributos (chaves, ou *keys*) e seus respectivos valores. Através de metaprogramação, todas as chaves da descrição textual serão convertidas em métodos dos componentes. Assim, um novo atributo pode ser criado na forma de texto e utilizado no template correspondente, sem a necessidade de alterar código-fonte.

```
use work.enumerated_types_pkg.all;
use work.boolean_vector_pkg.all;

entity <%= @system.name %> is
  port (
<% @system.ports.collect do |port| -%>
<%= "    #{port.port_name}: #{port.vhdl_direction} #{port.vhdl_type};" %>
<% end -%>
    clock: in bit;
    reset: in bit
  );
end entity;

architecture structural of <%= @system.name %> is
  -- auto-generated signals (from port definitions)
<% @system.nodes.each do |component| -%>
  <%= "-- signals for component '#{component.name}' of type #{component.type}" %>
  <% component.ports.each do |port| -%>
    <%= "signal #{port.signal_name}: #{port.vhdl_type};" %>
  <% end -%>
<% end -%>

begin
  ----------------------------------------------------------------------------
  -- Attributes
  ----------------------------------------------------------------------------
<% @system.attributes.each do |attribute| %>
<%= render attribute, binding, :instantiation %>
<% end -%>
  ----------------------------------------------------------------------------
  -- Premises
  ----------------------------------------------------------------------------
<% @system.premises.each do |premise| %>
<%= render premise, binding, :instantiation %>
<% end -%>
...
  ----------------------------------------------------------------------------
  -- Connections
  ----------------------------------------------------------------------------
  -- connect premise inputs with attribute outputs
<% @system.premises.each do |premise| -%>
<%= auto_connect_with_drivers(premise) %>
<% end -%>
  -- connect condition inputs with premise outputs
<% @system.conditions.each do |cond| -%>
<%= auto_connect_with_drivers(cond) %>
<% end -%>
 ...
  -- connect system ports with attribute ports
<% @system.ports.each do |port| -%>
<% if port.direction == :input -%>
<%= "  #{port.wants_port_named} <= #{port.port_name};" %>
<% else -%>
<%= "  #{port.port_name} <= #{port.wants_port_named};" %>
<% end -%>
<% end -%>
end architecture;
```

Figura 14. Trechos de código do arquivo system.vhdl.erb.

3 Uso da Ferramenta

3.1 Descrição da Aplicação PON

Para ser utilizada pela ferramenta, uma aplicação PON precisa ser descrita em termos de seus componentes fundamentais (atributos, premissas, condições, etc.). Tais componentes devem ser descritos em um arquivo de texto no formato YAML, como mostrado na Figura 10. O arquivo deve possuir a extensão .yml e deve ser colocado no diretório *input*.

3.2 Usando a Ferramenta de Linha de Comando

Partindo de uma aplicação PON descrita em um arquivo YAML na pasta `input`, o uso da ferramenta consiste em:

1. Entrar na pasta raiz da aplicação (e.g., `cd /pon2vhdl`).
2. Executar a ferramenta (digitar `script/pon2vhdl.bat`).
3. O arquivo VHDL de saída estará na pasta `/pon2vhdl/output`.

Ao ser executada, a aplicação apresentará um menu de seleção, listando os arquivos encontrados no diretório (Figura 15).

```
C:\work\pon2vhdl>script\pon2vhdl.bat
PON2VHDL -- (C) 2001 Ricardo Jasinski
(1) gate.yml
(2) gate_commented.yml
(3) wire.yml
Select an input file: 1
```

Figura 15. Menu de seleção exibido ao executar a ferramenta pon2vhdl.

Também é possível executar a ferramenta fornecendo o nome do arquivo de entrada como argumento, evitando a apresentação do menu de seleção (e.g., `script/pon2vhdl.bat input/wire.yml`).

3.3 Síntese em FPGA

Para sintetizar o código VHDL gerado em uma FPGA, basta realizar os seguintes passos:

1. Criar um novo projeto do Quartus.
2. Adicionar ao projeto todos os arquivos da pasta vhdl/synth (menu *Project* > *Add/Remove Files in Project*...)
3. Adicionar ao projeto o arquivo da pasta output, correspondente ao sistema gerado.
4. Compilar e sintetizar a aplicação (menu *Processing* > *Start Compilation*).

4 Resultados

4.1 Descrição das Aplicações de Exemplo

A ferramenta desenvolvida foi utilizada para implementar duas aplicações de exemplo: um simples "fio" (aplicação *wire*), o qual reproduz em sua saída os estímulos aplicados à sua entrada, e um portão eletrônico, (aplicação *electronic_gate*) o qual controla a abertura e fechamento de um portão a partir de um controle remoto.

4.1.1 Aplicação wire

A aplicação *wire* foi concebida e selecionada por ser a mais simples possível: deseja-se reproduzir o comportamento de um único fio, o qual repete em sua saída o valor lógico aplicado a sua entrada. Quanto à avaliação da ferramenta desenvolvida, esta aplicação serve como linha de base para estimar a menor quantidade de recursos lógicos consumida por uma aplicação. Em outras palavras, como a funcionalidade que se deseja implementar é a de um simples fio, qualquer hardware necessário pode ser considerado como *overhead*.

A Figura 16 mostra a descrição da aplicação *wire* no formato YAML:

```yaml
system:
  name: wire_application
  inputs: [ attribute_input ]
  outputs: [ attribute_output ]

fbes:
  fbe_wire:
    attributes: [ attribute_input, attribute_output ]

attributes:
  attribute_input:
    vhdl_type: boolean
  attribute_output:
    vhdl_type: boolean

premises:
  premise_input_high:
    attribute: attribute_input
    operation: EQUAL_TO
    value: true
  premise_input_low:
    attribute: attribute_input
    operation: EQUAL_TO
    value: false

conditions:
  condition_any_change:
    logic: disjunction
    premises: [ premise_input_high, premise_input_low ]

actions:
  action_copy_input:
    instigations: [ instigation_copy_input ]

instigations:
  instigation_copy_input:
    source_type: attribute
    source: attribute_input
    target: attribute_output

rules:
  rule_copy_input:
    condition: condition_any_change
    action: action_copy_input
```

Figura 16. Descrição YAML da aplicação wire.

O código VHDL gerado pela ferramenta desenvolvida para a aplicação *wire* pode ser visto no Anexo I – Código VHDL Gerado para as Aplicações de Exemplo.

4.1.2 Aplicação *electronic gate*

A aplicação *electronic_gate* foi selecionada por ser uma aplicação comum em hardware e de fácil compreensão. Trata-se de um sistema no qual um botão controla a abertura e o fechamento de um portão, acionando um motor através de suas saídas. Outro motivo para a escolha desta aplicação foi o fato de já existir uma implementação em software usando o PON. Assim como na versão em software, o sistema descrito em hardware possui apenas dois estados possíveis para o portão, aberto (*gate_state* = 2) ou fechado (*gate_state* = 0).

A Figura 17 mostra a descrição da aplicação *electronic_gate* no formato YAML. O código VHDL gerado automaticamente pela ferramenta pode ser visto no Anexo I – Código VHDL Gerado para as Aplicações de Exemplo.

```yaml
system:
  name: electronic_gate
  inputs: [ attribute_button_pressed ]
  outputs: [ attribute_gate_state ]

fbes:
  fbe_gate:
    attributes: [ attribute_gate_state ]
  fbe_remote_control:
    attributes: [ attribute_button_pressed ]

attributes:
  attribute_button_pressed:
    vhdl_type: boolean
  attribute_gate_state:
    vhdl_type: integer

premises:
  premise_button_pressed:
    attribute: attribute_button_pressed
    operation: EQUAL_TO
    value: true
  premise_gate_closed:
    attribute: attribute_gate_state
    operation: EQUAL_TO
    value: 0
  premise_gate_open:
    attribute: attribute_gate_state
    operation: EQUAL_TO
    value: 2

conditions:
  condition_command_open:
    logic: CONJUNCTION
    premises: [ premise_button_pressed, premise_gate_closed ]
  condition_command_close:
    logic: CONJUNCTION
    premises: [ premise_button_pressed, premise_gate_open ]

rules:
  rule_open_gate:
    condition: condition_command_open
    action: action_open_gate
  rule_close_gate:
    condition: condition_command_close
    action: action_close_gate

actions:
  action_open_gate:
    instigations: [ instigation_open_gate ]
  action_close_gate:
    instigations: [ instigation_close_gate ]

instigations:
  instigation_open_gate:
    source_type: constant
    source_vhdl_type: integer
    source_value: 2
    target: attribute_gate_state
  instigation_close_gate:
    source_type: constant
    source_vhdl_type: integer
    source_value: 0
    target: attribute_gate_state
```

Figura 17. Descrição YAML da aplicação electronic_gate.

4.2 Simulações Funcionais

As simulações funcionais foram realizadas com o software ModelSim Altera SE. A Figura 18 mostra o resultado de uma simulação para a aplicação *wire*, na qual o sinal de entrada (atributo *attribute_input*) sofre uma alteração, e após certa latência o sinal de saída (atributo *attribute_output*) é alterado.

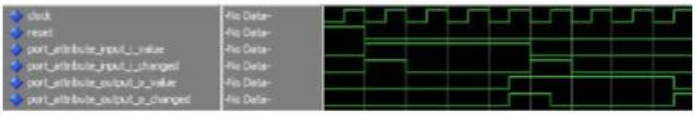

Figura 18. Simulação funcional da aplicação *wire* (apenas ports de entrada e saída).

Pela latência observada (4 ciclos de clock), pode-se concluir que o hardware inferido contém 4 flifp-flops em série. Analisando o circuito resultante através da ferramenta RTL Viewer, observa-se que os flip-flops encontram-se no atributo *input*, na premissa *input_high/low*, na condição *any_change* e no atributo *output*. Ou seja, cada estágio registrado da cadeia de notificações contribuiu com um atraso igual a um ciclo de clock.

A Figura 19 mostra os resultados para a mesma simulação da aplicação *wire*, porém incluindo todos os sinais relevantes. O grupo *notifications* evidencia o funcionamento da cadeia de notificações; os sinais internos de cada componente do PON também são apresentados na figura.

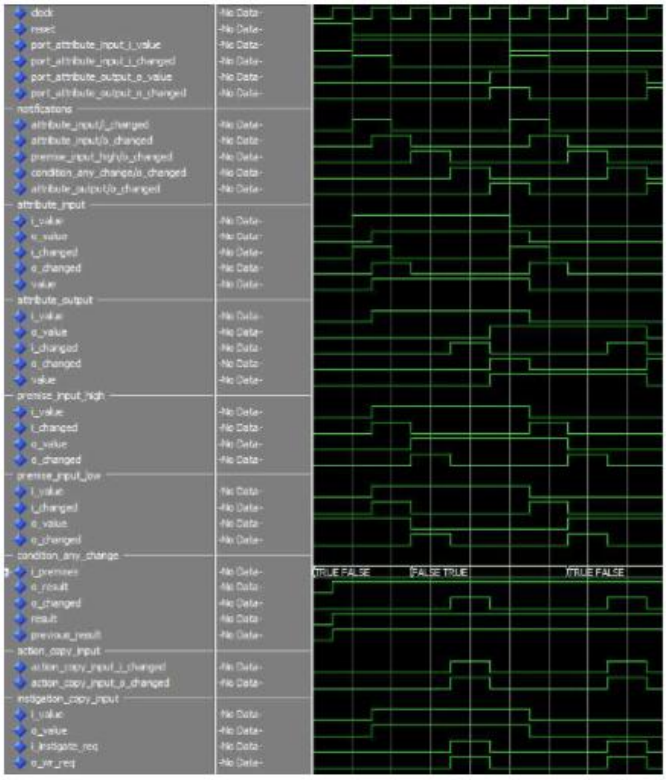

Figura 19. Simulação funcional da aplicação *wire* (todos os sinais relevantes).

A Figura 20 mostra o resultado de uma simulação para a aplicação *electronic_gate*, na qual o sinal de entrada (atributo *button_pressed*) sofre uma mudança, e após alguma latência o sinal de saída (atributo

gate_state) é alterado. Assim como na versão em software da aplicação, um valor igual a 0 para o atributo *gate_state* significa que o portão está fechado, e um valor igual a 2 indica que o portão está aberto.

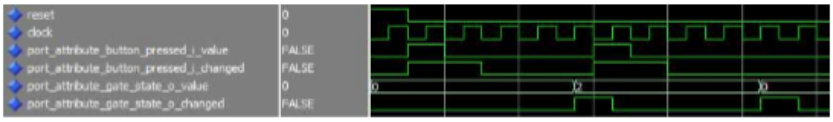

Figura 20. Simulação funcional da aplicação *electronic_gate* (apenas ports de entrada e saída).

Assim como na aplicação *wire*, a latência observada (4 ciclos de clock) indica que o hardware inferido contém 4 flifp-flops em série. Analisando o circuito resultante e considerando o caso do portão abrindo, observa-se que os flip-flops encontram-se no atributo *atribute_button_pressed*, na premissa *premise_button_pressed*, na condição *command_open* e no atributo *gate_state*. Novamente, cada estágio registrado da cadeia de notificações contribui com um atraso igual a um período de clock.

A Figura 21 mostra os resultados para a mesma simulação da aplicação *electronic_gate*, porém incluindo todos os sinais relevantes. O grupo *notifications* evidencia o funcionamento da cadeia de notificações; os sinais internos de cada componente do PON também são apresentados na figura.

4.3 Implementação em FPGA

Ambas as aplicações foram compiladas para FPGAs da família Cyclone IV, da Altera (dispositivo EP4CGX15BF14C6), utilizando o software Quartus II versão 11.1. A Tabela 3 mostra os resultados da compilação, bem como a latência observada para cada circuito. Quanto à quantidade de recursos lógicos utilizados, nenhuma das aplicações consumiu um número alto de elementos lógicos (LEs) em relação à capacidade das atuais FPGAs; em ambos os casos, foi utilizado menos de 1% do menor dispositivo da família Cyclone IV. Entretanto, se considerarmos o grau de simplicidade das aplicações, a quantidade de recursos utilizados pode ser considerada alta. No caso da aplicação *wire*, foram necessários 17 LEs, para um comportamento que consiste basicamente em atrasar um sinal de entrada em 4 pulsos de clock. No caso da aplicação *electronic_gate*, foram usados 23 LEs, para implementar uma funcionalidade que certamente poderia ser atingida com poucos elementos em uma implementação manual (certamente, menos de 10 LEs).

Tabela 3. Resultados da compilação para FPGAs da família Cyclone IV.

Aplicação	Elementos Lógicos	Flip-Flops	Pinos	Fmax	Latência
wire	17	10	6	458,9 MHz (*)	4 ciclos
electronic_gate	23	11	37	373,7 MHz (*)	4 ciclos

(*) restricted to 250 MHz due to minimum period restriction (max I/O toggle rate)

É interessante observar o bom trabalho realizado pela ferramenta de síntese, comprovado pela pequena diferença no número de registradores utilizados pelas duas aplicações. Deve-se lembrar que a aplicação *wire* possui dois atributos booleanos, enquanto a aplicação *electronic_gate* possui um atributo booelano e um do tipo inteiro. Como a largura do número inteiro não foi restringida no código VHDL, claramente a ferramenta de síntese otimizou o circuito resultante; embora no diagrama apresentado no RTL Viewer o registrador seja mostrado com a largura de 32 bits (Figura 3), o número de flip-flops utilizados (11) claramente indica a otimização realizada.

Além dos resultados numéricos, é interessante analisar o diagrama de blocos gerado pela ferramenta RTL Viewer, para confirmar que todos os componentes foram instanciados corretamente. A Figura 22 mostra os diagramas gerados para ambas as aplicações.

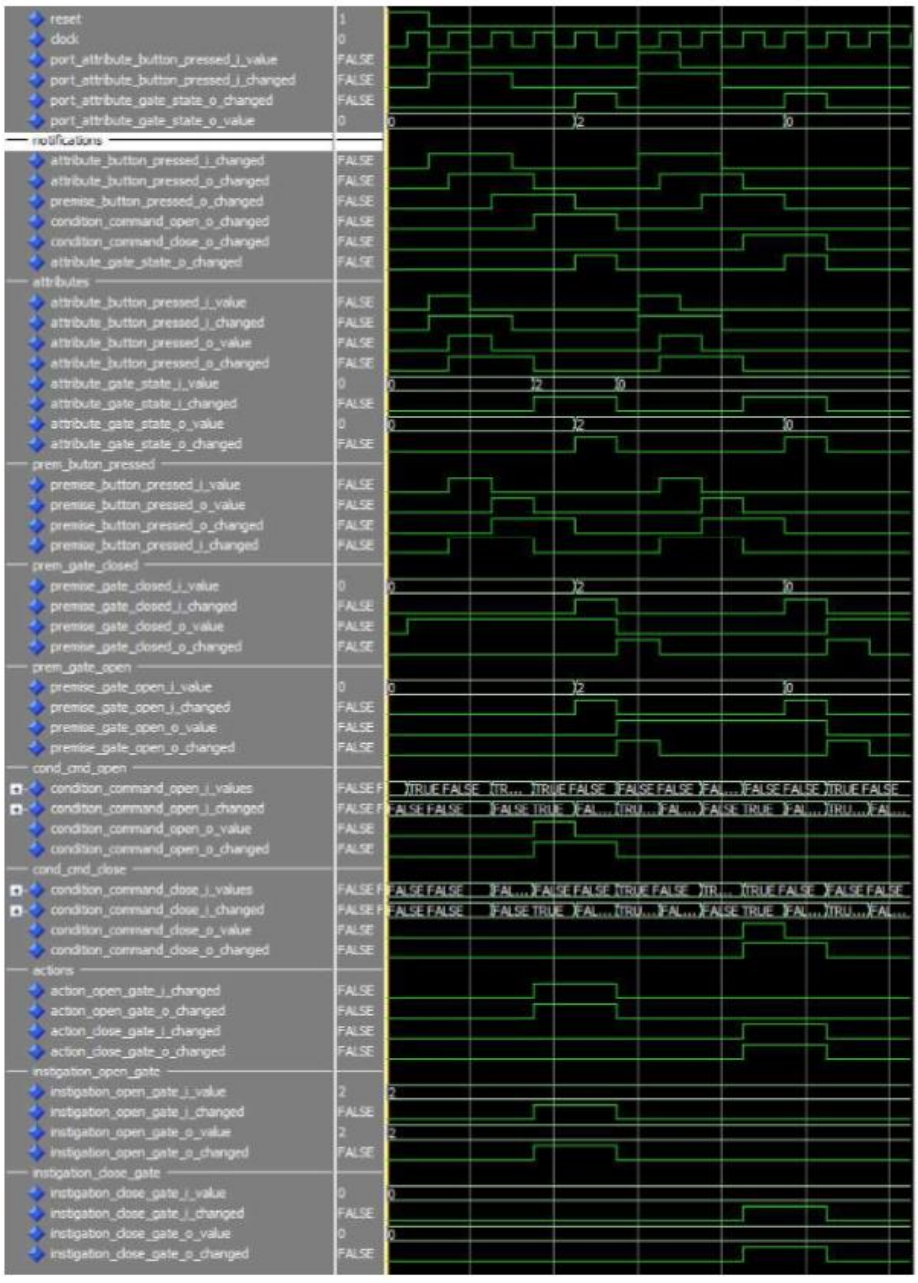

Figura 21. Simulação funcional da aplicação *electronic_gate* (todos os sinais relevantes).

(a) Aplicação *wire*. (b) Aplicação *electronic_gate*.

Figura 22. Diagramas de blocos produzidos pela ferramenta *RTL Viewer*.

4.4 Estatísticas do Código

Durante o desenvolvimento do framework, foi necessário desenvolver código em diversas linguagens de programação, principalmente em VHDL, Ruby e YAML. A Tabela 4 mostra o número de linhas de código geradas durante o desenvolvimento da ferramenta; as estatísticas foram obtidas como uso da ferramenta CLOC (Count Lines of Code) (Sourceforge, 2012).

Tabela 4. Número de linhas de código.

```
http://cloc.sourceforge.net v 1.55  T=1.0 s (61.0 files/s, 2939.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
VHDL                            19            230            112           1183
Ruby                            32            178             39            960
YAML                             3             23              0            152
DOS Batch                        4             15              0             36
XML                              2              0              0             10
Bourne Shell                     1              0              0              1
-------------------------------------------------------------------------------
SUM:                            61            446            151           2342
-------------------------------------------------------------------------------
```

De acordo com estatísticas apresentadas em (McConnel, 2001), para projetos entre 1k e 10k linhas de código a produtividade média de um programador é de 2000 a 2500 linhas de código por ano. Desta forma, pode-se estimar grosseiramente o esforço de desenvolvimento até agora em aproximadamente 1 homem-ano.

Analisando a quantidade de linhas de código escritas exclusivamente com o propósito de teste (i.e., testbenches em VHDL e testes unitários ou de integração em Ruby), o código de teste corresponde a aproximadamente 30% do total, com uma relação código de aplicação/código de testes de aproximadamente 1/0.5. Em aplicações comerciais, é comum trabalhar com a métrica de 1/1 a 1/4. Desta forma, em se continuando o desenvolvimento do framework, uma das primeiras atividades necessárias seria aumentar a cobertura dos testes existentes.

4.5 Limitações da Ferramenta

Como mencionado ao longo do item 2, algumas simplificações foram adotadas para que o framework pudesse ser desenvolvido dentro do tempo disponível. As principais limitações são descritas a seguir.

Primeiramente, não há resolução de conflitos para a execução das regras; a ação correspondente a cada regra é executada assim que a sua condição se torna verdadeira. Em teoria, isto poderia ocasionar conflitos caso duas regras tentassem acessar um mesmo componente. Para contornar este problema, assumiu-se que o conjunto de regras fornecido como entrada para o sistema seria isento de conflitos, ou seja, a aplicação seria segura "por *design*". Além disso, foi elaborado um sistema simples de prioridades, no qual a primeira regra (declarada mais próxima ao início do arquivo YAML) tem prioridade se várias regras tentarem sobrescrever o valor de um mesmo atributo. Isto pode ser observado no mux implementado pela ferramenta no diagrama da Figura 22(b).

Uma segunda simplificação adotada é a ausência de suporte à execução de instigações em série. Como mencionado no item 2.1.7, uma ação deveria possuir um parâmetro configurável, indicando se as instigações devem ser executadas simultaneamente (em paralelo) ou uma após a outra (em série). Para reduzir a complexidade num primeiro momento, apenas a versão paralela foi implementada. Entretanto, a modificação para executar as *instigations* em paralelo é relativamente simples, bastando conectar o sinal de *output_ready* de uma instigation ao sinal *instigate_request* da próxima *instigation*.

Uma terceira simplificação é a ausência de suporte ao componente método do PON, o qual encapsula um trecho de código que pode ser executado por uma instigação. Esta funcionalidade é facilmente implementada em linguagens de programação interpretadas ou compiladas, nas quais um ponteiro para uma função pode ser obtido e chamado posteriormente para execução. Em VHDL, no entanto, tal recurso inexiste. Além do mais, se considerarmos que um componente método poderia ser implementado como um bloco distinto de hardware, contendo ports de entrada e saída e operando em sincronia com o clock do sistema, seria necessário planejar cuidadosamente a sua integração com o restante do circuito, de modo a conectá-lo aos demais componentes do sistema gerado. Por estes motivos, a versão inicial da ferramenta desenvolvida não implementa o componente método, deixando este problema aberto para futuras discussões. Todavia, como foi demonstrado, é possível criar aplicações úteis que não dependem deste tipo de componente.

Finalmente, quanto ao componente premissa, foram implementadas versões especializadas apenas para comparar o valor de um atributo com um valor constate (booleano ou inteiro). Numa implementação mais completa, seria necessário suportar também premissas que comparassem os valores de dois atributos entre si.

5 Conclusões e Trabalhos Futuros

5.1 Conclusões

De uma maneira geral, o desenvolvimento do framework ocorreu como inicialmente planejado. Seguindo as técnicas de desenvolvimento incremental, uma versão preliminar do framework foi concebida, a qual foi capaz de gerar o código VHDL para a aplicação *wire*. A partir de então, o framework foi estendido e generalizado para suportar também a aplicação *electronic_gate*. Em teoria, qualquer aplicação que utilize os mesmos componentes empregados nessas aplicações também poderá ser implementada pelo framework.

No desenvolvimento, ressalta-se o suporte da linguagem de programação escolhida (Ruby) à geração de documentos parametrizáveis, através dos templates escritos em *Embedded Ruby*; todos os arquivos VHDL foram gerados desta forma. Destaca-se também o suporte às ferramentas de teste (RSpec) e interpretação de arquivos YAML, os quais foram essenciais ao desenvolvimento da ferramenta.

Em contrapartida, foi sentida a falta na linguagem VHDL de suporte a tipos genéricos de dados. A maioria das linguagens estaticamente tipadas (e.g., Ada, Pascal, C++, C#, Java, Eiffel, Haskell, etc.) possibilita o reuso de código através de funções que podem receber um objeto genérico como argumento. Por exemplo, uma mesma função matemática poderia ser usada com dados dos tipos *integer*, *float* e *real*; o programador escreveria o código uma única vez, e o mesmo seria especializado pelo compilador. É importante notar que não há nenhum motivo inato para não existir esta funcionalidade em VHDL (por exemplo, em ADA, *generic types* são suportados desde 1983). Esta funcionalidade eliminaria a necessidade de se criar diversos módulos desnecessariamente no framework desenvolvido (por exemplo, os componentes *boolean_attribute*, *integer_attribute* e *float_attribute* poderiam todos empregar o mesmo código-fonte). Atualmente, tais módulos possuem um alto nível de repetição de código entre si, o que é uma característica de código de baixa qualidade.

Finalmente, é interessante observar que, dos oito componentes do PON, três não incorrem em hardware (regras, ações e instigações), um é supérfluo (elementos da base de fatos) e outro não pode ser

implementado (método). Isto contribuiu para o baixo consumo de recursos de hardware observado nas aplicações implementadas (média de 3,33 elementos lógicos por componente do PON).

5.2 Trabalhos Futuros

Como mencionado no item 4.5 - Limitações da Ferramenta, várias simplificações foram adotadas nesta primeira versão do framework. Desta forma, uma das primeiras atividades a serem abordadas é a resolução de tais limitações, como sugerido no item 4.5.

Como apresentado no item 4.4 - Estatísticas do Código, a cobertura de testes da aplicação está abaixo do recomendável. Como a base de código já atingiu um tamanho a partir do qual haverá dificuldades em se prosseguir sem uma cobertura de testes adequada, uma das atividades mais importantes será a ampliação da cobertura de testes.

Outro ponto para melhoria são as mensagens de erro apresentadas ao usuário, quando um arquivo de entrada possui alguma informação inválida. Por exemplo, a mensagem de erro:

```
in `method_missing': attribute (NoMethodError)
```

poderia ser traduzida como:

```
Atributo não especificado para premissa 'premise_input_high'.
```

Isto ajudaria o usuário a corrigir o arquivo de entrada, até adequá-lo ao formato reconhecido pelo framework.

Para evitar uma complexidade desnecessária, todos os componentes em hardware tiveram seus ports nomeados de uma forma padrão (i.e., *i_value*, *o_ value*, *i_changed* e *o_changed*). Agora que o framework já está funcional, pode-se considerar a possibilidade de usar nomes mais específicos (e.g., *i_execute* para iniciar a execução das regras, *i_wr_req* para redefinir o valor de atributos, etc.).

Visando eliminar duplicação de código, pode ser possível eliminar a especialização das premissas de acordo com a origem dos valores comparados. Atualmente, são necessários módulos distintos para comparar um atributo com um valor constante, ou para comparar dois atributos entre si (ainda a ser implementado). No primeiro caso, o valor constante é passado como *generic* para o módulo instanciado. Todavia, um único módulo de hardware comum aos dois tipos de uso poderia ser empregado, bastando-se fornecer os valores constantes na entidade top-level e atribuindo o valor *false* à entrada *i_changed* correspondente.

Outro assunto a ser estudado é o suporte à restrição de tamanho em dados do tipo inteiro. Embora esta alteração pareça boa uma maneira de se produzir um hardware mais otimizado, os resultados apresentados no item 4.3 - Implementação em FPGA não corroboram esta suposição, mostrando que a ferramenta de síntese foi capaz de eliminar o hardware desnecessário.

Finalmente, após sanar as limitações e os pontos para melhoria apresentados, a continuação natural do trabalho seria desenvolver novas aplicações passíveis de implementação em hardware, e estender o framework para suportar estas aplicações. Isto envolverá a tarefa de aumentar o número de componentes do PON reconhecidos pelo framework, criando novas versões especializadas dos componentes existentes (e.g., atributos e premissas para valores de ponto flutuante). À medida que o framework se consolidar, também será útil uma documentação mais formal do formato de entrada dos dados, complementando o apresentado no item 2.6 - Formato de Entrada de Dados.

6 Referências

(Ruby, 2011) Sam Ruby, Dave Thomas e David Heinemeier Hansson, **Agile Web Development with Rails, 4th Edition**. ISBN: 978-1-93435-654-8. The Pragmatic Programmers, LLC, 2011.

(McConnel, 2004) Steve McConnel, **Code Complete: A Practical Handbook of Software Construction, 2nd Edition**. ISBN: 978-0735619678. Microsoft Press, 2004.

(Simão, 2001) J. M. Simão, **Proposta de uma Arquitetura de Controle para Sistemas Flexíveis de Manufatura Baseada em Regras e Agentes**. Dissertação de Mestrado, UTFPR - CPGEI, Curitiba, 2001.

(Simão, 2005) J. M. Simão, **A Contribution to the Development of a HMS Simulation Tool and Proposition of a Meta-Model for Holonic Control**. Doctoral Thesis, UTFPR - CPGEI, Curitiba, Brazil, 2005.

(Banaszewski, 2009) R. F. Banaszewski, **Paradigma Orientado a Notificações: Avanços e Comparações**. Dissertação de Mestrado, UTFPR - CPGEI, Curitiba, 2009. Disponível em: http://arquivos.cpgei.ct.utfpr.edu.br/Ano_2009/dissertacoes/Dissertacao_500_2009.pdf

(YAML, 2009) Oren Ben-Kiki, Clark Evans, Ingy döt Net, **YAML™ Specification Index**. Disponível em: www.yaml.org/spec.

(SourceForge, 2011) Al Danial, **CLOC – Count Lines of Code**. Northrop Grumman Corporation, 2011. Disponível em: http://cloc.sourceforge.net.

Anexo I – Código VHDL Gerado para as Aplicações de Exemplo

Aplicação *wire*

```
use work.enumerated_types_pkg.all;
use work.boolean_vector_pkg.all;

entity wire_application is
  port (
    port_attribute_input_i_value: in boolean;
    port_attribute_input_i_changed: in boolean;
    port_attribute_output_o_value: out boolean;
    port_attribute_output_o_changed: out boolean;
    clock: in bit;
    reset: in bit
  );
end entity;

architecture structural of wire_application is
  -- signals for component 'attribute_input' of type attribute
  signal attribute_input_i_value: boolean;
  signal attribute_input_i_changed: boolean;
  signal attribute_input_o_value: boolean;
  signal attribute_input_o_changed: boolean;
  -- signals for component 'attribute_output' of type attribute
  signal attribute_output_i_value: boolean;
  signal attribute_output_i_changed: boolean;
  signal attribute_output_o_value: boolean;
  signal attribute_output_o_changed: boolean;
  -- signals for component 'premise_input_high' of type premise
  signal premise_input_high_i_value: boolean;
  signal premise_input_high_o_value: boolean;
  signal premise_input_high_o_changed: boolean;
  signal premise_input_high_i_changed: boolean;
  -- signals for component 'premise_input_low' of type premise
  signal premise_input_low_i_value: boolean;
  signal premise_input_low_o_value: boolean;
  signal premise_input_low_o_changed: boolean;
  signal premise_input_low_i_changed: boolean;
  -- signals for component 'condition_any_change' of type condition
  signal condition_any_change_i_values: boolean_vector(1 downto 0);
  signal condition_any_change_i_changed: boolean_vector(1 downto 0);
  signal condition_any_change_o_value: boolean;
  signal condition_any_change_o_changed: boolean;
  -- signals for component 'action_copy_input' of type action
  signal action_copy_input_i_changed: boolean;
  signal action_copy_input_o_changed: boolean;
  -- signals for component 'instigation_copy_input' of type instigation
  signal instigation_copy_input_i_value: boolean;
  signal instigation_copy_input_i_changed: boolean;
  signal instigation_copy_input_o_value: boolean;
  signal instigation_copy_input_o_changed: boolean;

begin
  -------------------------------------------------------------------------------
  -- Attributes
  -------------------------------------------------------------------------------

  -- instantiate attribute 'attribute_input'
  attribute_input: entity work.boolean_attribute(behavior)
  port map(
    clock => clock,
    reset => reset,
    i_value => attribute_input_i_value,
    i_changed => attribute_input_i_changed,
    o_value => attribute_input_o_value,
    o_changed => attribute_input_o_changed
  );

  -- instantiate attribute 'attribute_output'
  attribute_output: entity work.boolean_attribute(behavior)
  port map(
    clock => clock,
    reset => reset,
    i_value => attribute_output_i_value,
```

```
    i_changed => attribute_output_i_changed,
    o_value => attribute_output_o_value,
    o_changed => attribute_output_o_changed
  );

  -------------------------------------------------------------------------------
  -- Premises
  -------------------------------------------------------------------------------

  -- instantiate premise 'premise_input_high'
  premise_input_high: entity work.boolean_premise_compare_with_constant(behavior)
  generic map(
    COMPARISON_TYPE => EQUAL_TO,
    CONSTANT_VALUE => true
  )
  port map(
    clock => clock,
    reset => reset,
    i_value => premise_input_high_i_value,
    i_changed => premise_input_high_i_changed,
    o_value => premise_input_high_o_value,
    o_changed => premise_input_high_o_changed
  );

  -- instantiate premise 'premise_input_low'
  premise_input_low: entity work.boolean_premise_compare_with_constant(behavior)
  generic map(
    COMPARISON_TYPE => EQUAL_TO,
    CONSTANT_VALUE => false
  )
  port map(
    clock => clock,
    reset => reset,
    i_value => premise_input_low_i_value,
    i_changed => premise_input_low_i_changed,
    o_value => premise_input_low_o_value,
    o_changed => premise_input_low_o_changed
  );

  -------------------------------------------------------------------------------
  -- Conditions
  -------------------------------------------------------------------------------

  -- instantiate condition 'condition_any_change'
  condition_any_change: entity work.condition(behavior)
  generic map (
    ALWAYS_NOTIFY_ON_INPUT_CHANGED => true,
    LOGICAL_OPERATION => disjunction
  )
  port map(
    clock => clock,
    reset => reset,
    i_premises => condition_any_change_i_values,
    i_changed => condition_any_change_i_changed,
    o_result => condition_any_change_o_value,
    o_changed => condition_any_change_o_changed
  );

  -------------------------------------------------------------------------------
  -- Instigations
  -------------------------------------------------------------------------------

  -- instantiate instigation 'instigation_copy_input'
  instigation_copy_input: entity work.instigation_set_boolean_attribute_value(behavior)
  port map(
    i_value => instigation_copy_input_i_value,
    i_instigate_req => instigation_copy_input_i_changed,
    o_value => instigation_copy_input_o_value,
    o_wr_req => instigation_copy_input_o_changed
  );

  -------------------------------------------------------------------------------
  -- Connections
  -------------------------------------------------------------------------------

  -- connect premise inputs with attribute outputs
  premise_input_high_i_value <= attribute_input_o_value;
```

```
  premise_input_high_i_changed <= attribute_input_o_changed;
  premise_input_low_i_value <= attribute_input_o_value;
  premise_input_low_i_changed <= attribute_input_o_changed;

  -- connect condition inputs with premise outputs
  condition_any_change_i_values(0) <= premise_input_high_o_value;
  condition_any_change_i_changed(0) <= premise_input_high_o_changed;
  condition_any_change_i_values(1) <= premise_input_low_o_value;
  condition_any_change_i_changed(1) <= premise_input_low_o_changed;

  -- rules: simply connect a condition 'changed' output with an action 'changed' input
  action_copy_input_i_changed <= condition_any_change_o_changed;

  -- actions: simply connect 'changed' input with 'changed' output
  action_copy_input_o_changed <= action_copy_input_i_changed;

  -- connect instigation inputs with action and attribute outputs
  instigation_copy_input_i_value <= attribute_input_o_value;
  instigation_copy_input_i_changed <= action_copy_input_o_changed;

  -- connect instigation outputs with attribute inputs
  attribute_output_i_value <= instigation_copy_input_o_value;
  attribute_output_i_changed <= instigation_copy_input_o_changed;

  -- connect system ports with attribute ports
  attribute_input_i_value <= port_attribute_input_i_value;
  attribute_input_i_changed <= port_attribute_input_i_changed;
  port_attribute_output_o_value <= attribute_output_o_value;
  port_attribute_output_o_changed <= attribute_output_o_changed;
end architecture;
```

Aplicação *electronic_gate*

```
use work.enumerated_types_pkg.all;
use work.boolean_vector_pkg.all;

entity electronic_gate is
  port (
    port_attribute_button_pressed_i_value: in boolean;
    port_attribute_button_pressed_i_changed: in boolean;
    port_attribute_gate_state_o_value: out integer;
    port_attribute_gate_state_o_changed: out boolean;
    clock: in bit;
    reset: in bit
  );
end entity;

architecture structural of electronic_gate is
  -- signals for component 'attribute_button_pressed' of type attribute
  signal attribute_button_pressed_i_value: boolean;
  signal attribute_button_pressed_i_changed: boolean;
  signal attribute_button_pressed_o_value: boolean;
  signal attribute_button_pressed_o_changed: boolean;
  -- signals for component 'attribute_gate_state' of type attribute
  signal attribute_gate_state_i_value: integer;
  signal attribute_gate_state_i_changed: boolean;
  signal attribute_gate_state_o_value: integer;
  signal attribute_gate_state_o_changed: boolean;
  -- signals for component 'premise_button_pressed' of type premise
  signal premise_button_pressed_i_value: boolean;
  signal premise_button_pressed_o_value: boolean;
  signal premise_button_pressed_o_changed: boolean;
  signal premise_button_pressed_i_changed: boolean;
  -- signals for component 'premise_gate_closed' of type premise
  signal premise_gate_closed_i_value: integer;
  signal premise_gate_closed_o_value: boolean;
  signal premise_gate_closed_o_changed: boolean;
  signal premise_gate_closed_i_changed: boolean;
  -- signals for component 'premise_gate_open' of type premise
  signal premise_gate_open_i_value: integer;
  signal premise_gate_open_o_value: boolean;
  signal premise_gate_open_o_changed: boolean;
  signal premise_gate_open_i_changed: boolean;
```

```
  -- signals for component 'condition_command_open' of type condition
  signal condition_command_open_i_values: boolean_vector(1 downto 0);
  signal condition_command_open_i_changed: boolean_vector(1 downto 0);
  signal condition_command_open_o_value: boolean;
  signal condition_command_open_o_changed: boolean;
  -- signals for component 'condition_command_close' of type condition
  signal condition_command_close_i_values: boolean_vector(1 downto 0);
  signal condition_command_close_i_changed: boolean_vector(1 downto 0);
  signal condition_command_close_o_value: boolean;
  signal condition_command_close_o_changed: boolean;
  -- signals for component 'action_open_gate' of type action
  signal action_open_gate_i_changed: boolean;
  signal action_open_gate_o_changed: boolean;
  -- signals for component 'action_close_gate' of type action
  signal action_close_gate_i_changed: boolean;
  signal action_close_gate_o_changed: boolean;
  -- signals for component 'instigation_open_gate' of type instigation
  signal instigation_open_gate_i_value: integer;
  signal instigation_open_gate_i_changed: boolean;
  signal instigation_open_gate_o_value: integer;
  signal instigation_open_gate_o_changed: boolean;
  -- signals for component 'instigation_close_gate' of type instigation
  signal instigation_close_gate_i_value: integer;
  signal instigation_close_gate_i_changed: boolean;
  signal instigation_close_gate_o_value: integer;
  signal instigation_close_gate_o_changed: boolean;

begin
  -------------------------------------------------------------------------------
  -- Attributes
  -------------------------------------------------------------------------------

  -- instantiate attribute 'attribute_button_pressed'
  attribute_button_pressed: entity work.boolean_attribute(behavior)
  port map(
    clock => clock,
    reset => reset,
    i_value => attribute_button_pressed_i_value,
    i_changed => attribute_button_pressed_i_changed,
    o_value => attribute_button_pressed_o_value,
    o_changed => attribute_button_pressed_o_changed
  );

  -- instantiate attribute 'attribute_gate_state'
  attribute_gate_state: entity work.integer_attribute(behavior)
  port map(
    clock => clock,
    reset => reset,
    i_value => attribute_gate_state_i_value,
    i_changed => attribute_gate_state_i_changed,
    o_value => attribute_gate_state_o_value,
    o_changed => attribute_gate_state_o_changed
  );

  -------------------------------------------------------------------------------
  -- Premises
  -------------------------------------------------------------------------------

  -- instantiate premise 'premise_button_pressed'
  premise_button_pressed: entity work.boolean_premise_compare_with_constant(behavior)
  generic map(
    COMPARISON_TYPE => EQUAL_TO,
    CONSTANT_VALUE => true
  )
  port map(
    clock => clock,
    reset => reset,
    i_value => premise_button_pressed_i_value,
    i_changed => premise_button_pressed_i_changed,
    o_value => premise_button_pressed_o_value,
    o_changed => premise_button_pressed_o_changed
  );

  -- instantiate premise 'premise_gate_closed'
  premise_gate_closed: entity work.integer_premise_compare_with_constant(behavior)
  generic map(
    COMPARISON_TYPE => EQUAL_TO,
```

```
    CONSTANT_VALUE => 0
  )
  port map(
    clock => clock,
    reset => reset,
    i_value => premise_gate_closed_i_value,
    i_changed => premise_gate_closed_i_changed,
    o_value => premise_gate_closed_o_value,
    o_changed => premise_gate_closed_o_changed
  );

  -- instantiate premise 'premise_gate_open'
  premise_gate_open: entity work.integer_premise_compare_with_constant(behavior)
  generic map(
    COMPARISON_TYPE => EQUAL_TO,
    CONSTANT_VALUE => 2
  )
  port map(
    clock => clock,
    reset => reset,
    i_value => premise_gate_open_i_value,
    i_changed => premise_gate_open_i_changed,
    o_value => premise_gate_open_o_value,
    o_changed => premise_gate_open_o_changed
  );

  ------------------------------------------------------------------------------
  -- Conditions
  ------------------------------------------------------------------------------

  -- instantiate condition 'condition_command_open'
  condition_command_open: entity work.condition(behavior)
  generic map (
    LOGICAL_OPERATION => CONJUNCTION
  )
  port map(
    clock => clock,
    reset => reset,
    i_premises => condition_command_open_i_values,
    i_changed => condition_command_open_i_changed,
    o_result => condition_command_open_o_value,
    o_changed => condition_command_open_o_changed
  );

  -- instantiate condition 'condition_command_close'
  condition_command_close: entity work.condition(behavior)
  generic map (
    LOGICAL_OPERATION => CONJUNCTION
  )
  port map(
    clock => clock,
    reset => reset,
    i_premises => condition_command_close_i_values,
    i_changed => condition_command_close_i_changed,
    o_result => condition_command_close_o_value,
    o_changed => condition_command_close_o_changed
  );

  ------------------------------------------------------------------------------
  -- Instigations
  ------------------------------------------------------------------------------

  -- instantiate instigation 'instigation_open_gate'
  instigation_open_gate: entity work.instigation_set_integer_attribute_value(behavior)
  port map(
    i_value => instigation_open_gate_i_value,
    i_instigate_req => instigation_open_gate_i_changed,
    o_value => instigation_open_gate_o_value,
    o_wr_req => instigation_open_gate_o_changed
  );

  -- instantiate instigation 'instigation_close_gate'
  instigation_close_gate: entity work.instigation_set_integer_attribute_value(behavior)
  port map(
    i_value => instigation_close_gate_i_value,
    i_instigate_req => instigation_close_gate_i_changed,
    o_value => instigation_close_gate_o_value,
```

```
    o_wr_req => instigation_close_gate_o_changed
  );

  ------------------------------------------------------------------------------
  -- Connections
  ------------------------------------------------------------------------------

  -- connect premise inputs with attribute outputs
  premise_button_pressed_i_value <= attribute_button_pressed_o_value;
  premise_button_pressed_i_changed <= attribute_button_pressed_o_changed;
  premise_gate_closed_i_value <= attribute_gate_state_o_value;
  premise_gate_closed_i_changed <= attribute_gate_state_o_changed;
  premise_gate_open_i_value <= attribute_gate_state_o_value;
  premise_gate_open_i_changed <= attribute_gate_state_o_changed;

  -- connect condition inputs with premise outputs
  condition_command_open_i_values(0) <= premise_button_pressed_o_value;
  condition_command_open_i_changed(0) <= premise_button_pressed_o_changed;
  condition_command_open_i_values(1) <= premise_gate_closed_o_value;
  condition_command_open_i_changed(1) <= premise_gate_closed_o_changed;
  condition_command_close_i_values(0) <= premise_button_pressed_o_value;
  condition_command_close_i_changed(0) <= premise_button_pressed_o_changed;
  condition_command_close_i_values(1) <= premise_gate_open_o_value;
  condition_command_close_i_changed(1) <= premise_gate_open_o_changed;

  -- rules: simply connect a condition 'changed' output with an action 'changed' input
  action_open_gate_i_changed <= condition_command_open_o_changed;
  action_close_gate_i_changed <= condition_command_close_o_changed;

  -- actions: simply connect 'changed' input with 'changed' output
  action_open_gate_o_changed <= action_open_gate_i_changed;
  action_close_gate_o_changed <= action_close_gate_i_changed;

  -- connect instigation inputs with action and attribute outputs
  instigation_open_gate_i_value <= 2;
  instigation_open_gate_i_changed <= action_open_gate_o_changed;
  instigation_close_gate_i_value <= 0;
  instigation_close_gate_i_changed <= action_close_gate_o_changed;

  -- connect instigation outputs with attribute inputs
  attribute_gate_state_i_value <= instigation_open_gate_o_value when
    instigation_open_gate_o_changed else instigation_close_gate_o_value;
  attribute_gate_state_i_changed <= instigation_open_gate_o_changed or
    instigation_close_gate_o_changed;

  -- connect system ports with attribute ports
  attribute_button_pressed_i_value <= port_attribute_button_pressed_i_value;
  attribute_button_pressed_i_changed <= port_attribute_button_pressed_i_changed;
  port_attribute_gate_state_o_value <= attribute_gate_state_o_value;
  port_attribute_gate_state_o_changed <= attribute_gate_state_o_changed;
end architecture;
```

Anexo B – Relatório Projeto PON-HD

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA E ELETRÔNICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RELATÓRIO LÓGICA RECONFIGURÁVEL

André Augusto Kaviatkovski
Gabriel Rodrigues Garcia

CURITIBA
2017

André Augusto Kaviatkovski
Gabriel Rodrigues Garcia

RELATÓRIO LÓGICA RECONFIGURÁVEL

Relatório apresentado à Disciplina lógica reconfigurável, do Curso Superior de Engenharia de Computação dos Departamentos Acadêmicos de Informática e Eletrônica da Universidade Tecnológica Federal do Paraná.

CURITIBA
2017

Sumário

Lista de Figuras

1 Introdução

O presente relátorio tem como objetivo demonstrar as técnicas utilizada para desenvolver o projeto da disciplina lógica reconfigurável, que utiliza FPGA para a criação de projetos de circuitos eletrônicos. Foi utilizado também a linguagem PON-HD (Paradigma orientado a notificações) [3] para gerar atráves do seu compilador o VHDL do circuito principal do projeto. Os próximos capítulos irão descrever a problema tema do projeto e descrever a forma como foi utilizado o PON-HD [3] no projeto.

2 Descrição do problema

A descrição do problema a seguir foi retirado do site do professor Heitor [1].

Suponha que sejá necessário projetar um equipamento controlável para monitorar uma esteira bidirecional que transporta caixas de papelão com produtos do setor de embalagem para o setor de Carga. Há três tamanhos de caixas: Pequeno, Médio e Grande, conforme a figura 1. De um lado os produtos são embalados nas caixas e colocados na esteira e do outro lado as caixas são coletadas e colocadas em um pallet. A ordem é aleatória, podendo ser colocado qualquer tipo de caixa na esteira a qualquer momento. Porém, a capacidade de cada pallet é limitada a um arranjo de caixas (P, M, G).

A esteira é acionada por um motor de passos que gira continuamente no sentido horário, onde existem três sensores ópticos posicionados assimetricamente ao longo da mesma, com a função de detectar automaticamente o tamanho da caixa, sendo que o sensor S1 está a uma distância "A"do sensor S2 e está a uma distância "B"do sensor S3. Considere que $B > [A + \Delta]$. Do lado do setor de carga, há um painel (neste caso, representado pelos 4 displays de sete segmentos) que mostra continuamente o número de caixas de cada tipo já colocado no pallet (que inicialmente é zerado). Há um semáforo no lado do setor de embalagem que

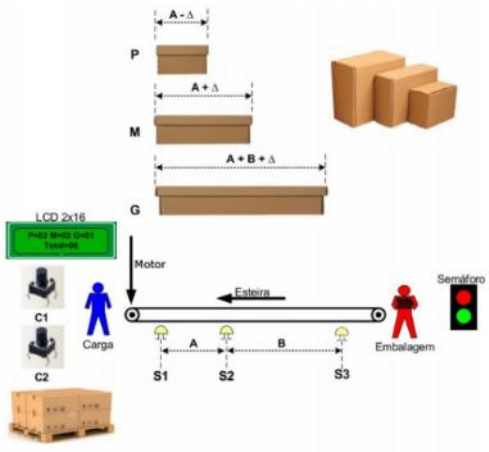

Figura 1: Demonstração do problema [1]

tem a função de sinalizar ao operador que a esteira está disponível ou não para se colocar uma nova caixa. O semáforo ficará verde se houver espaço no pallet para a caixa que acaba de passar pela esteira (na condição inicial também fica verde, pois não há nenhuma caixa na esteira ou no pallet). O semáforo ficará vermelho em duas situações:

- Se o total de caixas transportadas pela esteira atingir 9, pois o máximo permitido são 8 caixas.

- Se o operador do setor de carga perceber que não há espaço suficiente no pallet para acomodar a caixa que acaba de passar pela esteira. Ele sinaliza isto pressionando momentaneamente a chave C1. Para simplificar o sistema, suponha que acontecendo isto (semáforo vermelho) a esteira reverte (ou seja, o motor da esteira gira no sentido anti-horário) e a caixa volta para o setor de Embalagem, decrementando adequadamente os contadores (os sensores ignoram a passagem reversa da caixa). Tudo permanece parado, enquanto a carga do pallet está sendo despachada e outro pallet vazio é posicionado. Quando estiver apto a carregar um novo pallet, o operador pressiona mo-

mentaneamente a chave C2 e reinicia o ciclo. Isto faz com que o semáforo fique verde, os contadores de caixas sejam zerados (e o LCD atualizado) e a esteira ligue, permitindo trazer novas caixas.

3 Devolvimento do projeto

O kit utilizado no projto foi o DE1 (Development and Educational Board) da altera. Foi utilizado alguns dos barramentos da placa para poder representar o problema, no caso, os bostões *KEY 1* e *KEY 2* para representarem o papel das chaves C1 e C2, os leds *LEDG 7* e *LEDR 0* para fazerem o papel do semáforo. Foi utilizado também os leds *LEDR 7* e *LEDG 8* que representam quando o motor está no sentido horário e anti-horário respectivamente. Os sensores utilizados foram o modelo PHCT203 e foram ligados na portas GPIO do kit, a porta utilizada foi a port 0 e os pinos utilizdos foram $GPIO0[0]$, $GPIO0[1]$ e $GPIO0[2]$. O motor foi ligado na port 1 do GPIO, os pinos utilizados foram $GPIO1[0]$, $GPIO1[1]$, $GPIO1[2]$ e $GPIO1[3]$. Foi utilizado também o switch para definir um reset no motor, no caso o $SW8$, e também um switch que controlava e acionava todo circuito principal, no caso o $SW9$.

O projeto foi todo modelado no Quartus II em diagramas de blocos, porém para a criação do bloco principal, no caso o "Monitor", foi utilizado da linguagem PON-HD [3] para desenvolver a sua lógica, atráves do seu compilador o PON-HD [3] utiliza de arquivos com extenção ".pon"que gera um arquivo em VHDL, esse arquivo posteriormente foi usado no trabalho como bloco gerado pelo VHDL. Foi necessário também criar um debounce para controlar as entradas dos sensores umas vez que os mesmos geravam um ruído muito grande na suas saídas e isso prejudicava o funcionamento correto do projeto. O debounce também foi criado atráves da linguagem PON-HD [3], os seus blocos gerados no projeto encontra-se com o nome "AntiBounce". O motor foi controlado atráves de uma máquina de estados, esse projeto já havia sido criado anteriormente na

disciplina e o código foi reutilizado. Para controlar os displays de sete segmentos foi utilizado um código em VHDL encontrado na internet, o link com o código se encontra nas bibliográfia [2].

4 Conclusões

A linguagem PON-HD [3] se mostrou eficaz no desenvolvimento do projeto uma vez que a linguagem é simples porém não trivial, mas os resultados obtidos foram os desejados. Os pontos positivos da linguagem considerados pela equipe foram: A simplificação criada para na geração da VHDL e sua simples a utilização. Um ponto na que linguagem poderia ser melhorado, no caso, tanto na criação dos métodos quanto das regras poderiam implementar os outros operadores lógicos, a linguagem ainda só interpreta o operador "and". Em comporação com outras linguagens, principalmente o VHDL, o PON-HD [3] é mais simples de ser interpretada o que facilita a criação de projetos. O resultado do projeto como um todo também se mostrou alcançado uma vez que o sistema funcionou em todos os seus requisitos.

Referências

[1] Lopes, Silvério H. $http://bioserver.cpgei.ct.utfpr.edu.br/disciplinas/sistmicro/aulas/lab1-1-2017.pdf >$ acessado em 06 de julho as 18:30.

[2] VHDL codings tips and tricks. $< http://vhdlguru.blogspot.com.br/2010/03/vhdl-code-for-bcd-to-7-segment-display.html >$ acessado em 27 de junho as 09:45.

[3] Simão, J. $< http://www.dainf.ct.utfpr.edu.br/ jeansimao/PON/PON.htm >$ acessado em 25 de junho as 18:20.

Anexo C – Monitor.pon

```
//Programa Final Disciplina Lógica Reconfigurável - Prof. Erig
//Alunos: André Augusto Kaviatkovski
// Gabriel Garcia Rodrigues
//
// Aplicação do PONHD para o projeto de um equipamento para monitorar uma
// esteira bidirecional que transposta caixas de papelão com produtos do
// setor de Embalagem para o setor de Carga.
// A especificação do Projeto será feita em um documento a parte, que será
// anexado ao presente código para futura referência.

//Vamos Chamar nossa Entidade de Monitor
entity Monitor

//-------------------------------------------
//Criação dos Atributos
//-------------------------------------------

//Atributos que São entradas e Saídas do Sistema

attribute boolean atChave1 0 in  //C1
attribute boolean atChave2 0 in //C2
attribute boolean atSensor1 0 in //S1
attribute boolean atSensor2 0 in //S2
attribute boolean atSensor3 0 in //S3
attribute integer atCaixas_P 0 out //Quantidade de Caixas Pequenas
attribute integer atCaixas_M 0 out //Quantidade de Caixas Médias
attribute integer atCaixas_G 0 out //Quantidade de Caixas Grandes
attribute integer atCaixas_T 0 out //Quantidade de Caixas Totais (P + M + G)
attribute boolean atLED_G 1 out //Booleano para controlar o LED Verde (Green)
attribute boolean atLED_R 0 out //Booleano para controlar o LED Vermelho (Red)
attribute boolean atMotor_Hor 1 out //Booleano para Indicar Motor no Sentido Horário
attribute boolean atMotor_Anti 0 out //Booleano para Indicar Motor no Sentido Anti-Horário


//Atributos Internos ao Sistema

attribute boolean atLeitura 0 //Booleano para indicar quando uma caixa entra na posição de leitura
```

```
attribute integer atUltimo_Tamanho -1 //Vamos usar como um enum
para marcar o tamanho da última caixa que passou (0 = P; 1 = M;
2 = G; -1 = Unknown)



//-------------------------------------------
//Criação das Premissas
//-------------------------------------------

//Premissa para verificar se Estouramos o Limite de Caixas por
Pallet
premise prExcesso_Caixas atCaixas_T == 9

//Premissa para verificar se Total de caixas é maior que zero
premise prT_Caixas_M_Zero atCaixas_T > 0

//Premissa para Verificar se a Chave C1 está pressionada
premise prChave1_ON atChave1 == 1

//Premissa para Verificar se a Chave C1 está solta
premise prChave1_OFF atChave1 == 0

//Premissa para Verificar se a Chave C2 está pressionada
premise prChave2_ON atChave2 == 1

//Premissa para Verificar se a Chave C2 está solta
premise prChave2_OFF atChave2 == 0

//Premissa para Verificar se o LED Verde está ligado
premise prLED_G_ON atLED_G == 1

//Premissa para Verificar se o LED Verde está desligado
premise prLED_G_OFF atLED_G == 0

//Premissa para Verificar se o LED Vermelho está ligado
premise prLED_R_ON atLED_R == 1

//Premissa para Verificar se o LED Vermelho está ligado
premise prLED_R_OFF atLED_R == 0

//Premissa para Verificar se Sensor 1 Ativo
premise prSensor1_ON atSensor1 == 1

//Premissa para Verificar se Sensor 1 Não Ativo
premise prSensor1_OFF atSensor1 == 0

//Premissa para Verificar se Sensor 2 Ativo
premise prSensor2_ON atSensor2 == 1

//Premissa para Verificar se Sensor 2 Não Ativo
```

```
premise prSensor2_OFF atSensor2 == 0

//Premissa para Verificar se Sensor 3 Ativo
premise prSensor3_ON atSensor3 == 1

//Premissa para Verificar se Sensor 3 Não Ativo
premise prSensor3_OFF atSensor3 == 0

//Premissa para Verificar se Estamos Lendo uma caixa
premise prLeitura_ON atLeitura == 1

//Premissa para Verificar se Não Estamos Lendo uma caixa
premise prLeitura_OFF atLeitura == 0

//Premissa para Verificar se a Última caixa é Unknown
premise prUltima_U atUltimo_Tamanho == -1

//Premissa para Verificar se a Última caixa é Pequeno
premise prUltima_P atUltimo_Tamanho == 0

//Premissa para Verificar se a Última caixa é Médio
premise prUltima_M atUltimo_Tamanho == 1

//Premissa para Verificar se a Última caixa é Grande
premise prUltima_G atUltimo_Tamanho == 2



//-------------------------------------------
//Criação dos Métodos
//-------------------------------------------

//Método Para Setar o Semáforo Verde
method mtVerde_ON atLED_G = 1

//Método Para Desligar o Semáforo Verde
method mtVerde_OFF atLED_G = 0

//Método Para Setar o Semáforo Vermelho
method mtVermelho_ON atLED_R = 1

//Método Para Desligar o Semáforo Vermelho
method mtVermelho_OFF atLED_R = 0

//Método Para Incrementar a Contagem das Caixas Pequenas
method mtP_Add atCaixas_P = atCaixas_P + 1

//Método Para Decrementar a Contagem das Caixas Pequenas
method mtP_Sub atCaixas_P = atCaixas_P - 1

//Método Para Zerar a Contagem das Caixas Pequenas
```

```
method mtP_Zero atCaixas_P = 0

//Método Para Incrementar a Contagem das Caixas Médias
method mtM_Add atCaixas_M = atCaixas_M + 1

//Método Para Decrementar a Contagem das Caixas Médias
method mtM_Sub atCaixas_M = atCaixas_M - 1

//Método Para Zerar a Contagem das Caixas Médias
method mtM_Zero atCaixas_M = 0

//Método Para Incrementar a Contagem das Caixas Grandes
method mtG_Add atCaixas_G = atCaixas_G + 1

//Método Para Decrementar a Contagem das Caixas Grandes
method mtG_Sub atCaixas_G = atCaixas_G - 1

//Método Para Zerar a Contagem das Caixas Grandes
method mtG_Zero atCaixas_G = 0

//Método Para Incrementar a Contagem Total de Caixas
method mtT_Add atCaixas_T = atCaixas_T + 1

//Método Para Decrementar a Contagem Total de Caixas
method mtT_Sub atCaixas_T = atCaixas_T - 1

//Método Para Zerar a Contagem das Caixas Totais
method mtT_Zero atCaixas_T = 0

//Método Para Setar a Condição de Leitura
method mtLeitura_ON atLeitura = 1

//Método Para Desativar a Condição de Leitura
method mtLeitura_OFF atLeitura = 0

//Método Para Setar a Última Caixa como Pequena
method mtUltima_P atUltimo_Tamanho = 0

//Método Para Setar a Última Caixa como Média
method mtUltima_M atUltimo_Tamanho = 1

//Método Para Setar a Última Caixa como Grande
method mtUltima_G atUltimo_Tamanho = 2

//Método Para Setar a Última Caixa como Unknown (Estado Inicial,
sem última Caixa!)
method mtUltima_U atUltimo_Tamanho = -1

//Método Para Setar o Motor no Sentido Horário
method mtHor_ON atMotor_Hor = 1
```

```
//Método Para Desligar o Motor no Sentido Horário
method mtHor_OFF atMotor_Hor = 0

//Método Para Setar o Motor no Sentido Anti-Horário
method mtAnti_ON atMotor_Anti = 1

//Método Para Desligar o Motor no Sentido Anti-Horário
method mtAnti_OFF atMotor_Anti = 0



//-------------------------------------------
//Criação das Regras
//-------------------------------------------


//----Conjunto de Regras Relacionadas ao Excesso de Caixas:

//Liga o Led Vermelho
rule rlEC_1 mtVermelho_ON prExcesso_Caixas and prLED_G_ON and
prSensor1_OFF

//Desliga o Led Verde
rule rlEC_2 mtVerde_OFF prExcesso_Caixas and prLED_G_ON and
prSensor1_OFF

//Desliga o Motorr no Sentido Horário
rule rlEC_3 mtHor_OFF prExcesso_Caixas and prLED_G_ON and
prSensor1_OFF

//Liga o Motor no Sentido Anti-Horário
rule rlEC_4 mtAnti_ON prExcesso_Caixas and prLED_G_ON and
prSensor1_OFF

//Subtrai 1 do Total
rule rlEC_5 mtT_Sub prExcesso_Caixas and prLED_G_ON and
prSensor1_OFF

//Subtrai 1 do P, Se P foi a Última
rule rlEC_6 mtP_Sub prExcesso_Caixas and prLED_G_ON and
prSensor1_OFF and prUltima_P

//Subtrai 1 do M, Se M foi a Última
rule rlEC_7 mtM_Sub prExcesso_Caixas and prLED_G_ON and
prSensor1_OFF and prUltima_M

//Subtrai 1 do G, Se G foi a Última
rule rlEC_8 mtG_Sub prExcesso_Caixas and prLED_G_ON and
prSensor1_OFF and prUltima_G
```

```
//---Conjunto de Regras Relacionadas à leitura de uma Caixa

//Setar Condição de Leitura
rule rlLE_1 mtLeitura_ON prLeitura_OFF and prLED_G_ON and
prSensor1_ON

//Adiciona 1 Ao total de Caixas
rule rlLE_2 mtT_Add prLeitura_OFF and prLED_G_ON and
prSensor1_ON

//Detecta Se é Uma Caixa Pequena e Adiciona 1 ao P
rule rlLE_3_1 mtP_Add prLeitura_OFF and prLED_G_ON and
prSensor1_ON and prSensor2_OFF

//Complementar à Anterior... Marca a Última Caixa como Pequena
rule rlLE_3_2 mtUltima_P prLeitura_OFF and prLED_G_ON and
prSensor1_ON and prSensor2_OFF

//Detecta Se é Uma Caixa Média e Adiciona 1 ao M
rule rlLE_4_1 mtM_Add prLeitura_OFF and prLED_G_ON and
prSensor1_ON and prSensor2_ON and prSensor3_OFF

//Complementar à Anterior... Marca a Última Caixa como Média
rule rlLE_4_2 mtUltima_M prLeitura_OFF and prLED_G_ON and
prSensor1_ON and prSensor2_ON and prSensor3_OFF

//Detecta Se é Uma Caixa Grande e Adiciona 1 ao G
rule rlLE_5_1 mtG_Add prLeitura_OFF and prLED_G_ON and
prSensor1_ON and prSensor2_ON and prSensor3_ON

//Complementar à Anterior... Marca a Última Caixa como Grande
rule rlLE_5_2 mtUltima_G prLeitura_OFF and prLED_G_ON and
prSensor1_ON and prSensor2_ON and prSensor3_ON

//Se estava em Condição de Leitura, quando a caixa passa do
Sensor 1 ele considera que acabou a leitura
rule rlLE_6 mtLeitura_OFF prLeitura_ON and prLED_G_ON and
prSensor1_OFF


//---Conjunto de Regras Relacionadas a pressionar a Chave 1
(Parar Operação)

//Desliga o LED Verde
rule rlC1_1 mtVerde_OFF prChave1_ON and prLED_G_ON

//Liga o LED Vermelho
rule rlC1_2 mtVermelho_ON prChave1_ON and prLED_G_ON

//Se estiver Lendo ele Desativa a Leitura
rule rlC1_3 mtLeitura_OFF prChave1_ON and prLED_G_ON
```

```
//Subtrai 1 Caixa do Total
rule rlC1_4 mtT_Sub prChave1_ON and prLED_G_ON and
prT_Caixas_M_Zero

//Subtrai 1 Caixa de P, Se última Caixa Pequena
rule rlC1_5_1 mtP_Sub prChave1_ON and prLED_G_ON and prUltima_P

//Subtrai 1 Caixa de M, Se última Caixa Média
rule rlC1_5_2 mtM_Sub prChave1_ON and prLED_G_ON and prUltima_M

//Subtrai 1 Caixa de G, Se última Caixa Grande
rule rlC1_5_3 mtG_Sub prChave1_ON and prLED_G_ON and prUltima_G

//Desliga o Motor do Sentido Horário
rule rlC1_6 mtHor_OFF prChave1_ON and prLED_G_ON

//Liga o Motor no Sentido Anti-Horário
rule rlC1_7 mtAnti_ON prChave1_ON and prLED_G_ON


//---Conjunto de Regras Relacionadas a pressionar a Chave 2
(Retomar Operação)

//Liga o LED Verde
rule rlC2_1 mtVerde_ON prChave2_ON and prLED_R_ON

//Desliga o LED Vermelho
rule rlC2_2 mtVermelho_OFF prChave2_ON and prLED_R_ON

//Zera o Contador de Caixas Total
rule rlC2_3 mtT_Zero prChave2_ON and prLED_R_ON

//Zera o Contador de Caixas Pequenas
rule rlC2_4 mtP_Zero prChave2_ON and prLED_R_ON

//Zera o Contador de Caixas Médias
rule rlC2_5 mtM_Zero prChave2_ON and prLED_R_ON

//Zera o Contador de Caixas Grandes
rule rlC2_6 mtG_Zero prChave2_ON and prLED_R_ON

//Desliga o Motor no Sentido Anti-Horário
rule rlC2_7 mtAnti_OFF prChave2_ON and prLED_R_ON

//Liga o Motor no Sentido Horário
rule rlC2_8 mtHor_ON prChave2_ON and prLED_R_ON

//Seta a última caixa para Unknown
rule rlC2_9 mtUltima_U prChave2_ON and prLED_R_ON
```

```
//---Conjunto de Regras Inicialização (Retomar Operação)

//Regra para garantir inicio Led verde ON
rule rlINTI_1 mtVerde_ON prLED_G_OFF and prLED_R_OFF
```

Anexo D – Avaliação PON-HD

Avaliação PON-HD

1. Aprendizado e utilização

Aprender a utilizar o PON-HD foi bastante fácil por meio da utilização dos materiais anexados, em especial, os exemplos existentes ajudam a entender como definir algumas coisas, muito embora eu, no início, quando eu escrevi meus primeiros códigos eu tentei atribuir valores “true” e “false” para os atributos do tipo Booleano, ao invés de 1 e 0 que seria o correto, aqui os exemplos me mostraram que eu estava fazendo errado.

Todavia, há de se levar em conta que os alunos possuem um bom conhecimento de computação e possuem boa experiência na criação de códigos.

A dificuldade principal encontrada pelos alunos não foi relacionada diretamente com o PON-HD, mas sim com a conceituação do PON, uma vez que se tratava em pensar um programa de uma forma diferente, utilizando um paradigma próprio, que exige um planejamento muito maior antes da criação do código, pois alterações no código ou na lógica não são tão simples. Isto foi vivenciado pelos alunos que no modelo inicial acabaram por encontrar inconsistências e, ao tentar corrigir o programa, acabaram se perdendo e a única solução encontrada foi reiniciar a programação do zero, considerando os elementos que antes haviam sido relevados.

Este problema, todavia, ensinou aos alunos uma lição valiosa, com o PON também é possível você fazer uma subdivisão dos problemas e criar um bloco individual para cada parte do projeto, que podem se comunicar e trabalhar como um todo, embora você possa fazer um único bloco que faz tudo, nada impede você de elaborar blocos menores e ligar eles por meio de notificações.

Outra coisa que os alunos perceberam é que, muito embora o esforço inicial para se começar a programar seja grande, uma vez superada a etapa inicial, o aprendizado subsequente é bastante rápido e intuitivo.

2. Pontos Fortes e Fracos

São pontos fortes do PON-HD, na concepção dos alunos:

- Linguagem de alto nível;
- A linguagem é mais acessível a um programador do que VHDL puro;
- Proximidade com o Raciocínio Humano;
- Se bem projetado antes, a implementação é bastante simples e direta;
- Desempenho interessante, uma vez que explora o paralelismo permitido pela FPGA.

São pontos fracos do PON-HD, na concepção dos alunos:

- A inexistência da operação lógica “OR”, acaba por gerar uma certa redundância na escrita do código, uma vez que se você quer que duas situações executem o mesmo método, você precisa criar duas Regras distintas, que as vezes possuem um único elemento diferente nos testes, para executar a mesma resposta;
- A documentação ainda é um pouco simples, o grupo não entendeu como funcionaria a utilização de uma regra como condição para outra;
- Existem ainda alguns bugs no Sistema, em especial, os alunos detectaram que caso seja criado um método, mas nenhuma regra ative este, nenhum alerta ou mesmo erro é reportado, mas o VHDL gerado não irá compilar;
- A ausência, dentro das faculdades ligadas as áreas de programação, de um maior contato dos alunos com linguagens de programação indutivas, que poderiam facilitar o contato inicial dos programadores com o PON-HD, e que pode gerar um certo preconceito por partes destes, pelo menos em um primeiro momento.

3. Conclusão

Na concepção dos alunos o PON-HD é uma ferramenta fácil e interessante, que pode ser utilizada em sala de aula como uma alternativa ao ensino de linguagens específicas como VHDL, além disso proporcionam aos alunos uma forma diferente de pensar programação (pensamento Indutivo).

Os problemas e fraquezas encontrados são muito mais ligados aos estágios iniciais do presente projeto, bem como a ausência de uma base anterior dos alunos e não são tão ligados à ferramenta, pois estas exigem conhecimentos diferentes dos clássicos aprendidos dentro do curso.

De toda forma, a oportunidade de utilização do PON-HD, concedida aos alunos, propiciou a estes um desenvolvimento acadêmico valoroso, uma vez que não se tratou apenas do aprendizado de uma ferramenta específica, mas também de todo um arcabouço único, que, de outra forma, não faria parte da formação destes.