UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

ALESANDRO ALESSI
ANDRÉ LUIZ CIRINO DOS SANTOS

SISTEMA EMBARCADO EM FPGA PARA RECONHECIMENTO DE OBJETOS VERMELHOS EM VÍDEO ATRAVÉS DO PROCESSAMENTO DE IMAGENS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2015

ALESSANDRO ALESSI
ANDRÉ LUIZ CIRINO DOS SANTOS

SISTEMA EMBARCADO EM FPGA PARA RECONHECIMENTO DE OBJETOS VERMELHOS EM VÍDEO ATRAVÉS DO PROCESSAMENTO DE IMAGENS

Relatório da disciplina de Trabalho de Conclusão de Curso de Engenharia de Computação apresentado ao Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Engenheiro em Computação".

Orientador: Prof. Dr. Carlos Raimundo Erig Lima

CURITIBA

2015

RESUMO

Este trabalho apresenta a implementação de um sistema embarcado dedicado ao processamento digital de imagens através da aplicação da técnica de segmentação por limiarização para reconhecimento de objetos vermelhos, utilizando a plataforma de desenvolvimento e educação Altera DE2 em conjunto com o módulo de câmera TRDB D5M e um monitor VGA. O sistema é desenvolvido através da metodologia modular e o processamento de imagens é implementado inteiramente em *hardware*, utilizando as linguagens de descrição de *hardware* VHDL e Verilog. Para criação do projeto, foi utilizada a ferramenta Qsys da Altera. Nela, foram utilizados componentes que a Altera disponibiliza juntamente com um componente personalizado criado para o processamento de imagem. Este sistema utiliza o barramento Avalon para comunicação entre os componentes e tem um processador Nios II implementado. Por fim, este trabalho serve de base para o desenvolvimento de sistemas mais robustos.

Palavras-chave: Sistemas Embarcados, FPGA, Processamento Digital de Imagens, Reconhecimento de Objetos, Vídeo, VHDL, Verilog.

ABSTRACT

This paper presents the implementation of an embedded system dedicated to digital image processing by applying the threshold segmentation technique for recognition of red objects, using the Altera DE2 development and education platform in combination with the TRDB D5M camera module. The system is developed by modular methodology and the image processing is implemented entirely in hardware, using VHDL and Verilog hardware description languages. To create the design, it was used the Qsys tool from Altera. In it, components provided by Altera were used along with a custom component created for image processing. This system uses the Avalon bus for communication between components and has a Nios II processor implemented in it. Finally, this study provides the basis for the development of more robust systems.

Keywords: Embedded Systems, FPGA, Digital Image Processing, Object Recognition, Video, VHDL, Verilog.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

FPGA	Field Programmable Gate Array
VHSIC	Very High Speed Integrated Circuit
VGA	Video Graphics Array
DVI	Digital Visual Interface
CPU	Central Processing Unit
RISC	Reduced Instruction Set Computer
IRQ	Interrupt ReQuest
RGB	Red-Green-Blue
CFA	Color Filter Array
HAL	Hardware Abstraction Layer
IDE	Integrated Development Environment
BSP	Board Support Package
API	Application Program Interface
ANSI	American National Standards Institute
CPLD	Complex Programmable Logic Device
SoC	System on a Chip
NTSC	National Television System Committee
PAL	Phase Alternating Line
SDRAM	Synchronous Dynamic Random-Access Memory
SRAM	Static Random-Access Memory
RAM	Random-Access Memory
PLL	Phase-Locked Loop
DAC	Digital-to-Analog Converter
LED	Light-Emitting Diode
LCD	Liquid-Crystal Display
VHDL	VHSIC Hardware Description Language
ASIC	Application Specific Integrated Circuits
IEEE	Institute of Electrical and Electronics Engineers
JTAG	Joint Test Action Group
UART	Universal Asynchronous Receiver/Transmitter
FIFO	First-in First-out
FPS	Frames Per Second

SUMÁRIO

1 INTRODUÇÃO

Nas últimas décadas, graças aos grandes avanços no campo da tecnologia, muitos sistemas embarcados começaram a fazer parte do cotidiano da sociedade moderna. Atualmente, estamos cada vez mais dependentes destes aparelhos seja para pesquisa, trabalho, necessidade ou entretenimento. Dispositivos multimídia, como celulares por exemplo, têm contribuído imensamente para a comunicação entre as pessoas.

Grande parte do processamento multimídia é muito complexo e exije bastante computação, como por exemplo o processamento de imagens. Para que haja melhor *performance*, é necessário um projeto inteligente de *hardware* e *software*.

Uma plataforma que está ganhando espaço na área de pesquisa e desenvolvimento de sistemas embarcados por ser extremamente flexível é a FPGA ou *Field-Programmable Gate Array*. Trata-se de um dispositivo lógico que contém uma matriz bi-dimensional de células lógicas e interconexões programáveis (CHU, 2011). Estas células podem ser configuradas para realizar funções customizadas através de uma linguagem de descrição de *hardware*, como o Verilog ou o VHDL (VHSIC *Hardware Description Language*). Além disso, também é possível criar processadores de *software* em conjunto com o projeto de *hardware*, o que a torna uma excelente ferramenta para o desenvolvimento de um projeto dedicado ao processamento de imagens.

1.1 OBJETIVO

O objetivo deste trabalho é desenvolver um sistema embarcado em FPGA dedicado ao reconhecimento de objetos vermelhos através do processamento de imagens captadas por uma câmera e posteriormente exibidas em um monitor VGA. O processamento será realizado em *hardware* e o resultado estará disponível na saída VGA do dispositivo.

Para que isso seja possível, será necessário exatidão na temporização dos componentes e controladores utilizados, de forma a garantir a perfeita comunicação entre os

diversos componentes e periféricos do sistema. Dentre eles, podem ser destacados: o decodificador D5M, as memórias de programa e de *buffer* de vídeo, o processador Nios II e o controlador de vídeo VGA.

Será utilizado um processador para coordenar alguns dos componentes como, por exemplo, o componente responsável pela configuração da câmera utilizada no projeto. Para o processamento das imagens será utilizado um algoritmo de segmentação, mais especificamente, a limiarização, explicado em mais detalhes na seção 3.3.4.

Neste trabalho serão utilizados a plataforma de desenvolvimento DE2 da Altera, um módulo de câmera TDRB-D5M da empresa Terasic, e um monitor VGA que serão descritos no capítulo 5.

1.2 REQUISITOS DO SISTEMA

Através da análise do projeto, os seguintes requisitos funcionais e não funcionais foram estabelecidos para o sistema.

1.2.1 REQUISITOS FUNCIONAIS

RF1: O sistema deve obter a imagem proveniente de um módulo de câmera digital.

RF2: O sistema deve ser capaz de alterar a exposição à luz da câmera.

RF3: O sistema deve processar a imagem inteiramente por *hardware*.

RF4: O sistema deve ser capaz de reconhecer objetos de cor vermelha na imagem.

RF5: O sistema deve gravar a imagem atual em um dispositivo de memória.

RF6: O sistema deve mostrar a imagem processada em um monitor VGA.

1.2.2 REQUISITOS NÃO FUNCIONAIS

RNF1: O sistema deverá ter uma taxa de quadros superior a 10 FPS.

RNF2: O sistema deverá utilizar a segmentação por limiarização para processamento da imagem.

RNF3: A imagem processada deverá estar limiarizada em branco e preto.

RNF4: O processamento de imagem deve ser implementado em VHDL e Verilog, utilizando uma FPGA.

RNF5: O ajuste da exposição à luz deverá ser realizada pelos botões contidos na FPGA.

1.3 TRABALHOS CORRELATOS

Alguns trabalhos semelhantes utilizam a FPGA para a detecção de imagem ou padrões. Em (HEMATIAN et al., 2011) é proposto um sistema baseado em FPGA para a detecção de padrões, neste caso o olho humano, mais especificamente na íris. O projeto é composto apenas pela FPGA, por uma câmera e um monitor VGA. Neste artigo ainda são testados vários algoritmos para a detecção de padrões e o resultado de seus desempenhos.

A capacidade da FPGA em realizar operações com imagens também pode ser verificada em (CHO et al., 2007), onde foi desenvolvido um sistema de captura de cores através do histograma da imagem capturada. Nele, o sistema recebe a imagem de uma câmera, processa e exibe o resultado através da interface DVI, marcando as cores propostas.

1.4 MOTIVAÇÃO

A maior motivação para a realização deste trabalho é o interesse, por parte dos autores, na área de sistemas embarcados. Este trabalho tornará possível a consolidação e entendimento do que foi estudado durante o tempo de graduação. Além disso, como discutido anteriormente, a área de sistemas embarcados está evoluindo cada vez mais, assim como a de processamento digital de imagens.

1.5 ESTRUTURA DO TRABALHO

A estrutura deste documento é baseada em capítulos. O presente capítulo apresenta o trabalho, explicitando os objetivos e a motivação acerca do tema abordado. Os capítulos 2 ao 7 se referem à teoria utilizada para a realização deste trabalho. O capítulo 8 trata da implementação propriamente dita. Já o capítulo 9 descreve os testes e resultados alcançados. No capítulo 10 são discutidas conclusões, juntamente de alguns possíveis trabalhos futuros. Por fim, são listadas as referências utilizadas no desenvolvimento do projeto.

2 SISTEMAS EMBARCADOS

Um sistema embarcado é um sistema computadorizado especializado que é parte de um ambiente mais complexo. Estes sistemas geralmente possuem um processador embarcado e são construídos para realizar tarefas de uma maneira muito mais efetiva que os computadores de mesa tanto em relação a sua *performance* e consumo de energia, como em tamanho. Muitas aplicações que fazem uso de uma interface digital utilizam um sistema embarcado. Alguns sistemas possuem sistema operacionais já outros, mais específicos, têm sua lógica inteiramente implementada em um programa (OSHANA, 2006).

Sistemas embarcados usualmente são compostos de uma combinação de *hardware* (memórias, dispositivos de entrada/saída, etc.) e *software* (programa com as tarefas para o processador). Alguns sistemas podem ter mais ou menos destas características, dependendo da sua função, dando flexibilidade e uma gama de possibilidades para otimização e customização. Na figura 1 é ilustrado um exemplo de sistema embarcado típico.

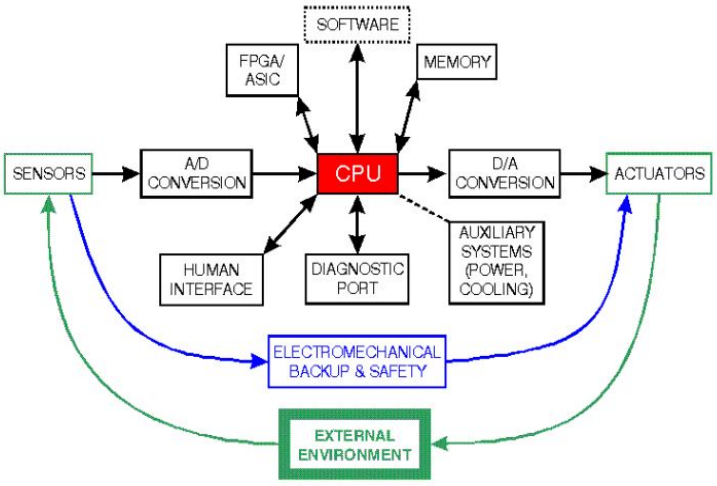

Figura 1: Sistema Embarcado Típico. (OSHANA, 2006)

De acordo com (OSHANA, 2006), um sistema embarcado pode realizar as seguintes funções:

- Monitorar o ambiente: o sistema lê as informações dos sensores de entrada, processa essa informação e mostra ao usuário.
- Controlar o ambiente: o sistemas gera e transmite aos atuadores comandos de controle.
- Transformar a informação: o sistema transforma os dados coletados de acordo com a necessidade.

2.1 PROCESSADORES EMBARCADOS

Um processador embarcado é um microprocessador utilizado em um sistema embarcado. Como dito anteriormente, estes processadores são menores e consomem menos energia. De acordo com (ELETRONICS, 2015), eles podem ser divididos em duas categorias:

- Microprocessadores: *chip* de CPU.
- Microcontroladores: além do *chip* de CPU, há vários outros periféricos interligados.

Em uma FPGA os processadores podem ser *hard* ou *soft*. Processadores *hard* são implementados em uma pastilha de silício e, assim, não podem ser mais alterados. Eles oferecem uma *performance* maior que os processadores *soft*, dependendo de fatores como sua arquitetura, taxa de *clock* e tecnologia aplicada no processo de fabricação (ALTERA, 2015e).

Em contrapartida, o processador *soft* é descrito inteiramente em lógica programável através de linguagens de descrição de *hardware* (explicadas no capítulo 6). Assim, ele tem um alto grau de customização e pode facilmente ser alterado para se ajustar à necessidade de projeto. Sua *performance* e custo dependem principalmente da FPGA em que está implementado, mas geralmente são menores que à dos processadores *hard* (ALTERA, 2015e).

2.2 NIOS II

O processador Nios II é um processador *soft*, definido em uma liguagem de descrição de *hardware*, que pode ser implementado nas FPGAs da Altera através do programa Quartus II, descrito na seção 4.1. Ele pode ser utilizado em conjunto com uma variedade

de componentes para assim formar um sistema completo, como pode ser visto na figura 2.

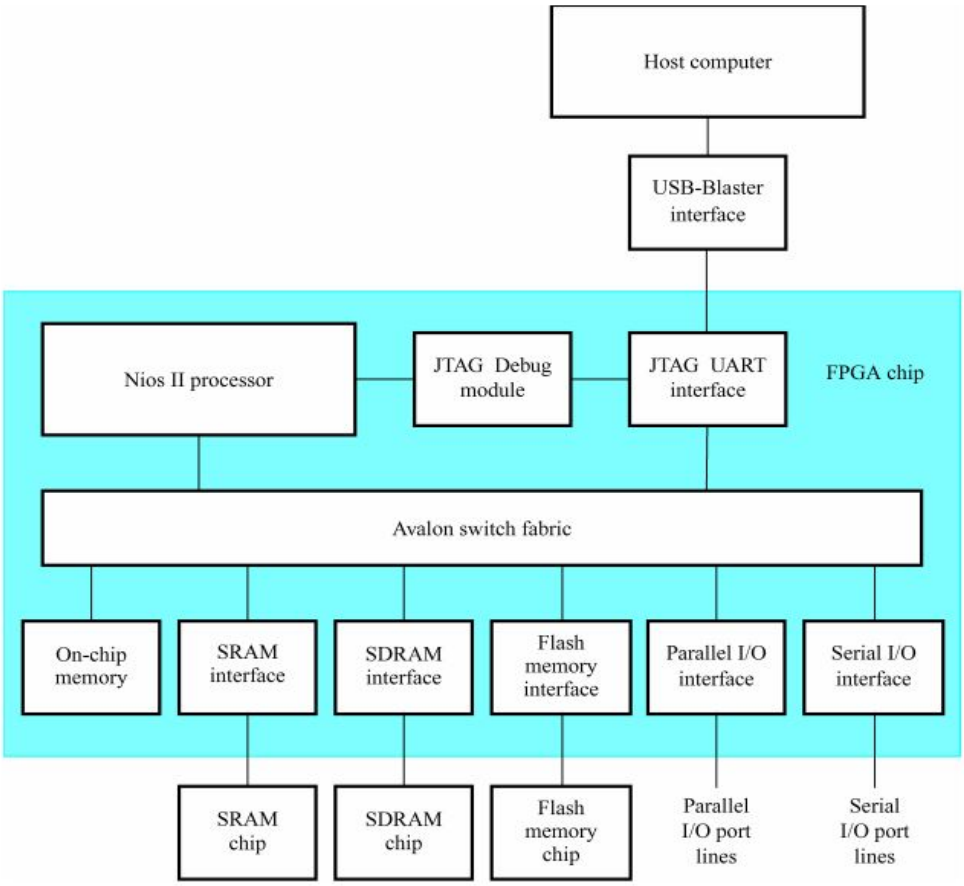

Figura 2: Exemplo de um sistema Nios II em uma plataforma Altera DE2. (ALTERA, 2013a)

O processador e as interfaces necessárias para conexão entre outros componentes do sistema são implementados no *chip* da FPGA. Como visto na figura 2, esses componentes comunicam-se entre si através da interconexão *Avalon Switch Fabric*, explicada em mais detalhes na seção 7.2.

Sua arquitetura é RISC e suas operações lógicas e aritméticas são realizadas em operandos dentro de registradores de propósito geral. Todos os seus registradores são de 32 *bits*. Além disso, ele utiliza barramentos de instrução e dados separadamente, o que é conhecido como arquitetura Harvard(ALTERA, 2013a). O núcleo do processador é mostrado na figura 3.

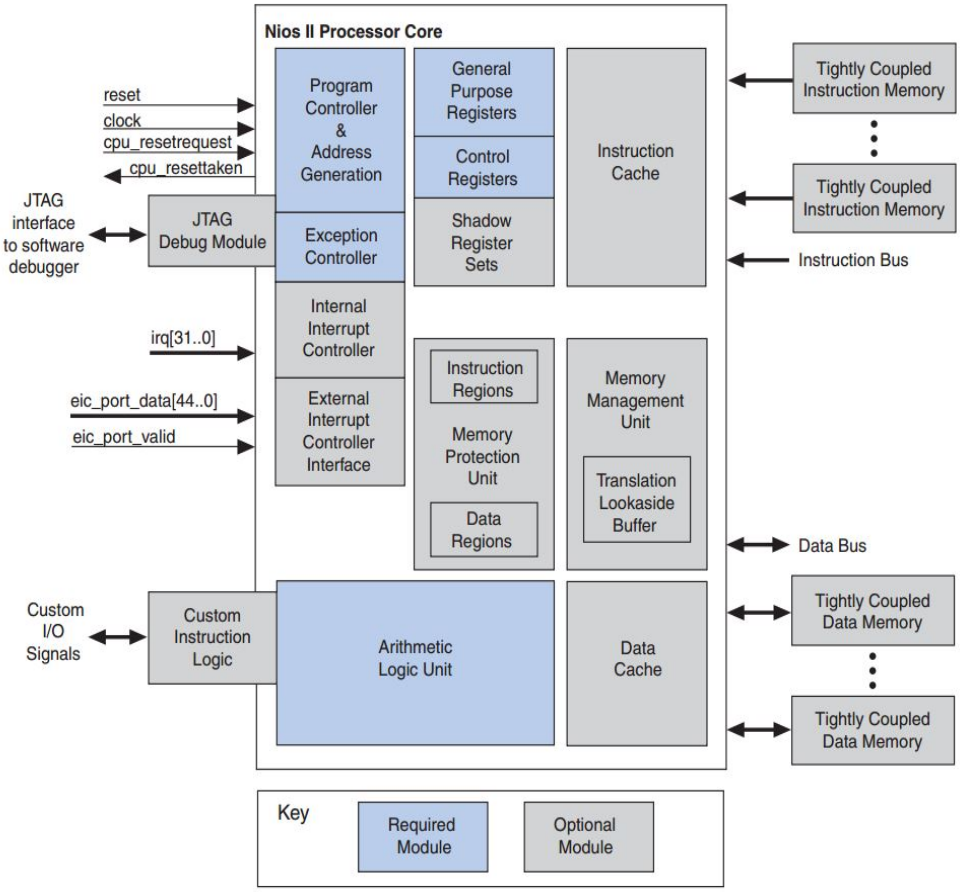

Figura 3: Diagrama de blocos do núcleo do processador Nios II. (ALTERA, 2014b)

O processador Nios II pode ser implementado em três diferentes configurações:

- Nios II/f: versão rápida projetada para uma maior *performance*.
- Nios II/s: versão padrão que requer menos recursos ao custo de uma menor *performance*.
- Nios II/e: versão econômica com um conjunto limitado de recursos.

Tabela 1: Comparação das versões do Nios II. (CHU, 2011)

	Nios II/e	**Nios II/s**	**Nios II/f**
Processor Pipeline	1 stage	5 stages	6 stages
Branch Prediction	-	static	dynamic
Multiplication	software	3-cycle multiplier	1-cycle multiplier
Shift n	software	3-cycle barrel shifter	1-cycle barrel shifter
Instruction Cache	-	0.5 KB to 64 KB	0.5 KB to 64 KB
Data Cache	-	-	0.5 KB to 64 KB
MMU/MPU	-	-	optional
Circuit Size	540 LEs	1030 LEs	1600 LEs
Max Clock Rate	195 MHz	110 MHz	140 MHz
Performance	18 MIPS	55 MIPS	145 MIPS

2.2.1 INTERRUPÇÃO NO NIOS II

A arquitetura do processador Nios II suporta até 32 interrupções internas de *hardware*. Seu núcleo dispõe de 32 IRQs sensíveis à nível e a prioridade de cada uma é determinada através de *software*.

Através do registrador de controle *ienable*, que contém um *bit* de habilitação para cada IRQ, pode-se habilitar ou desabilitar uma interrupção. Além disso, as interrupções são habilitadas ou não globalmente, utilizando o *bit* PIE do registrador de controle *status* (ALTERA, 2014b).

Em resumo, para que uma interrupção por *hardware* seja gerada, as condições abaixo devem ser verdadeiras:

- O *bit* PIE do registrador *status* deve ser 1.
- Uma IRQ é atribuída.
- O correspondente *bit* do registrador *ienable* é 1.

3 VÍDEO E IMAGENS

Vídeo nada mais é que a composição de vários quadros (ou imagens) exibidas numa rápida sucessão. Essas imagens são mostradas numa frequência mínima para que sejam percebidas pelo olho humano com fluidez. Em um vídeo típico, por exemplo, os quadros são mostrados de 30 a 120 vezes por segundo.

Para exibir um vídeo através de um monitor VGA é necessário transmitir os *pixels* no formato RGB, seguindo as suas especificações de temporização. Para tal, dois sinais são necessários: o *hsync* e o *vsync*, responsáveis pelas sincronizações horizontal e vertical do quadro, respectivamente.

Já no módulo de câmera TRDB-D5M, cada imagem é rodeada por regiões denominadas *vertical blanking* e *horizontal blanking* (ver figura 4). Esses espaços são utilizados na geração dos sinais FVAL e LVAL, cuja função é indicar quando os dados de *pixel* estão disponíveis nos pinos da câmera (TERASIC, 2009).

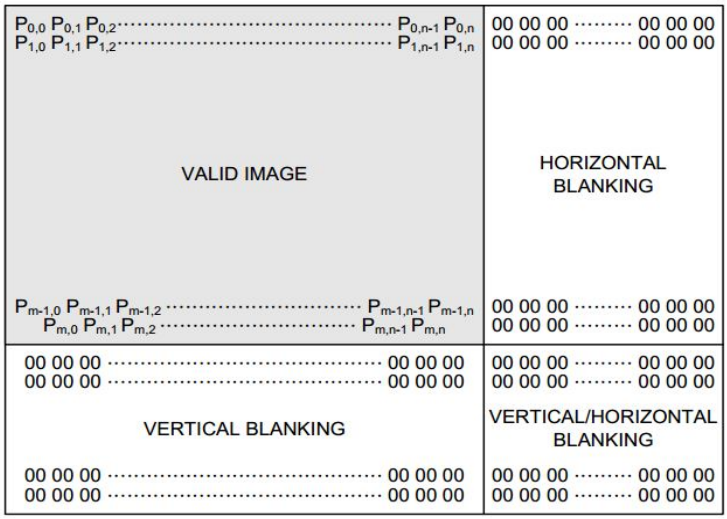

Figura 4: Ilustração espacial da imagem na câmera TRDB-D5M. (TERASIC, 2009)

3.1 RESOLUÇÃO E TAXA DE ATUALIZAÇÃO DE QUADROS

Um quadro é uma matriz bi-dimensional de *pixels* e sua resolução é definida como o número de *pixels* nos eixos x e y, como visto na figura 5. Neste projeto, a resolução adotada será o padrão VGA de 640x480 e a taxa de atualização de quadros utilizada na saída VGA será de 60 quadros por segundo.

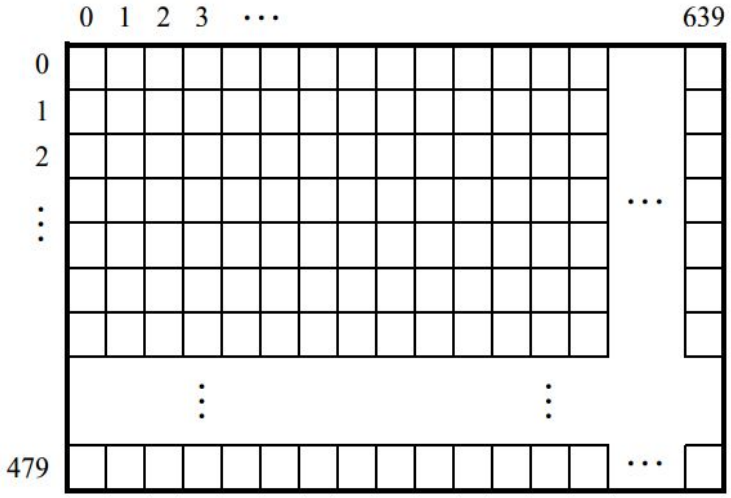

Figura 5: Disposição dos *pixels* no quadro com resolução de 640x480. (ALTERA, 2013b)

3.2 BUFFER DE QUADROS

Um *buffer* de quadros é uma memória que armazena toda a informação contida em um imagem. A imagem contida nele está sempre disponível para leitura, facilitando com isso o processamento da imagem. Como neste trabalho será utilizado uma resolução de 640x480, a quantidade total de *pixels* a ser armazenada no *buffer* é de 307.200 *pixels* de 24 *bits* cada. Isto equivale, aproximadamente, a 1MB de dados, o que demanda uma memória relativamente grande. Por essa razão, a memória que exercerá o papel de *buffer* será a SDRAM (detalhada na subseção 5.1.5).

3.3 PROCESSAMENTO DIGITAL DE IMAGENS

Neste trabalho, o tratamento da imagem é composto de duas etapas: a transformação da imagem do padrão *Bayer* para o formato RGB e depois a aplicação do algoritmo de limiarização na imagem, detalhadas a seguir. Isto se deve ao fato de que a câmera TRDB-D5M, como muitas outras, fornece a imagem no padrão *Bayer* (TERASIC, 2009).

3.3.1 PADRÃO BAYER

Devido à necessidade de redução de custo na produção, a maioria das câmeras digitais faz uso do CFA, ou Color Filter Array, para garantir a aquisição das cores. Para isso, elas utilizam três sensores para adquirir cada uma das três cores primárias (vermelho, verde e azul). Esse sistema funciona com o sensor filtrando cada cor de uma vez. Como consequência, a luz é fragmentada nas três componentes monocromáticas e cada pixel passa a representar uma cor, como visto na figura 6.

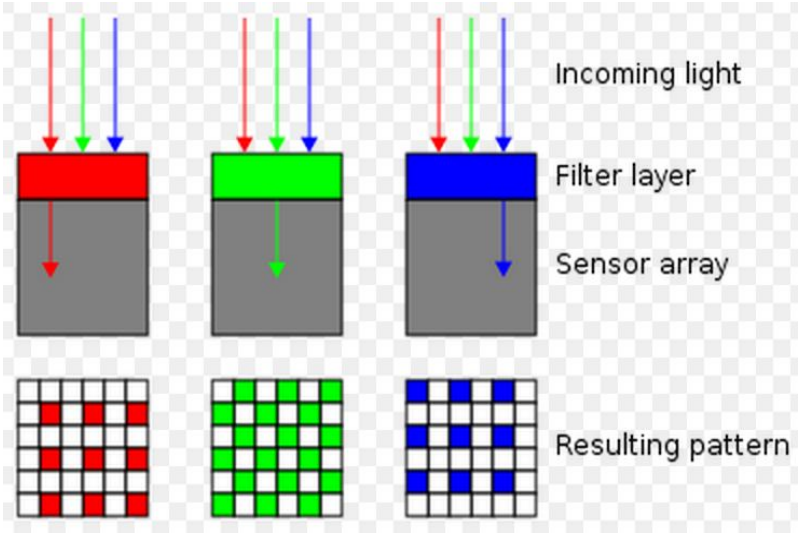

Figura 6: Filtro de cores e o padrão gerado. (WIKIPEDIA, 2015)

Embora a imagem obtida não esteja completa, a informação restante pode ser obtida com grande grau de exatidão através da utilização de um processo conhecido como interpolação ou demosaico.

No módulo TRDB-D5M, os dados de *pixel* são transmitidos no padrão *Bayer* composto de quatro cores representando as cores dos três filtros (G1, G2 , R e B). Este padrão pode ser visto na figura 7.

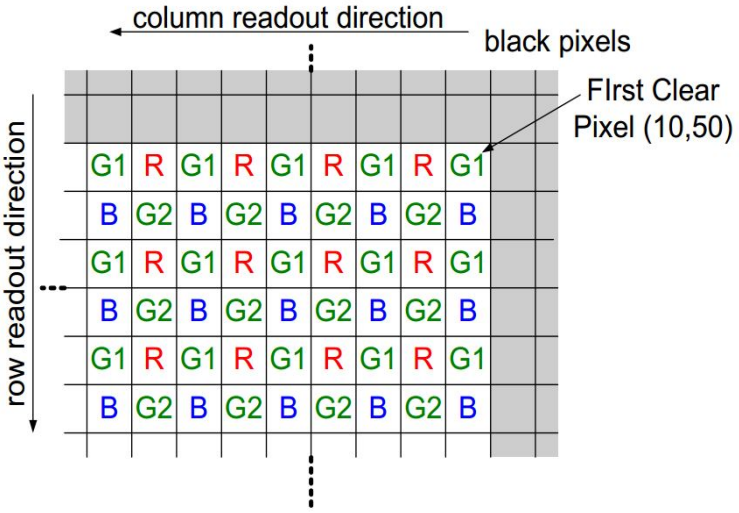

Figura 7: Detalhamento do padrão *Bayer* no módulo TRDB-D5M. (TERASIC, 2009)

3.3.2 PADRÃO RGB

O formato RGB é composto pela intensidade das três cores primárias: vermelho, verde e azul. A quantidade de valores de intensidade está associada ao número de *bits* para cada canal. No caso deste projeto, cada canal tem 8 *bits* de resolução que, quando transmitidos para VGA, passarão a ter os 10 *bits* necessários para o funcionamento correto do DAC, detalhado na subseção 5.1.2.

3.3.3 ALGORITMOS DE INTERPOLAÇÃO E DEMOSAICO

Como dito anteriormente, a imagem provida pela câmera TRDB-D5M precisa passar pelo processo de interpolação. A qualidade da imagem depende muito da técnica utilizada e há vários algoritmos de interpolação que podem ser utilizados para tal.

O algoritmo mais simples, do ponto de vista computacional, é o algoritmo do vizinho mais próximo, onde para cada *pixel* interpolado é atribuído o valor do *pixel* de mesma cor mais próximo (MIKLOS, 2004). O núcleo de interpolação deste algoritmo é definido como

$$h(x) = \begin{cases} 1 & \text{se} \quad 0 \leqslant |x| < 0,5 \\ 0 & \text{se} \quad 0,5 \leqslant |x| \end{cases}$$

A câmera TRDB-D5M suporta uma função chamada *Binning* que reduz a resolução da imagem através da combinação de *pixels* adjacentes e de mesma cor em um só

pixel, como visto na figura 8 (TERASIC, 2009).

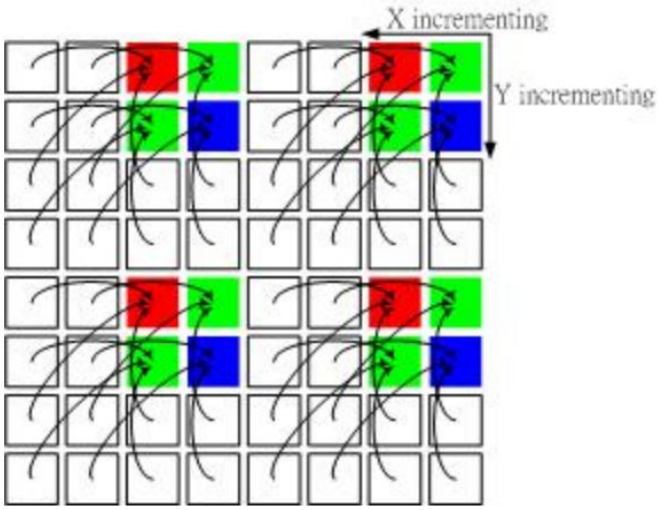

Figura 8: Modo de leitura da câmera TRDB-D5M utilizando a função *Binning*. (TERASIC, 2009)

Isto faz com que o algoritmo do vizinho mais próximo seja tão eficaz quanto outros algoritmos mais complexos. Como consequência, o algoritmo mais simples será utilizado no presente trabalho e é melhor visto na figura 9.

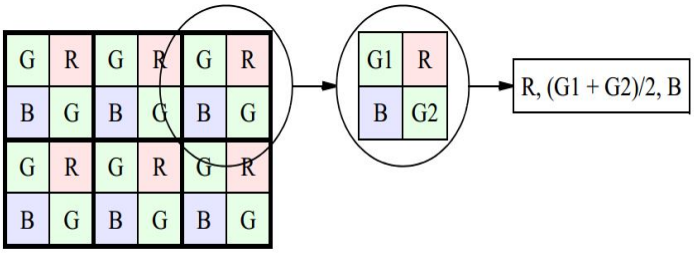

Figura 9: Interpolação através dos *pixels* vizinhos para determinação do *pixel* RGB. (ALTERA, 2013b)

3.3.4 ALGORITMO DE SEGMENTAÇÃO POR LIMIARIZAÇÃO

A cor de um *pixel* contém uma ótima informação para classificá-lo em muitas aplicações. Por exemplo, a cor da pele humana pode ser utilizada para encontrar o rosto de uma pessoa numa imagem. É claro que muita confusão pode ser feita se for a única informação disponível. Entretanto, pode ser de grande ajuda a outros algoritmos mais complexos na obtenção de padrões em imagens.

No presente projeto será utilizado um algoritmo para classificação de cada *pixel* na imagem como ”vermelho”ou ”não vermelho”. Isto é conhecido como segmentação. Existem

muitos algoritmos com esta função, porém o utilizado será o de limiarização (G.SHAPIRO; STOCKMAN, 2001).

O algoritmo de limiriazação é definido como

$$g(x,y) = \begin{cases} 1 & \text{se} \quad f(x,y) \geqslant T \\ 0 & \text{se} \quad T > f(x,y) \end{cases}$$

Com isso, *pixels* cujos valores estejam acima de um limiar T serão avaliados como vermelho e transformados em branco, enquanto os outros serão transformados em preto. Isso resultará numa imagem que mostra apenas *pixels* brancos e pretos.

4 FERRAMENTAS E SOFTWARE

4.1 QUARTUS II

Quartus II é um *software* que oferece um completo ambiente de projeto multi-plataforma para FPGAs (ALTERA, 2011). Todas as fases de projeto são contempladas, desde sua criação até a simulação e programação da FPGA. O *hardware* a ser projetado pode ser descrito tanto em Verilog, como em VHDL, ou até mesmo adicionando e conectando diferentes blocos funcionais através de sua interface gráfica, tornando a interconexão entre os módulos fácil e, também, permitindo uma alta produtividade e *performance*.

4.1.1 MEGAFUNCTIONS

Megafunctions são blocos pré-testados de IP que são otimizados para fazer uso eficiente da arquitetura da FPGA (ALTERA, 2015b). Com isso, utilizando essas funções para implementação de um sistema complexo, o projetista não perde tempo refazendo blocos funcionais comuns já existentes. Assim que instanciados no *software* Quartus II, os blocos IP podem ser configurados de acordo com a necessidade do projeto.

4.1.2 QSYS

Qsys é uma ferramenta de integração de sistema disponível dentro do Quartus II que gera automaticamente lógica para interconectar funções de propriedade intelectual IP e componentes (ALTERA, 2014a). Com isso, o processo de desenvolvimento do sistema torna-se mais rápido e eficiente, pois elimina as tarefas que consomem muito tempo, além de tornar possível a reutilização de componentes em outros projetos. Esta ferramenta também possibilita a construção de um sistema com microprocessador embarcado que inclui memórias e periféricos. É nele que o processador *soft* Nios II e os outros componentes do presente trabalho são implementados.

4.2 NIOS II EMBEDDED DESIGN SUITE

É um pacote completo de desenvolvimento para projeto de *software* no processador Nios II. Além de ferramentas de desenvolvimento, ele contém vários *plug-ins*, *drivers* de dispositivos e uma biblioteca HAL (*Hardware Abstraction Layer*). Incluí também várias ferramentas proprietárias e *open-sources* (como o GNU C/C++) para criação de programas.

Uma das ferramentas do Nios II EDS é o Nios II *Software Build Tools*. Ele é um ambiente integrado de desenvolvimento projetado como um conjunto de *plug-ins* para o *software* Eclipse IDE. Ele provê uma plataforma de desenvolvimento que é compatível com todos os sistemas de processadores embarcados Nios II (ALTERA, 2015d).

Um projeto criado no Nios II EDS consiste em (ver figura 10):

- Um projeto de aplicação que é uma coleção de arquivos fontes.
- Projetos de bibliotecas personalizadas.
- Um projeto BSP (*Board Support Package*).

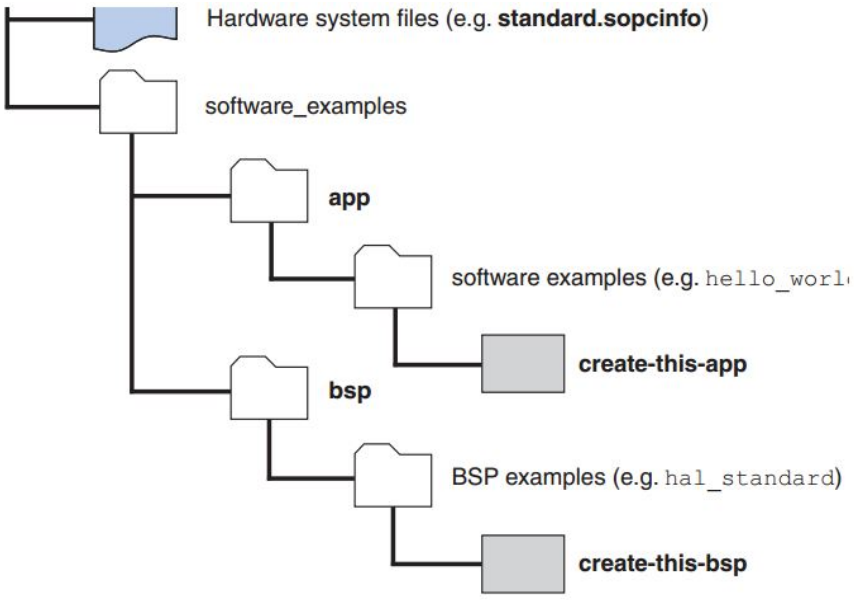

Figura 10: Estrutura típica de um projeto Nios II EDS. (ALTERA, 2014c)

O Nios II EDS gera automaticamente um completo BSP , incluindo os *drivers* para os periféricos do sistema. O projeto BSP é uma biblioteca especializada contendo código de suporte específico à configuração de *hardware* do projeto. Nele está contida a biblioteca HAL, que é um ambiente embarcado com a função de disponibilizar uma

interface de *driver* de dispositivo simples para comunicação entre as camadas de aplicação e de *hardware*, como visto na figura 11. A API (*Application Program Interface*) HAL é integrada com a biblioteca padrão ANSI C, permitindo a utilização de bibliotecas conhecidas.

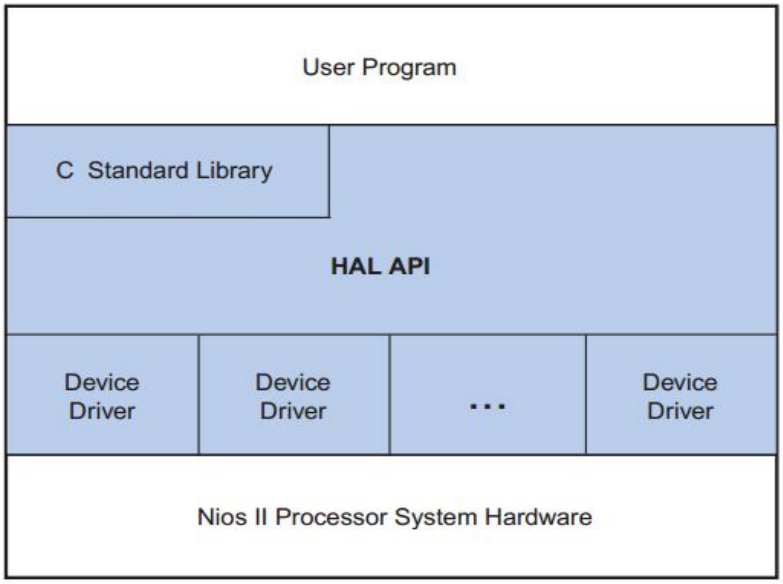

Figura 11: Camadas de um projeto criado com a biblioteca HAL. (ALTERA, 2014c)

5 PLATAFORMA DE DESENVOLVIMENTO E HARDWARE

Muitas empresas da área de sistemas embarcados disponibilizam plataformas de desenvolvimento e dispositivos de lógica programável como microcontroladores, CPLDs, FPGAs e SoCs, mas duas empresas ocupam um lugar de destaque no mercado de FPGAs: Altera e Xilinx (como pode ser visto na figura 12). Ambas provêm uma gama de *kits* de desenvolvimento que diferenciam em preço, complexidade e área de aplicação (ALTERA, 2015a) (XILINX, 2015). Além delas, muitas empresas terceirizadas criam *kits* como a plataforma de desenvolvimento Altera DE2 e o módulo de câmera TRDB-D5M, que são fabricadas pela empresa Terasic (TERASIC, 2015).

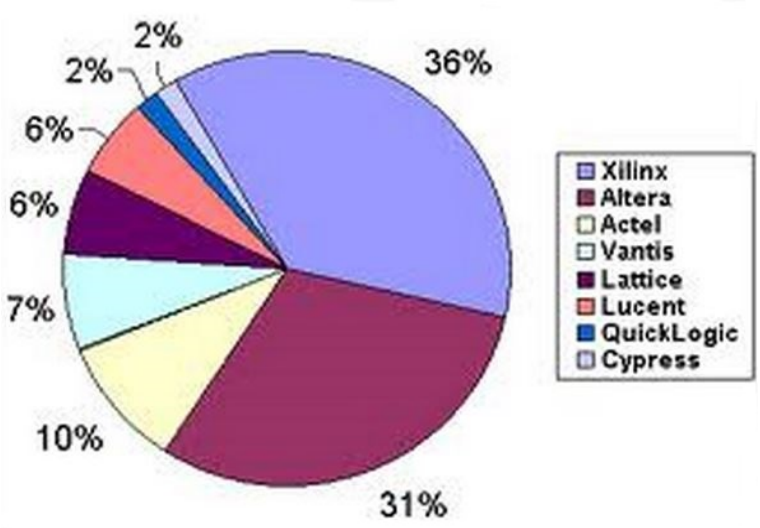

Figura 12: Fatia de mercado de FPGAs. (SOURCETECH411, 2013)

5.1 DE2 E COMPONENTES

A plataforma de desenvolvimento e educação Altera DE2 é considerada ideal para o ensino de lógica digital, arquitetura de computadores e FPGAs. Foi desenvolvida com o intuito de contribuir no ensino provido por universidades e seus laboratórios. O que torna esta placa versátil é a variadade de recursos presentes nela, conforme visto na figura 13. Alguns deles são:

- FPGA Cyclone II EP2C35F672C6.
- *Codecs* de áudio.
- Entrada de vídeo com suporte aos formatos de Vídeo NTSC e PAL.
- Porta Serial RS232.
- Porta infravermelha.
- Entrada PS/2 para mouse e teclado.
- Entrada Ethernet 10/100.
- USB 2.0.
- Memórias externas: SDRAM, SRAM e FLASH.

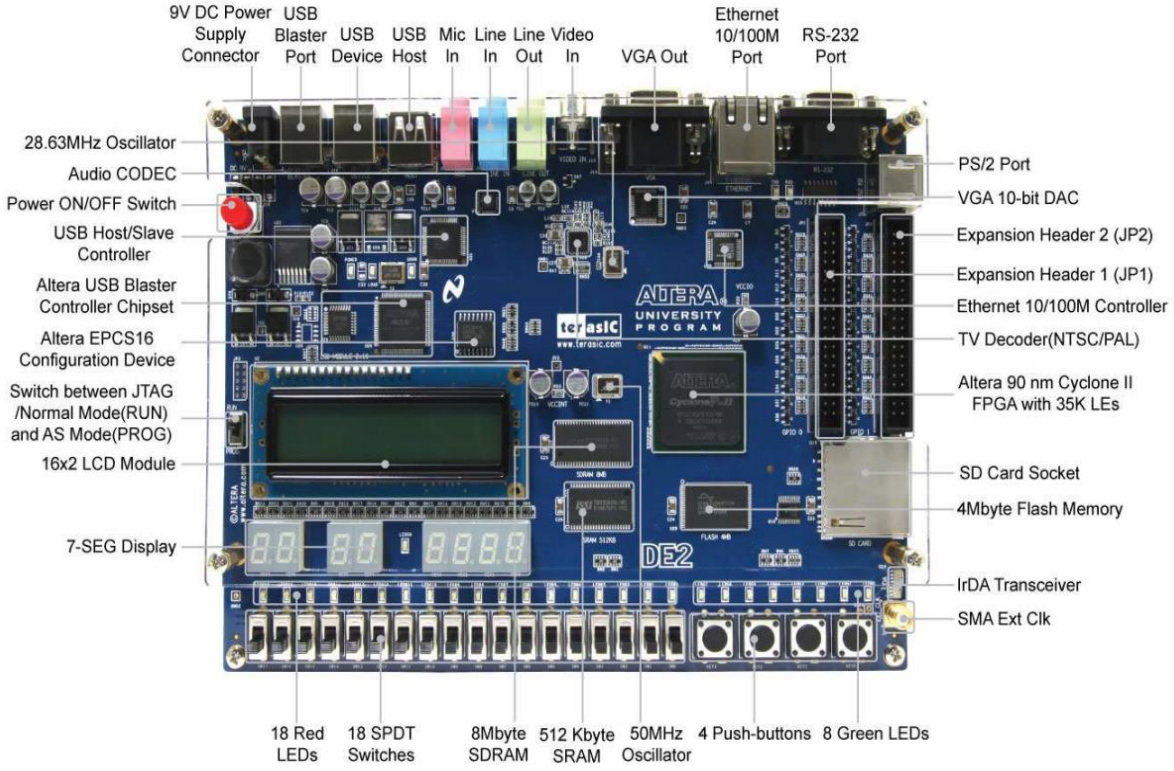

Figura 13: Placa de desenvolvimento Altera DE2. (ALTERA, 2012)

5.1.1 FPGA - ALTERA CYCLONE II 2C35

A FPGA Cyclone II 2C35 foi desenvolvida para obter alta *performance* e baixo consumo de energia. Ela tem 33.216 elementos lógicos, que são os blocos fundamentais

da FPGA. Além disso, contém 105 blocos de memória RAM M4K, 483.840 bits de RAM e 35 multiplicadores embarcados. O dispositivo também conta com 4 PLLs que fornecem multiplicação e divisão de *clock* e 475 pinos de entrada/saída (ALTERA, 2012).

5.1.2 VGA DAC - ADV7123

O dispositivo ADV7123 é um DAC de alta velocidade que contém três canais, cada um com 10 *bits*. Eles são usados para produzir os sinais analógicos vermelho, verde e azul. Este *chip* pode suportar resoluções de até 1600x1200 *pixels* a uma frequência de 100 MHz (ALTERA, 2012).

5.1.3 DISPOSITIVOS DE ENTRADA/SAÍDA

A plataforma Altera DE2 contém 4 botões (todos com *Schmitt Trigger* para *debounce*), 18 chaves interruptoras, 18 LEDs vermelhos, 9 LEDs verdes, 8 *displays* de 7 segmentos e um módulo de LCD, de 16x2 (ALTERA, 2012).

5.1.4 SRAM - IS61LV25616

A memória SRAM contida nesta plataforma pode armazenar até 512 KB de dados e pode ser acessada através do processador Nios II (ALTERA, 2012). Neste projeto, o papel a ser desempenhado por ela é de memória de programa.

5.1.5 SDRAM - A2V64S40CTP

Este *chip* tem capacidade de armazenamento igual a 8 MB, organizados em 4 bancos de 1 MBx16 *bits* para melhora de *performance* (ALTERA, 2012). Para correto funcionamento, esta memória necessita de um sinal de *clock* adiantado de -3ns em relação ao *clock* do sistema. A SDRAM, neste trabalho, terá a função de *buffer* de quadros devido a sua grande capacidade de armazenamento.

5.2 MÓDULO DE CÂMERA - TRDB D5M

Este módulo de câmera de 5 Megapixel, ver figura 14, foi desenvolvido pela empresa Terasic para ser utilizado com as plataformas de desenvolvimento Altera DE1, DE2 e DE4. Abaixo, estão os recursos de maior importância para o presente trabalho.

- Alta taxa de atualização de quadros.
- Fácil configuração de seus registradores através de sua interface I^2C.
- Controles programáveis de ganho, taxa de atualização, tamanho de quadro e tempo de exposição.
- *Reset* por *hardware* e *software*.
- Circuito de PLL integrado.
- Suporte a diferentes modos de leitura.
- Conexão com a plataforma Altera DE2 através dos pinos de expansão disponíveis nela.

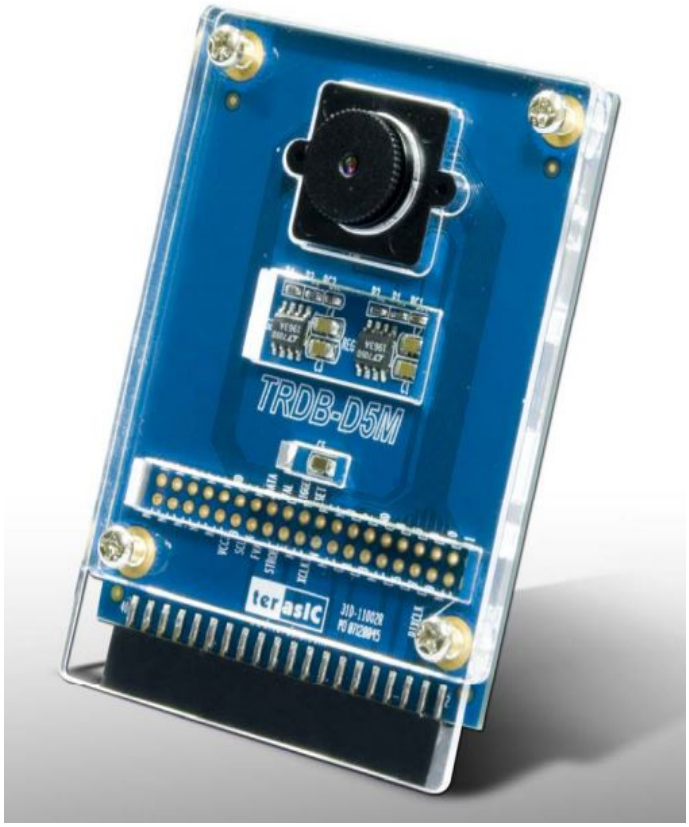

Figura 14: Módulo de Câmera TRDB-D5M. (TERASIC, 2009)

6 LINGUAGENS DE DESCRIÇÃO DE HARDWARE

São utilizadas na descrição formal de circuitos eletrônicos, mais comumente em circuitos digitais. Grande parte de projetos de sistemas digitais utilizam estas linguagens, pois elas possibilitam que o circuito seja sintetizado e simulado antes de qualquer implementação física. Além disso, permitem que sejam registradas as patentes dos projetos (PEDRONI, 2010).

Estas linguagens independem da tecnologia e de fabricante, de modo que são portáveis e reutilizáveis. Após implementado, o código pode ser utilizado para ser testado em uma FPGA ou empregado em uma fábrica para a criação de um *chip*.

6.1 VHDL

VHDL é uma linguagem descritora de *hardware* independente de fabricantes. O código descreve o comportamento de um *hardware* e o circuito físico é inferido pelo compilador. Tem como aplicações a síntese de circuitos digitais para FPGAs e CPLDs e também a fabricação de máscaras para ASICs (PEDRONI, 2010). É resultado de um projeto financiado pelo departamento de defesa dos Estados Unidos na década de 80, sendo a primeira linguagem de descrição de *hardware* padronizada pelo IEEE.

O código em VHDL é divido em 3 partes principais:

- Cabeçalho: para declarações de bibliotecas e pacotes.
- Entidade: para declarações das entradas e saídas.
- Arquitetura: para o código propriamente dito.

No Cabeçalho estão as declarações de biblioteca e pacotes que o compilador necessita para processar o projeto. Duas bibliotecas são inseridas por padrão: *std* e *work*. A primeira contém declarações dos dados básicos, como BIT e BOOLEAN, e a segunda parte detém o local onde está armazenado o projeto.

A Entidade é dividida em duas seções: PORT e GENERIC. PORT é a parte responsável pelas definições das portas de entrada e saída do sistema. Os sinais podem ser definidos em: IN, para os sinais que entram no circuito, OUT, para os sinais que saem do circuito, INOUT, para sinais bidirecionais, ou BUFFER, para sinais que são utilizados internamente.

A Arquitetura é a terceira parte de um código VHDL. Nela está o código do circuito que será desenvolvido. Ela é dividida em duas partes, a primeira é a declarativa onde podem ser criados sinais que serão utilizados internamente no circuito e desta forma não necessitam ser declarados na Entidade e a segunda parte representa a lógica que formará o circuito. Esta parte é separada da parte declarativa através da palavra reservada BEGIN.

Em VHDL é possível a realização de códigos concorrentes ou sequenciais. Códigos em VHDL tendem a serem concorrentes e, assim, para a utilização de uma programação sequencial é necessário a utilização de um processo (PROCESS), o qual fará com que o código respeite a ordem dos comandos.

6.2 VERILOG

É uma linguagem descritora de *hardware* padronizada pelo IEEE. O Verilog se diferencia do VHDL por ter uma semântica mais próxima à linguagem C, tornando a programação mais simples, e dando mais liberdade ao programador (DOTCOM, 2012).

O desenvolvimento é realizado em hierarquias, e cada uma delas comunica-se com outras através dos sinais que são declarados no cabeçalho do programa. Os sinais são divididos em entradas, saídas e bidirecionais (entrada/saída). Internamente, cada módulo pode conter a combinação de: declaração das variáveis, blocos concorrentes e sequenciais e instâncias de outros módulos.

Verilog é naturalmente concorrente e, para a utilização de blocos sequenciais é necessária a utilização de um bloco entitulado BEGIN.

7 COMUNICAÇÃO

Para que o sistema funcione corretamente é necessário a perfeita comunicação entre os módulos implementados. Sendo assim, a interconexão de componentes é de suma importância para o presente projeto.

7.1 BARRAMENTO I^2C

O barramento I^2C foi criado pela antiga Philips Semiconductors, agora NXP Semiconductors, e é um padrão adotado por vários dispositivos para interconexão entre periféricos (NXP SEMICONDUCTORS, 2014). Este protocolo utiliza um barramento de apenas dois fios para transmitir informações: a linha de dados serial (SDA) e a linha de *clock* (SCL). Ambas as linhas apresentam valor lógico padrão igual a "1", ou "alto", através de dois resistores *pull-up*. Por meio delas, cada dispositivo conectado ao barramento é mapeado através de um endereço único e pode atuar como transmissor ou receptor, sendo considerado mestre ou escravo dependendo da configuração. Um dispositivo mestre é aquele que inicia e termina a transferência de dados no barramento e gera o sinal de *clock*, que pode chegar até 400 kHz. Já o escravo é qualquer dispositivo endereçado para comunicação pelo mestre.

Todas as transações começam com um *bit* de início e são terminadas por outro de fim. Após a condição de início, o dispositivo mestre envia *byte* por *byte* até que finalize a comunicação, recebendo sempre um *bit* de confirmação ACK ou não confirmação NACK após cada *byte* enviado (ver figura 15).

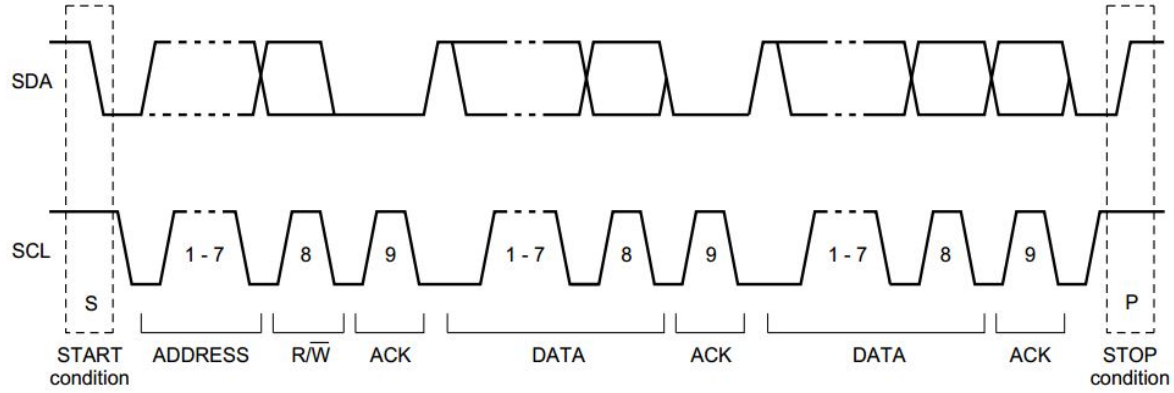

Figura 15: Transferência de dados completa, utilizando o protocolo I^2C. (NXP SEMICONDUCTORS, 2014)

Os sete *bits* mais significativos do primeiro *byte* são o endereço do escravo e o menos significativo indica se a transferência é de escrita ”0”, ou leitura ”1”. Enquanto a linha está livre, o barramento está na condição *idle*, ou seja, ambas as linhas estão altas. Uma transição de alto para baixo na linha SDA enquanto SCL estiver alto define a condição de início e uma transição de baixo para alto em SDA enquanto SCL estiver alto define a condição de fim. O módulo de câmera TRDB D5M, descrito na seção 5.2, utiliza o protocolo I^2C para configurar seus registradores de acordo com a necessidade do projeto.

7.2 INTERFACES ALTERA AVALON

As interfaces Avalon permitem que os componentes criados no *software* Qsys, explicado na subseção 4.1.2, se comuniquem em grande velocidade. São sete interfaces no total, mas as três mais importantes para este projeto são (ALTERA, 2015c):

- *Avalon Streaming Interface* (Avalon-ST): suporta o fluxo unidirecional de dados.
- *Avalon Memory Mapped Interface* (Avalon-MM): uma interface de escrita/leitura baseada na conexão mestre-escravo.
- *Avalon Conduit Interface*: interface que acomoda sinais externos ao sistema.

Cada interface tem o seu protocolo de comunicação e, assim, cada componente implementa automaticamente os protocolos das interfaces que utiliza. Caso o componente seja personalizado, deve-se implementar manualmente o protocolo para que ele consiga se comunicar com os outros módulos. Na figura 16, estão descritos os sinais de uma

comunicação através da *Avalon Streaming Interface*. Nela, pode-se observar que os dados são encapsulados em pacotes e há sinais de controle especificados pelo protocolo para que não haja problemas na comunicação.

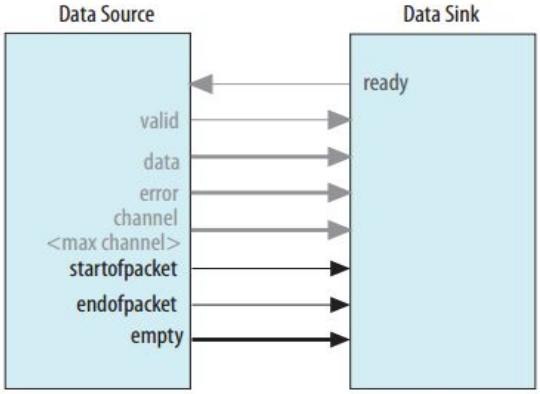

Figura 16: Exemplo de transferência de pacotes *Avalon Streaming* entre componentes. (ALTERA, 2015c)

8 IMPLEMENTAÇÃO

Como explicado anteriormente, neste projeto foi adotado a metodologia modular. Sendo assim, a implementação dele foi toda feita através de módulos, ou componentes. Para tal, foram utilizados os *software* Quartus II e Qsys para modelagem e descrição de *hardware*, em conjunto com o *plug-in* Eclipse Nios EDS. Além disso, foram utilizadas as linguagens VHDL e Verilog para todo o *hardware* descrito neste projeto.

8.1 VISÃO GERAL DO SISTEMA

Na figura 17, estão organizados num diagrama de blocos os componentes do sistema, dando assim uma visão geral. Em azul são mostrados os componentes disponibilizados pela Altera e em verde é destacado o componente de processamento de imagem desenvolvido pela dupla. Os dispositivos interfaceados pelos componentes estão em vermelho.

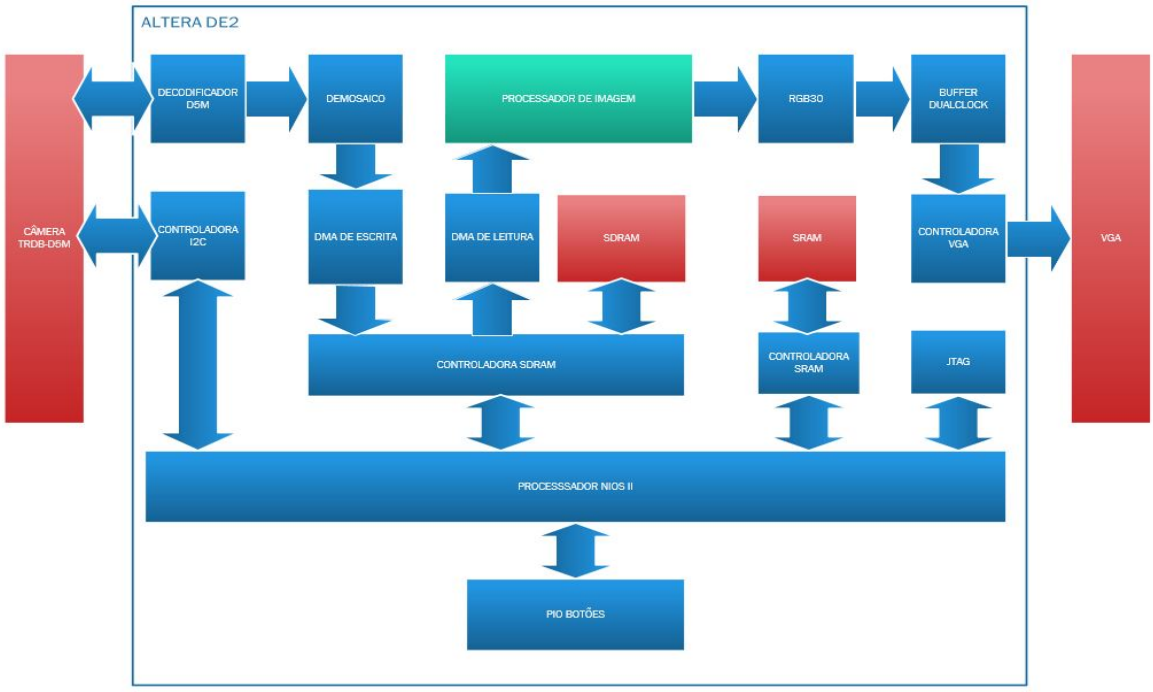

Figura 17: Visão geral do sistema organizado em módulos. (Fonte: Própria)

8.1.1 FLUXO DOS DADOS NO SISTEMA

Ao ligar o sistema, o processador começa a configuração da câmera através da inicialização da controladora I^2C. Terminada a configuração, o decodificador D5M começa a receber as imagens da câmera TRDB-D5M. Através da interface *Avalon Streaming*, as imagens são encapsuladas em pacotes e passadas quadro a quadro entre os módulos. O decodificador encaminha o quadro capturado para o demosaico que, em seguida, realiza a transformação da imagem no padrão *Bayer* para o formato RGB de 8 *bits* para cada um dos três canais. Feito isso, o quadro é encaminhado para a memória SDRAM, que age como um *buffer* de quadros, onde a imagem é armazenada através da controladora DMA de escrita. Logo após, os dados são lidos do *buffer* e transferidos para o processador de imagem, com o auxílio do DMA de leitura. Depois de processada, a imagem é passada para o formato RGB de 30 *bits* para atender ao requisito do DAC VGA. Feito isso, a imagem é enviada para a controladora VGA, que é encarregada de exibi-lá no monitor. Todos os módulos estão descritos nas tabelas 2 e 3 e suas implementações explicadas na próxima seção.

Tabela 2: Descrição dos módulos implementados neste projeto.
Parte 1 de 2. (Fonte: Própria)

Nome do Módulo	**Descrição**	**Autoria**
Processador Nios II/f	Responsável pela execução dos processos da camada de *software* e comunicação com outros componentes que gerenciam E/S e controle de barramento.	Altera
JTAG UART	É utilizado para descarregar a configuração de *hardware* na plataforma de desenvolvimento DE2, além de permitir o *debug* do sistema.	Altera
Controladora SRAM	Realiza a comunicação com a memória externa SRAM, responsável por armazenar o programa da camada de *software* do sistema.	Altera
Controladora I^2C	Responsável pela configuração dos registradores da câmera TRDB-D5M, através da implementação do protocolo I^2C	Altera
Decodificador D5M	Tem a função de decodificar a imagem transmitida pela câmera TRDB-D5M, utilizando os sinais de controle recebidos por ela.	Altera

Tabela 3: Descrição dos módulos implementados neste projeto.
Parte 2 de 2. (Fonte: Própria)

Nome do Módulo	**Descrição**	**Autoria**
Demosaico	Este módulo transforma a imagem inicialmente no padrão *Bayer* para o formato RGB de 24 *bits*. Para isso, é utilizado um algoritmo de interpolação.	Altera
Controladora DMA de escrita	Responsável pela escrita da imagem recebida do módulo de demosaico, um quadro por vez, na memória SDRAM.	Altera
PLL	Este módulo tem como função fornecer sinais de *clock* diferenciados para a controladora SDRAM, para controladora VGA e para a câmera D5M.	Altera
Controladora SDRAM	Responsável pela configuração e comunicação com a memória SDRAM externa.	Altera
Controladora DMA de leitura	Sua função é recuperar o quadro armazenado na memória SDRAM e transmiti-lo para o módulo de processamento de imagem.	Altera
Processador de Imagem	É neste módulo que está implementado o algoritmo de segmentação por limiarização. Nele a imagem é filtrada para que somente objetos de cor vermelha sejam visíveis.	Própria
RGB30	Ajusta os *pixels* da imagem para o formato requisitado pelo DAC VGA, que é o RGB 30 *bits*.	Altera
Buffer Dual-Clock	Tem a função de intermediar a comunicação entre o módulo RGB30 e a controladora VGA, que não estão no mesmo domínio de *clock*.	Altera
Controladora VGA	É responsável pela temporização correta do padrão VGA para a resolução de 640x480 e taxa de atualização de 60Hz.	Altera
PIO Botões	Este módulo é responsável por mapear os botões de entrada/saída para que eles tenham a função de alterar o tempo de exposição dos sensores da câmera.	Altera

8.2 IMPLEMENTAÇÃO DO SISTEMA - QSYS

Os componentes do sistema foram implementados no *software* Qsys e podem ser vistos na figura 18.

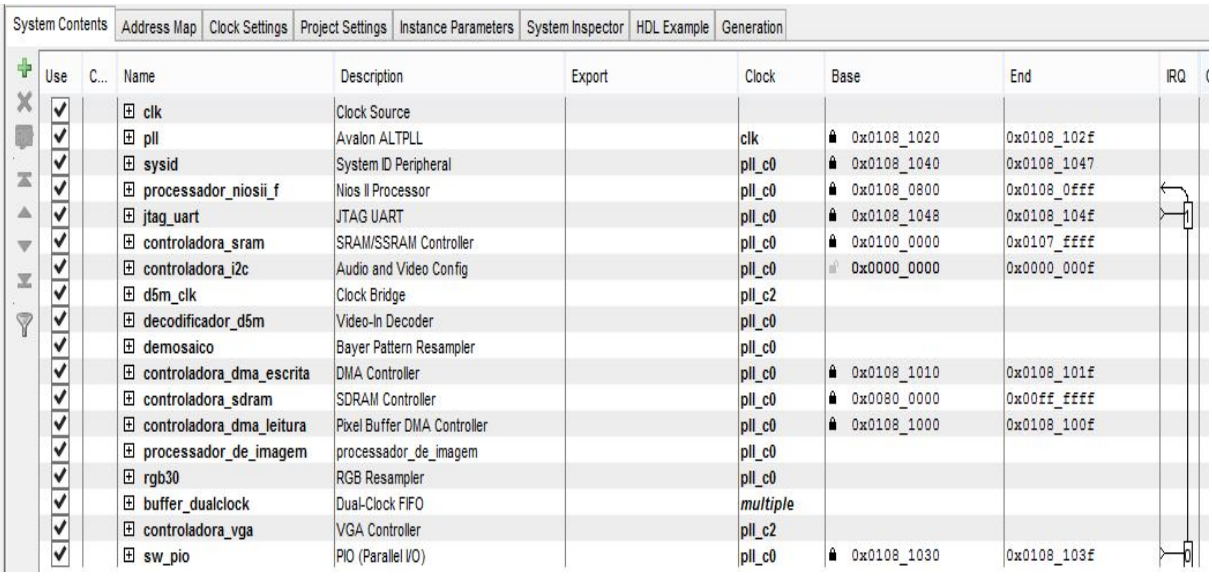

Figura 18: Componentes implementados no *software* Qsys. (Fonte: Qsys)

Eles se comunicam entre si utilizando as interfaces Avalon, explicadas na seção 7.2. Essas interfaces funcionam como um barramento de comunicação entre os componentes. A figura 19 ilustra a comunicação através do barramento Avalon.

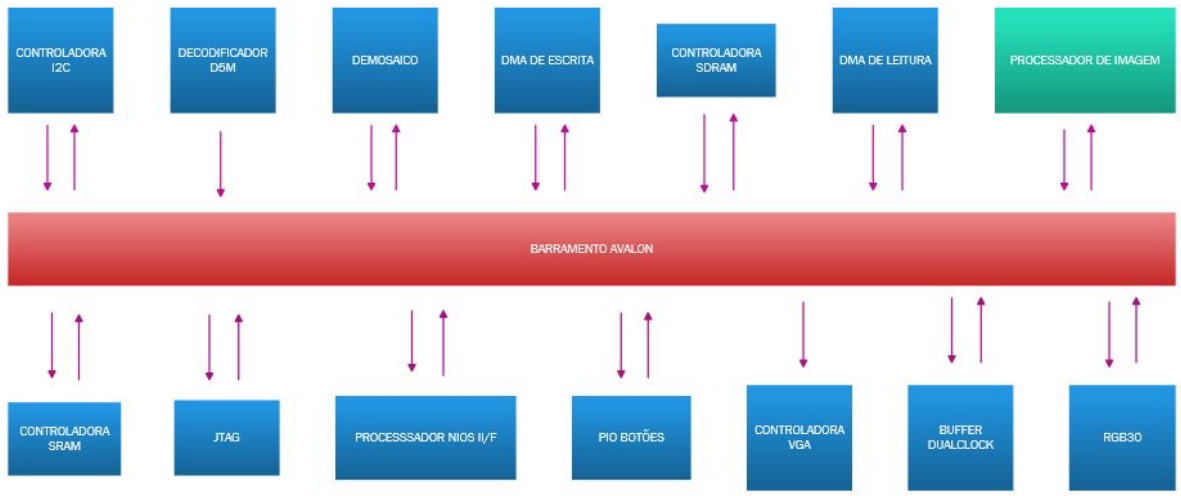

Figura 19: Interconexão dos componentes através do barramento Avalon. (Fonte: Própria)

A seguir, serão explicadas a implementação e configuração dos componentes mais importantes no Qsys.

8.2.1 PROCESSADOR NIOS II/F

O processador Nios II implementado foi o tipo f, por ser o mais rápido. Neste projeto, ele opera com frequência de *clock* de 50 MHz, que é o padrão do componente. Foi configurada uma memória *cache* de instrução com 4kB. Além disso, foram habilitadas operações aritméticas de multiplicação e divisão em *hardware*, para melhor *performance*. A tela de configuração pode ser vista na figura 20.

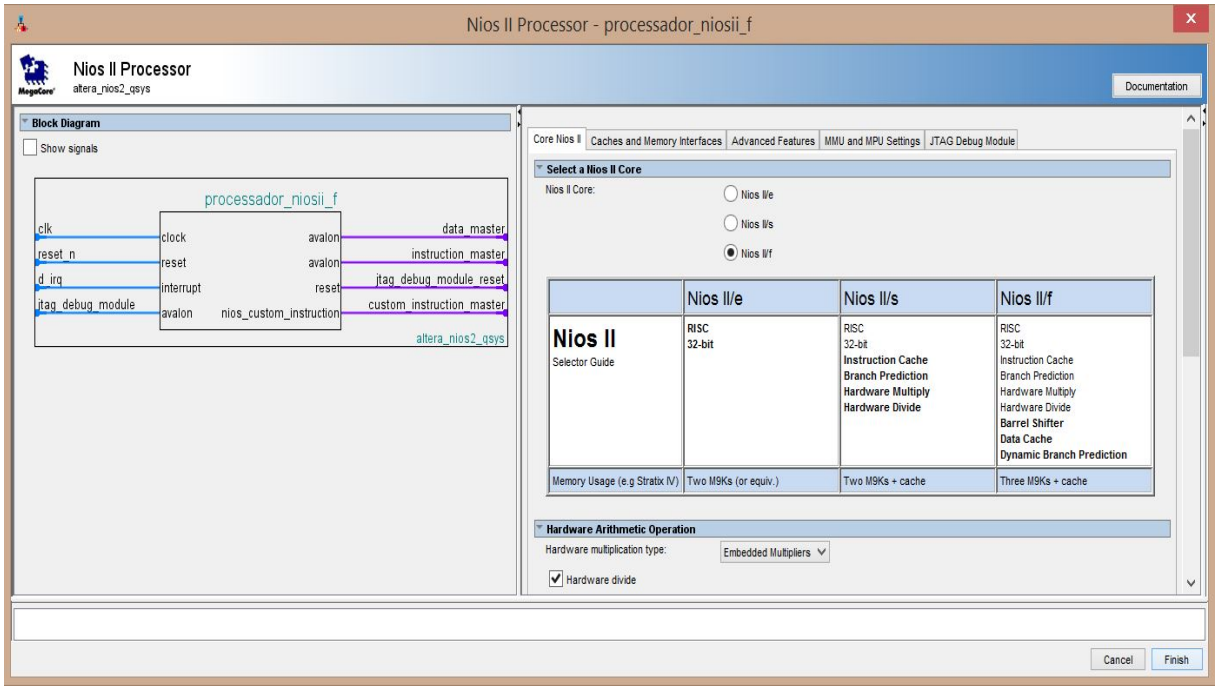

Figura 20: Tela de configuração do processador Nios II. (Fonte: Qsys)

8.2.2 CONTROLADORA SRAM

A memória SRAM foi utilizada como memória de programa, armazenando as instruções do processador. Para o correto interfaceamento dela, foi necessário a utilização de uma controladora SRAM específica para o *chip* descrito na subseção 5.1.2, como visto na figura 21. Nela, podem ser vistos os sinais de controle, escrita e leitura de dados para memória.

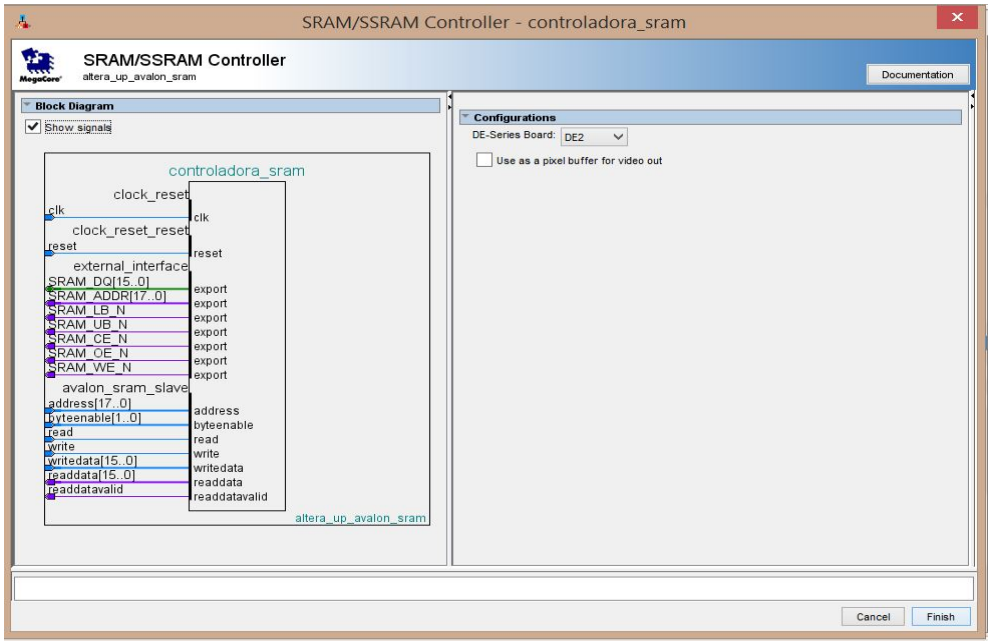

Figura 21: Tela de configuração da controladora SRAM. (Fonte: Qsys)

8.2.3 CONTROLADORA I^{2}C

Para configuração do módulo D5M, foi utilizada a controladora I^{2}C. Ela foi configurada para auto inicialização da câmera quando o sistema é ligado. Também foi habilitado o sinal de configuração da exposição à luz dos sensores da câmera. A configuração deste componente é vista na figura 22, onde também são explicitados os sinais para comunicação serial e controle de exposição.

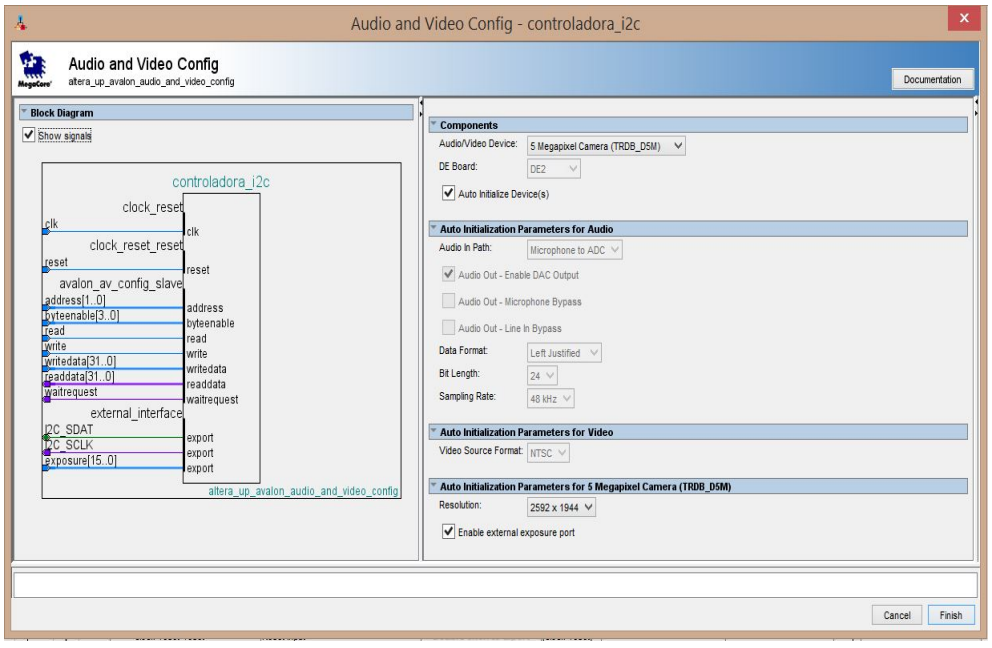

Figura 22: Tela de configuração da controladora I^{2}C. (Fonte: Qsys)

8.2.4 DECODIFICADOR D5M

Este componente decodifica o vídeo gerado pela câmera, quadro por quadro, no formato padrão Bayer. Ele possui um *buffer* FIFO interno para sincronização dos *clocks* da câmera e do sistema, como visto na figura 23. O parâmetro para configuração deste componente foi a seleção do módulo D5M como fonte de vídeo.

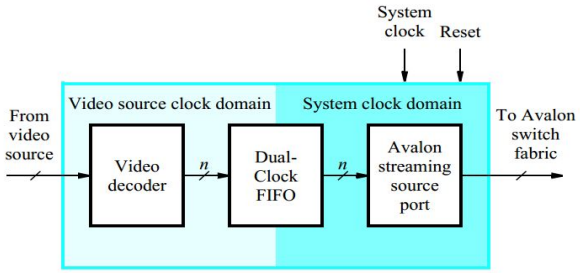

Figura 23: Diagrama de bloco do Decodificador D5M. (ALTERA, 2013b)

8.2.5 DEMOSAICO

Após a decodificação, a imagem é convertida para o formato RGB de 24 *bits*, utilizando o algoritmo de demosaico do vizinho mais próximo (descrito na subseção 3.3.3). A configuração deste componente é realizada indicando a câmera D5M como fonte de vídeo e, também, o tamanho de cada quadro de entrada como 1280x960. Isso resulta numa imagem de 640x480 na saída (figura 24).

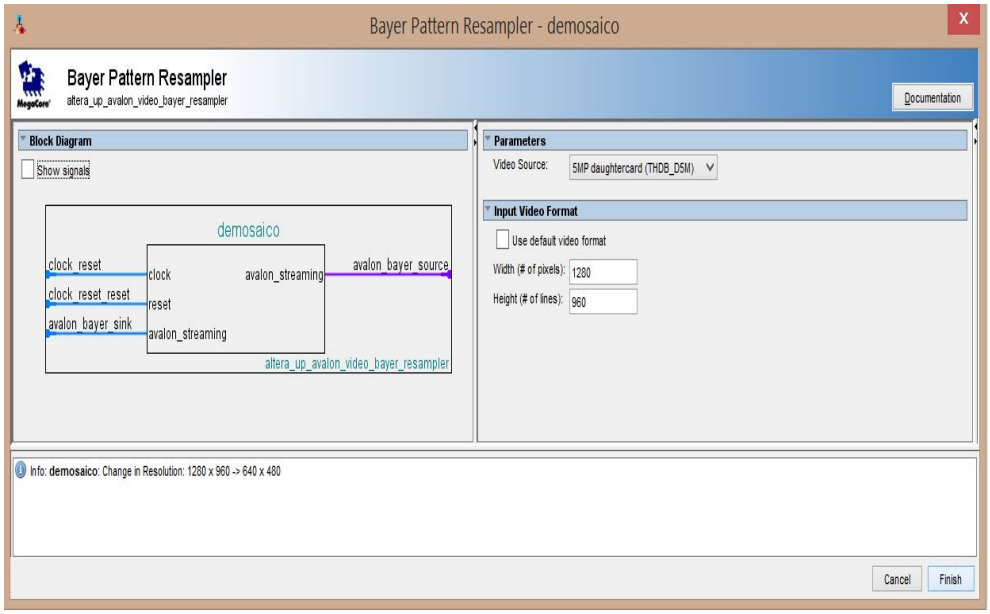

Figura 24: Tela de configuração do componente Demosaico. (Fonte: Qsys)

8.2.6 CONTROLADORA DMA DE ESCRITA

A controladora DMA de escrita é configurada para salvar a imagem na memória SDRAM, cujo endereço no sistema é 0x00800000. Seus parâmetros de configuração são indicados na imagem da figura 25.

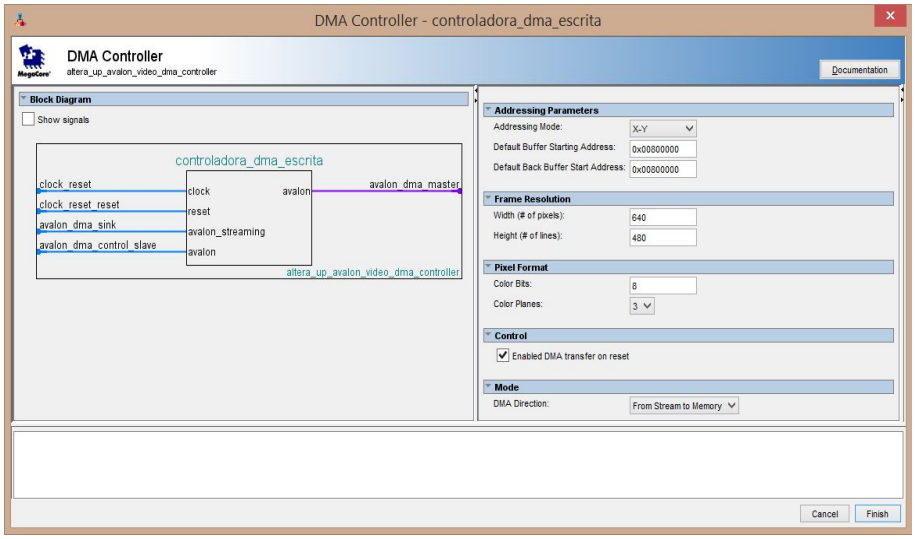

Figura 25: Tela de configuração da Controladora DMA de escrita. (Fonte: Qsys)

O modo escolhido para endereçamento da imagem na memória é o X-Y. Nele, o endereço contém as coordenadas x e y do *pixel*. O formato de endereço para a resolução de 640x480 é visto na figura 26.

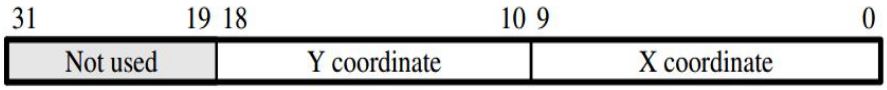

Figura 26: Formato de endereçamento para o modo X-Y. (ALTERA, 2013b)

8.2.7 CONTROLADORA SDRAM

Para interfacear a memória SDRAM foi necessária a utilização de uma controladora, pois a SDRAM exige uma lógica de controle para operações de atualização e temporização. A memória foi ajustada para ter uma largura de 16 *bits* com 4 bancos e largura de endereço de 12 linhas por 8 colunas, obedecendo os parâmetros descritos no *datasheet* do *chip*. Sua configuração completa pode ser vista na figura 27.

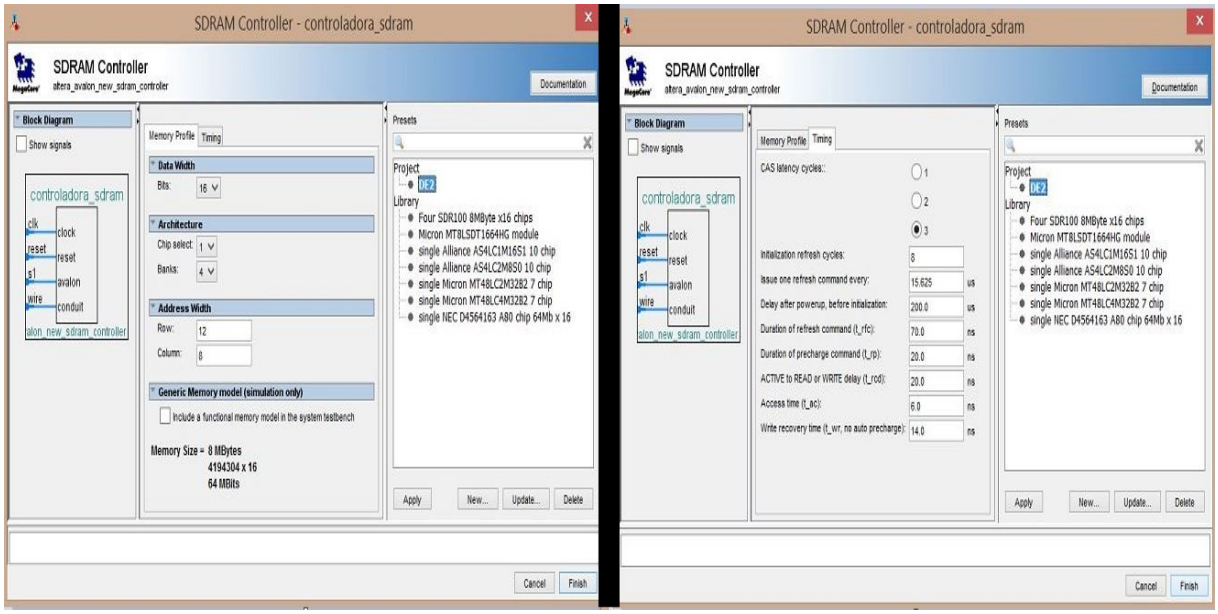

Figura 27: Telas de configuração da Controladora SDRAM. (Fonte: Qsys)

8.2.8 PLL

O PLL foi configurado para fornecer sinais de *clock* de 50MHz adiantado de -3ns para a memória SDRAM externa e 25MHz para ambas a controladora VGA e a câmera D5M. Essas configurações podem ser vistas nas imagens da figura 28.

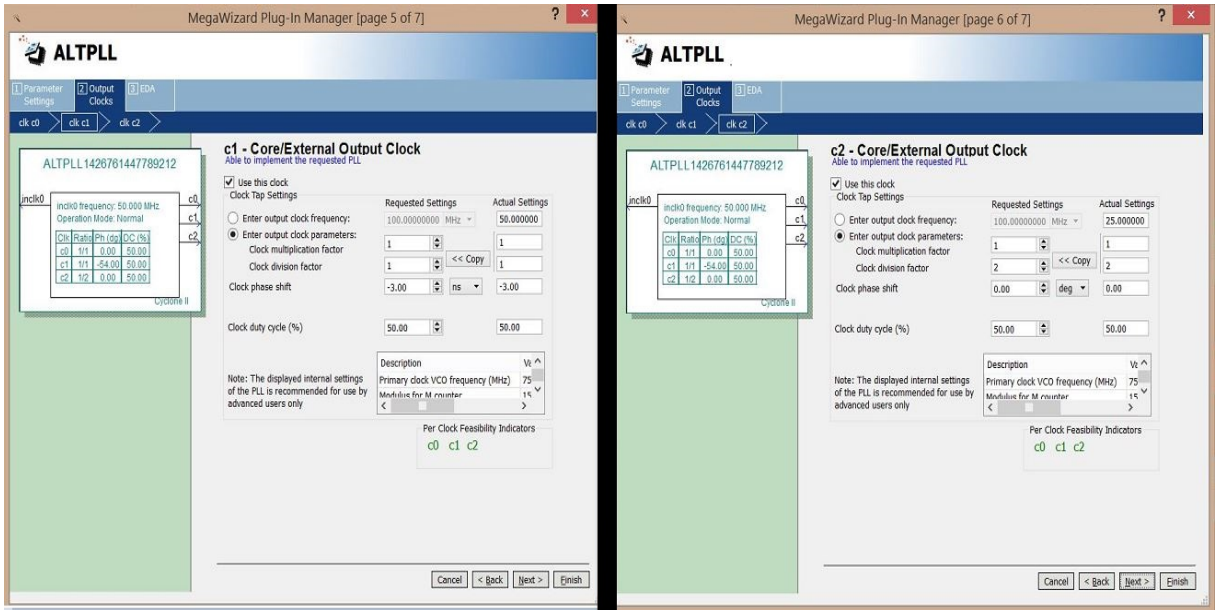

Figura 28: Telas de configuração dos *clocks* no componente PLL. (Fonte: Qsys)

8.2.9 CONTROLADORA DMA DE LEITURA

A controladora DMA de leitura é configurada para resgatar a imagem da memória SDRAM e enviar para o componente de processamento de imagem. Seus parâmetros de configuração são indicados na imagem 29.

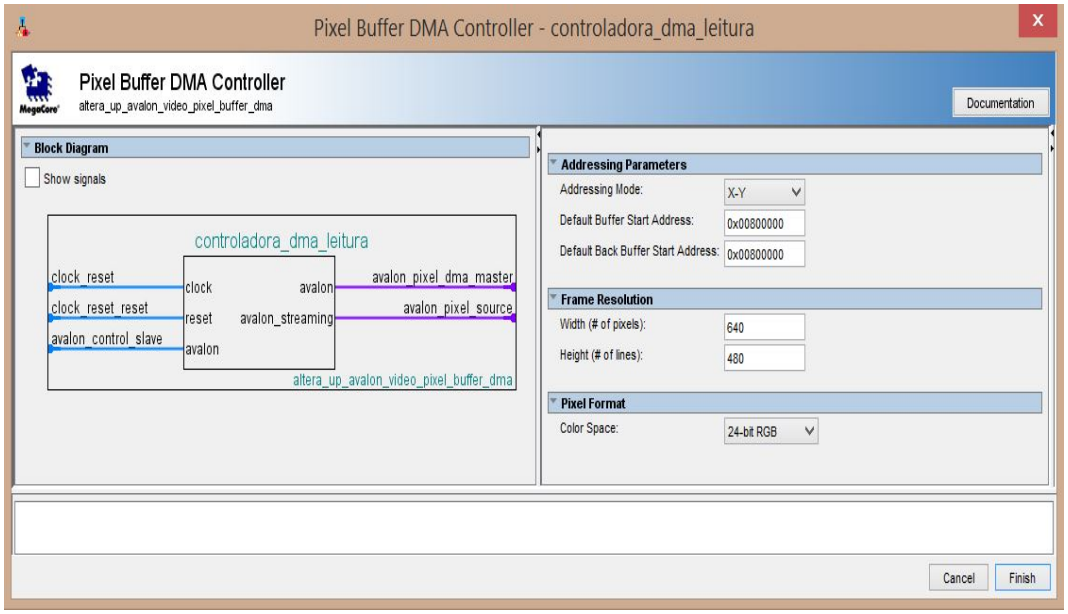

Figura 29: Tela de configuração da Controladora DMA de leitura. (Fonte: Qsys)

8.2.10 PROCESSAMENTO DE IMAGEM

Foi necessária a criação de dois arquivos VHDL na implementação deste componente: um para a interface Avalon e outro para o processamento de imagem. O arquivo da interface é o principal e contém os sinais de comunicação, como visto na figura 30, além da instância do componente de processamento.

```
entity st_interface is
   port(
      clock                   : in  std_logic;
      reset_n                 : in  std_logic;
      -- Sink
      stream_in_ready         : out std_logic;
      stream_in_startofpacket : in  std_logic;
      stream_in_endofpacket   : in  std_logic;
      stream_in_valid         : in  std_logic;
      stream_in_data          : in  std_logic_vector(23 downto 0);
      -- Source
      stream_out_ready        : in  std_logic;
      stream_out_startofpacket : out std_logic;
      stream_out_endofpacket  : out std_logic;
      stream_out_valid        : out std_logic;
      stream_out_data         : out std_logic_vector(23 downto 0)
   );
end st_interface;
```

Figura 30: Trecho do arquivo VHDL da interface Avalon explicitando os sinais de comunicação. (Fonte: Quartus II)

Para que o módulo de processamento de imagem seja capaz de receber e transmitir a imagem para os outros componentes do sistema pelo barramento Avalon, foi necessário a implementação da interface *Avalon Streaming*. Nela, o componente transmissor é chamado de *source* e o receptor de *sink*.

Cinco sinais foram necessários em ambos os sentidos para que a comunicação seja eficiente (ALTERA, 2015c):

- data[23..0]: representa os *pixels* no formato RGB de 24 *bits* e é transmitido do *source* para o *sink*.
- startofpacket: sinal de controle para indicar o início do pacote de dados e é gerado pelo componente *source*.
- endofpacket: sinal de controle para indicar o fim do pacote de dados e é gerado pelo componente *source*.
- ready: sinal gerado pelo componente *sink* para avisar o componente trasmissor que pode iniciar a transmissão.
- valid: gerado pelo próprio componente *source*, sua função é validar os outros sinais de transmissão.

Já o arquivo VHDL de processamento de imagem implementa o algoritmo de segmentação por limiarização (explicado na subseção 3.3.4), como pode ser visto na figura 31.

```
    constant th : integer := 85;
    signal   r  : integer := 0;
    signal   g  : integer := 0;
    signal   b  : integer := 0;

begin
    process(clock)
    begin
        if(rising_edge(clock)) then
            if(reset_n = '0') then
                data_out <= "000000000000000000000000";
            elsif(th < r-(b+g)) then
                data_out <= "111111111111111111111111";
            else
                data_out <= "000000000000000000000000";
            end if;
        end if;
    end process;
    r <= to_integer(unsigned(data_in(23 downto 16)));
    g <= to_integer(unsigned(data_in(15 downto 8)));
    b <= to_integer(unsigned(data_in(7 downto 0)));
```

Figura 31: Trecho do arquivo VHDL do Processador de Imagem explicitando a implementação do algoritmo de segmentação por limiarização. (Fonte: Quartus II)

Após terminados os arquivos da interface e de processamento, é iniciada a criação do componente. Primeiramente, são adicionados os arquivos no editor de componentes do Qsys, como visto na figura 32. Logo após, é feita uma análise deles para verificar se tem algum erro. Feito isso, os sinais contidos no arquivo de interface são configurados para as interfaces Avalon correspondentes (ver figura 33), finalizando a criação do componente.

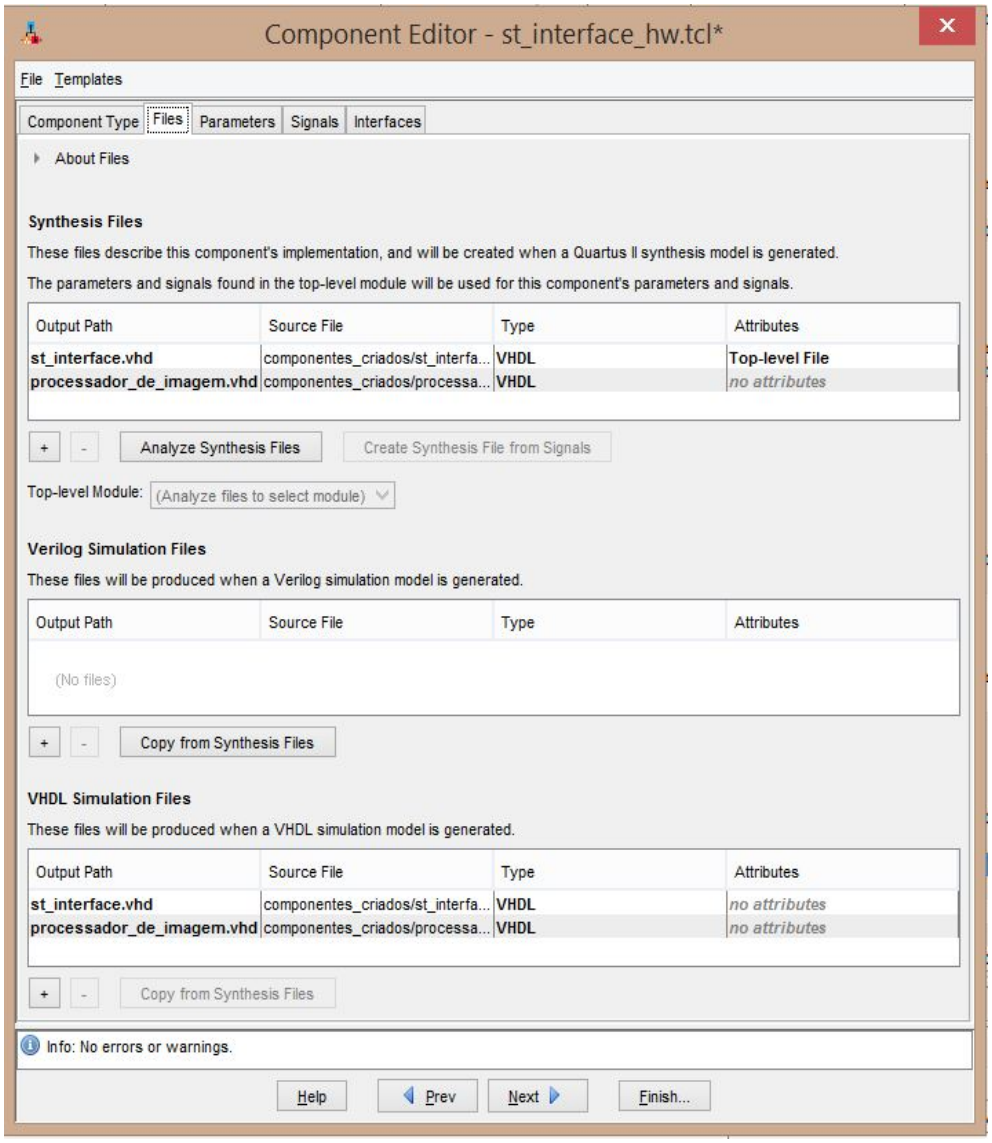

Figura 32: Tela de edição do componente Processador de Imagem para adição de arquivos. (Fonte: Qsys)

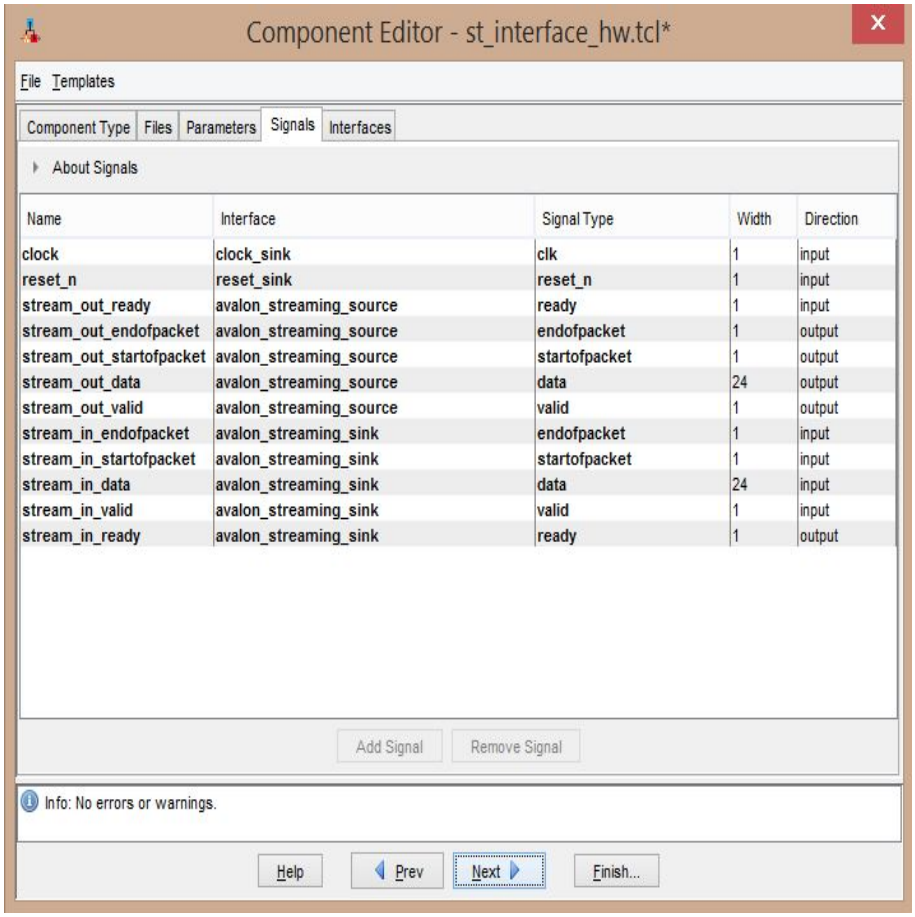

Figura 33: Tela de edição do componente Processador de Imagem para configuração dos sinais. (Fonte: Qsys)

Na figura 34 é exibido o componente de processamento de imagem criado no editor, explicitando os sinais de comunicação.

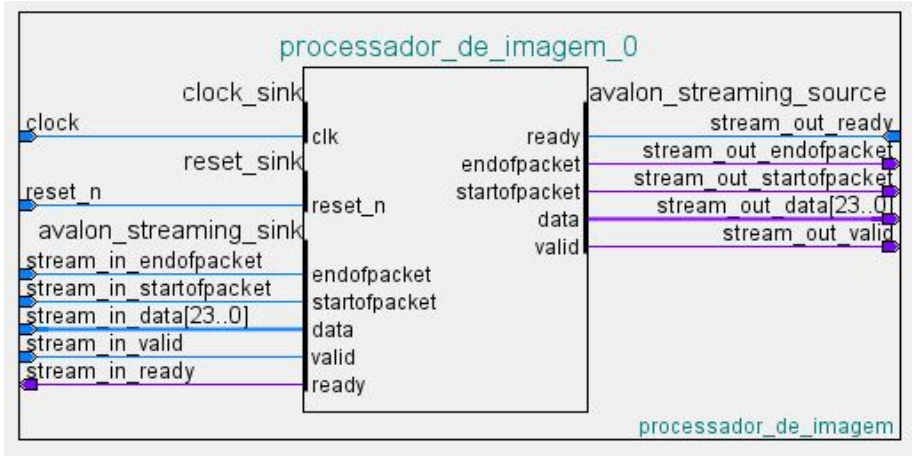

Figura 34: Detalhes da interface *Avalon Streaming* implementada no componente de processamento de imagem. (Fonte: Qsys)

8.2.11 RGB30

Este componente foi configurado para receber a imagem no formato 24 *bits* e formata-lá para o padrão RGB de 30 *bits*, enviando o quadro para o *Buffer Dual-Clock* através do barramento Avalon (Ver figura 35).

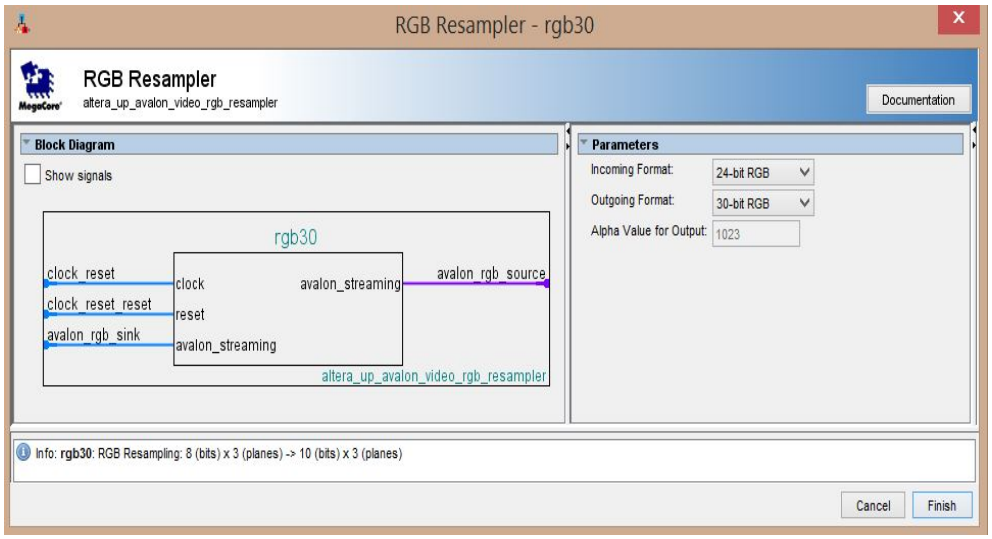

Figura 35: Tela de configuração do componente RGB30. (Fonte: Qsys)

8.2.12 BUFFER DUAL-CLOCK

Este componente foi configurado para receber a imagem no formato RGB de 30 *bits* e enviar para a controladora VGA, como visto na figura 36, alterando o domínio do *clock* de 50MHz na entrada para 25MHz na saída.

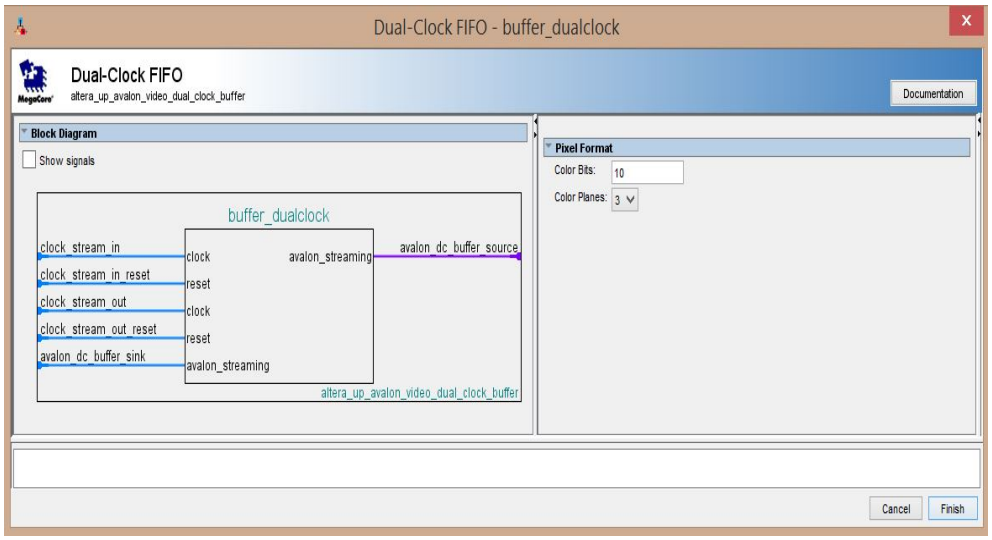

Figura 36: Tela de configuração do componente *Buffer Dual-Clock*. (Fonte: Qsys)

8.2.13 CONTROLADORA VGA

A controladora VGA foi configurada para a plataforma DE2, oferecendo como saída uma imagem com resolução de 640x480. Para tal configuração, as temporizações necessárias para sincronização VGA geradas pela controladora para um *clock* de 25MHz são descritas na imagem da figura 37.

VGA mode		Horizontal Timing Spec				
Configuration	Resolution(HxV)	a(us)	b(us)	c(us)	d(us)	Pixel clock(Mhz)
VGA(60Hz)	640x480	3.8	1.9	25.4	0.6	25 (640/c)

VGA mode		Vertical Timing Spec			
Configuration	Resolution (HxV)	a(lines)	b(lines)	c(lines)	d(lines)
VGA(60Hz)	640x480	2	33	480	10

Figura 37: Especificação das temporizações da VGA na plataforma DE2. (ALTERA, 2012)

8.2.14 PIO BOTÕES

Este componente foi configurado como um dispositivo de entrada para que, quando apertado os botões, o tempo de exposição à luz dos sensores da câmera sejam ajustados tanto para mais como para menos. Para tal, foi habilitada a sua interrupção síncrona com a borda de descida do *clock* do sistema, gerando uma IRQ para o processador Nios II, como visto na figura 38.

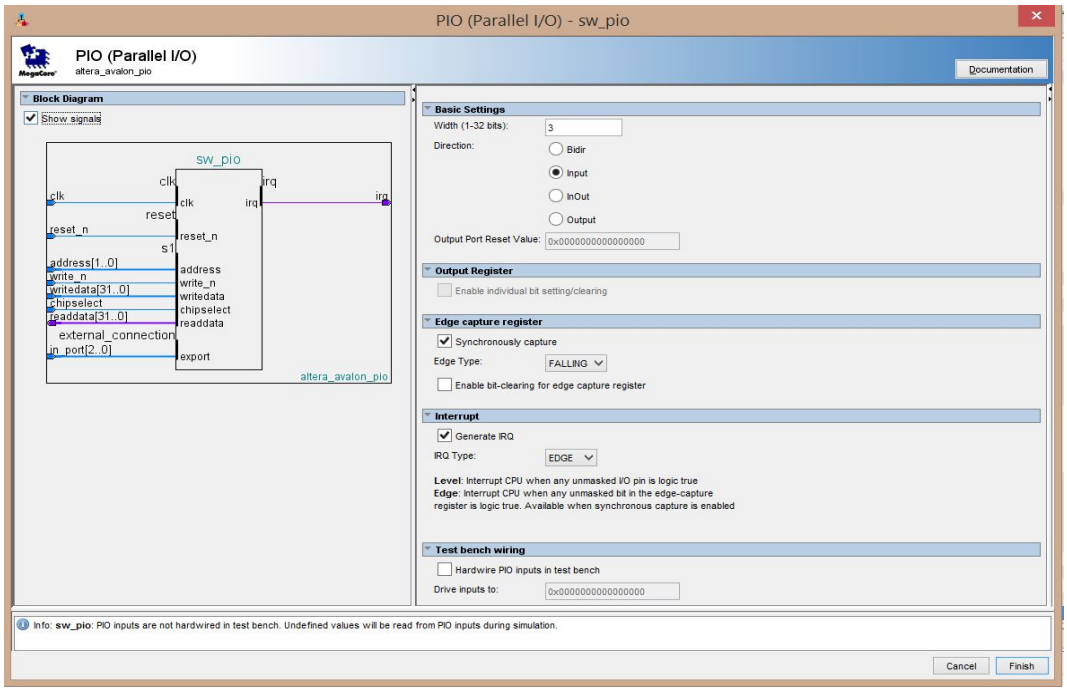

Figura 38: Tela de configuração do componente PIO Botões. (Fonte: Qsys)

8.3 IMPLEMENTAÇÃO DO SISTEMA - QUARTUS II

Após criado o sistema no ambiente Qsys, é necessária a integração dele ao projeto no Quartus II. Isso é feito através da instanciação dele no arquivo VHDL principal do projeto, como visto na figura 39.

```
component nios_system
    port(
        signal clk_clk                                  : in    std_logic;
        signal reset_reset_n                            : in    std_logic;
        signal altpll_0_c1_clk                          : out   std_logic;
        signal d5m_clk_out_clk_clk                      : out   std_logic;
        -- SRAM
        signal sram_external_interface_DQ    : inout std_logic_vector(15 downto 0);
        signal sram_external_interface_ADDR  : out   std_logic_vector(17 downto 0);
        signal sram_external_interface_LB_N  : out   std_logic;
        signal sram_external_interface_UB_N  : out   std_logic;
        signal sram_external_interface_CE_N  : out   std_logic;
        signal sram_external_interface_OE_N  : out   std_logic;
        signal sram_external_interface_WE_N  : out   std_logic;
        -- Decodificador de Vídeo
        signal d5m_decoder_external_interface_PIXEL_CLK   : in    std_logic;
        signal d5m_decoder_external_interface_LINE_VALID  : in    std_logic;
        signal d5m_decoder_external_interface_FRAME_VALID : in    std_logic;
        signal d5m_decoder_external_interface_PIXEL_DATA  : in    std_logic_vector(11 downto 0);
        -- Inicializadora da Câmera D5M
        signal d5m_config_external_interface_I2C_SDAT     : inout std_logic;
        signal d5m_config_external_interface_I2C_SCLK     : out   std_logic;
        signal d5m_config_external_interface_exposure     : in    std_logic_vector(15 downto 0);
        -- SDRAM Pixel Buffer
        signal sdram_controller_wire_addr                 : out   std_logic_vector(11 downto 0);
        signal sdram_controller_wire_ba                   : out   std_logic_vector(1 downto 0);
        signal sdram_controller_wire_cas_n                : out   std_logic;
        signal sdram_controller_wire_cke                  : out   std_logic;
        signal sdram_controller_wire_cs_n                 : out   std_logic;
        signal sdram_controller_wire_dq                   : inout std_logic_vector(15 downto 0) := (others => 'X');
        signal sdram_controller_wire_dqm                  : out   std_logic_vector(1 downto 0);
        signal sdram_controller_wire_ras_n                : out   std_logic;
        signal sdram_controller_wire_we_n                 : out   std_logic;
        -- SW PIO
        signal sw_pio_external_connection_export          : in std_logic_vector(2 downto 0);
        -- VGA
        signal vga_controller_external_interface_CLK      : out   std_logic;
        signal vga_controller_external_interface_HS       : out   std_logic;
        signal vga_controller_external_interface_VS       : out   std_logic;
        signal vga_controller_external_interface_BLANK    : out   std_logic;
        signal vga_controller_external_interface_SYNC     : out   std_logic;
        signal vga_controller_external_interface_R        : out   std_logic_vector(9 downto 0);
        signal vga_controller_external_interface_G        : out   std_logic_vector(9 downto 0);
        signal vga_controller_external_interface_B        : out   std_logic_vector(9 downto 0));
end component;
```

Figura 39: Instanciamento do Sistema Nios em VHDL. (Fonte: Quartus II)

Além disso, são necessárias as atribuições dos pinos da plataforma DE2 aos sinais do sistema de acordo com o manual, como mostra a figura 40.

```
entity embedded_project is
    port(
        -- Entrada
        CLOCK_50   : in     std_logic;
        KEY        : in     std_logic_vector(3 downto 0);
        -- Câmera D5M
        GPIO_1     : inout  std_logic_vector(35 downto 0);
        -- SRAM
        SRAM_DQ    : inout std_logic_vector (15 downto 0);
        SRAM_ADDR  : out   std_logic_vector (17 downto 0);
        SRAM_CE_N  : out   std_logic;
        SRAM_WE_N  : out   std_logic;
        SRAM_OE_N  : out   std_logic;
        SRAM_UB_N  : out   std_logic;
        SRAM_LB_N  : out   std_logic;
        -- SDRAM
        DRAM_DQ    : inout  std_logic_vector (15 downto 0);
        DRAM_ADDR  : out    std_logic_vector (11 downto 0);
        DRAM_BA_1  : buffer std_logic;
        DRAM_BA_0  : buffer std_logic;
        DRAM_CAS_N : out    std_logic;
        DRAM_RAS_N : out    std_logic;
        DRAM_CLK   : out    std_logic;
        DRAM_CKE   : out    std_logic;
        DRAM_CS_N  : out    std_logic;
        DRAM_WE_N  : out    std_logic;
        DRAM_UDQM  : buffer std_logic;
        DRAM_LDQM  : buffer std_logic;
        -- VGA
        VGA_CLK    : out   std_logic;
        VGA_HS     : out   std_logic;
        VGA_VS     : out   std_logic;
        VGA_BLANK  : out   std_logic;
        VGA_SYNC   : out   std_logic;
        VGA_R      : out   std_logic_vector (9 downto 0);
        VGA_G      : out   std_logic_vector (9 downto 0);
        VGA_B      : out   std_logic_vector (9 downto 0)
    );
end embedded_project;
```

Figura 40: Sinais atribuídos aos pinos dos periféricos. (Fonte: Quartus II)

Após esta etapa, o sistema Nios esta completo e o bloco gerado para o componente no Quartus II pode ser visto na figura 41. Nele, são explicitados os grupos de sinais para interfaceamento com os botões *switch*, a saída VGA, as memórias externas SRAM e SDRAM e a câmera D5M.

Figura 41: Bloco do sistema Nios implementado no *software* Qsys. (Fonte: Quartus II)

8.4 IMPLEMENTAÇÃO DO SISTEMA - NIOS EDS

Com o sistema criado no Qsys e instanciado no projeto principal do Quartus II, pode-se iniciar a elaboração do programa que o processador Nios irá executar. Este *software* é criado no *plug-in* Eclipse Nios EDS.

O programa foi necessário para inicialização dos componentes e tratamento de interrupções dos botões. Estes foram implementados para alterar o tempo de exposição dos sensores da câmera TRDB-D5M, disparando interrupções para que o processador Nios II configure o registrador correspondente da câmera. Além disso, para que o módulo funcione corretamente, o processador automaticamente inicializa o componente Controladora I^2C através da biblioteca HAL assim que o sistema inicializa, como mostra a figura 42.

```
void alt_sys_init( void )
{
    ALTERA_AVALON_JTAG_UART_INIT ( JTAG_UART, jtag_uart);
    ALTERA_AVALON_SYSID_QSYS_INIT ( SYSID, sysid);
    ALTERA_UP_AVALON_AUDIO_AND_VIDEO_CONFIG_INIT ( D5M_CONFIG, d5m_config);
    ALTERA_UP_AVALON_VIDEO_PIXEL_BUFFER_DMA_INIT ( PIXEL_BUFFER, pixel_buffer);
}
```

Figura 42: Função de inicialização de alguns componentes do sistema. (Fonte: Nios EDS)

A configuração da interrupção no componente dos botões foi realizada através da criação de uma função de inicialização, como visto na figura 43.

```
33 static void init_sw_interrupts() {
34     /* Recast the edge_capture pointer to match the
35     alt_irq_register() function prototype. */
36     void* edge_capture_ptr = (void*) &edge_capture;
37     /* Enable all 4 button interrupts. */
38     IOWR_ALTERA_AVALON_PIO_IRQ_MASK(SW_PIO_BASE, 0x7);
39     /* Reset the edge capture register. */
40     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(SW_PIO_BASE, 0x0);
41     /* Register the ISR. */
42     #ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
43     alt_ic_isr_register(SW_PIO_IRQ_INTERRUPT_CONTROLLER_ID, SW_PIO_IRQ, handle_sw_interrupts, edge_capture_ptr, 0x0)
44     #else
45     alt_irq_register( SW_PIO_IRQ, edge_capture_ptr, handle_sw_interrupts);
46     #endif
47 }
```

Figura 43: Função para a inicialização das interrupções no componente dos botões. (Fonte: Nios EDS)

8.5 CONFIGURAÇÃO DA CÂMERA

A configuração da câmera TRDB-D5M é realizada através da escrita em qualquer um de seus registradores de 16 *bits*. Para escrever neles foi necessária a utilização do barramento I^2C.

Por padrão a câmera transmite os *pixels* à uma taxa de 25 MHz. Pela necessidade de projeto esta frequência foi alterada para 50 MHz, adequando-se ao sistema. Isto foi realizado através da configuração do PLL interno do módulo.

Outras configurações foram necessárias para que a imagem fosse corretamente adquirida. Na tabela 4, estão os registradores que foram ajustados, junto de suas descrições e valores.

Tabela 4: Descrição dos principais registradores e configurações da câmera TRDB-D5M. (TERASIC, 2009)

Nome do Registrador	**Endereço**	**valor**	**Descrição**
Read Mode	0x20	0xC000	Habilita o modo de leitura espelho de todas as linhas e colunas do sensor.
Shutter Width Lower	0x09	exposicão	Ajusta o tempo de exposição à luz.
Horizontal Blank	0x05	0x0000	Define o tempo adicionado na leitura ao final de cada linha. Quanto maior o valor, mais exposto à luz o sensor fica e, consequentemente, menor a taxa de atualização de quadros.
Vertical Blank	0x06	0x0019	Define o tempo adicionado na leitura ao final de cada quadro. Quanto maior o valor, mais exposto à luz o sensor fica e, consequentemente, menor a taxa de atualização de quadros.
Pixel Clock Control	0x0A	0x8000	Configura os sinais LVAL, FVAL e DATA para serem capturados na borda de subida.
PLL Control	0x10	0x0051	Habilita o PLL interno da câmera.
PLL Config 1	0x11	0x1801	Configura o PLL interno da câmera para multiplicar o *clock* de entrada por 24.
PLL Config 2	0x12	0x0005	Configura o PLL interno da câmera para dividir o *clock* de entrada por 12.
PLL Control	0x10	0x0053	Após a configuração do PLL, essa opção faz com que o módulo de câmera utilize o *clock* gerado pelo PLL.
Test Pattern Control	0xA0	0x0000	Desabilita o teste de padrão.
Row Start	0x01	0x0036	Ajusta o início de leitura do sensor na linha 54.
Column Start	0x02	0x0010	Ajusta o início de leitura do sensor na coluna 16.
Row Size	0x03	0x077F	Configura o tamanho da linha para 1919.
Column Size	0x04	0x09FF	Configura o tamanho da coluna para 2559.
Row Address Mode	0x22	0x0011	Habilita o modo de leitura da linha como *binning* combinado com *skipping*.
Column Address Mode	0x23	0x0011	Habilita o modo de leitura da coluna como *binning* combinado com *skipping*.

Estes registradores foram configurados através do arquivo Verilog para auto-inicialização da câmera, como mostra a figura 44.

```
// Internal Assignments
always @(*)
begin
   case (rom_address)
   0     :  data  <= {8'hBA, 8'h00, 16'h0000};
   1     :  data  <= {8'hBA, 8'h20, 16'hc000}; // Mirror Row and Columns
   2     :  data  <= {8'hBA, 8'h09, exposure}; // Exposure
   3     :  data  <= {8'hBA, 8'h05, 16'h0000}; // H_Blanking
   4     :  data  <= {8'hBA, 8'h06, 16'h0019}; // V_Blanking
   5     :  data  <= {8'hBA, 8'h0A, 16'h8000}; // change latch
   6     :  data  <= {8'hBA, 8'h2B, 16'h000b}; // Green 1 Gain
   7     :  data  <= {8'hBA, 8'h2C, 16'h000f}; // Blue Gain
   8     :  data  <= {8'hBA, 8'h2D, 16'h000f}; // Red Gain
   9     :  data  <= {8'hBA, 8'h2E, 16'h000b}; // Green 2 Gain
   10    :  data  <= {8'hBA, 8'h10, 16'h0051}; // set up PLL power on
   11    :  data  <= {8'hBA, 8'h11, 16'h1801}; // PLL_m_Factor<<8+PLL_n_Divider
   12    :  data  <= {8'hBA, 8'h12, 16'h0005}; // PLL_p1_Divider
   13    :  data  <= {8'hBA, 8'h10, 16'h0053}; // set USE PLL
   14    :  data  <= {8'hBA, 8'h98, 16'h0000}; // disble calibration
   15    :  data  <= {8'hBA, 8'hA0, 16'h0000}; // Test pattern control
   16    :  data  <= {8'hBA, 8'hA1, 16'h0000}; // Test green pattern value
   17    :  data  <= {8'hBA, 8'hA2, 16'h0FFF}; // Test red pattern value
   18    :  data  <= {8'hBA, 8'h01, 16'h0036}; // set start row
   19    :  data  <= {8'hBA, 8'h02, 16'h0010}; // set start column
   20    :  data  <= {8'hBA, 8'h03, 16'h077f}; // set row size
   21    :  data  <= {8'hBA, 8'h04, 16'h09ff}; // set column size
   22    :  data  <= {8'hBA, 8'h22, 16'h0011}; // set row mode in bin mode
   23    :  data  <= {8'hBA, 8'h23, 16'h0011}; // set column mode in bin mode
   24    :  data  <= {8'hBA, 8'h49, 16'h01A8}; // row black target
   default  :  data  <= {8'h00, 8'h00, 16'h0000};
   endcase
end
```

Figura 44: Configuração dos registradores da câmera em Verilog. (Fonte: Quartus II)

9 TESTES E RESULTADOS

Para verificar se o sistema atende aos requisitos listados na seção 1.2, foram realizados dois testes que serão descritos a seguir.

9.1 TESTE PRÉ-PROCESSAMENTO

Este experimento foi realizado sem o processamento da imagem, com o intuito de verificar se o sistema está captando corretamente as imagens da câmera e transmitindo-as sem problemas ao monitor VGA. Isto foi feito utilizando duas bolas diferentes:

- Uma bola de cor vermelha intensa, opaca, lisa e com um desenho em preto.
- Outra bola de cor vermelha de pouca intensidade (rosada), também opaca e com saliências em toda sua circunferência.

O ambiente de testes estava relativamente bem iluminado e o fundo das imagens estava claro e uniforme, precisando alterar apenas um pouco o tempo de exposição dos sensores. Consequentemente, a taxa de quadros permaneceu em torno de 28 FPS (*Frames per second*). Como visto nas figuras 45 e 46, o sistema conseguiu recuperar as imagens da câmera e exibi-las no monitor VGA, resultando no cumprimento dos requisitos descritos na tabela 5.

Tabela 5: Requisitos cumpridos pelo teste pré-processamento. (Fonte: Própria)

Requisito	**Nota**
RF1	O sistema obteve a imagem da câmera com êxito.
RF2	O sistema é capaz de alterar a exposição à luz dos sensores da câmera.
RF5	O sistema grava corretamente a imagem captada na memória SDRAM.
RNF1	O sistema atingiu uma taxa de quadros superior a 10 FPS.
RNF5	O ajuste da exposição à luz foi realizada através dos botões *switch*.

Figura 45: Imagem de uma bola vermelha, opaca e lisa antes do processamento de imagem. (Fonte: Própria)

Figura 46: Imagem de uma bola rosada, opaca e com saliências antes do processamento de imagem. (Fonte: Própria)

9.2 TESTE PÓS-PROCESSAMENTO

Este experimento teve como objetivo o teste do componente de processamento de imagens e do algoritmo de segmentação por limiarização. Ele foi realizado com as mesmas condições do teste anterior, sendo utilizadas nas imagens as mesmas bolas.

A taxa de quadros por segundo sofreu uma pequena alteração e passou a ficar na faixa de 25 FPS. Diferença essa não perceptível na visualização da imagem no monitor VGA.

O limiar utilizado pelo algoritmo de segmentação por limiarização foi estabelecido no valor de T = 85, dentre os possíveis limites de 0-255. Chegou-se neste valor por tentativa e erro até que os objetos vermelhos estivessem bem destacados na imagem.

Pode-se verificar na imagem 47, que a bola opaca e lisa foi inteiramente reconhecida. Até mesmo o contorno do desenho pode ser visto nela. Já na imagem 48, verifica-se que a bola rosa foi parcialmente capturada com o limite no valor de T = 85.

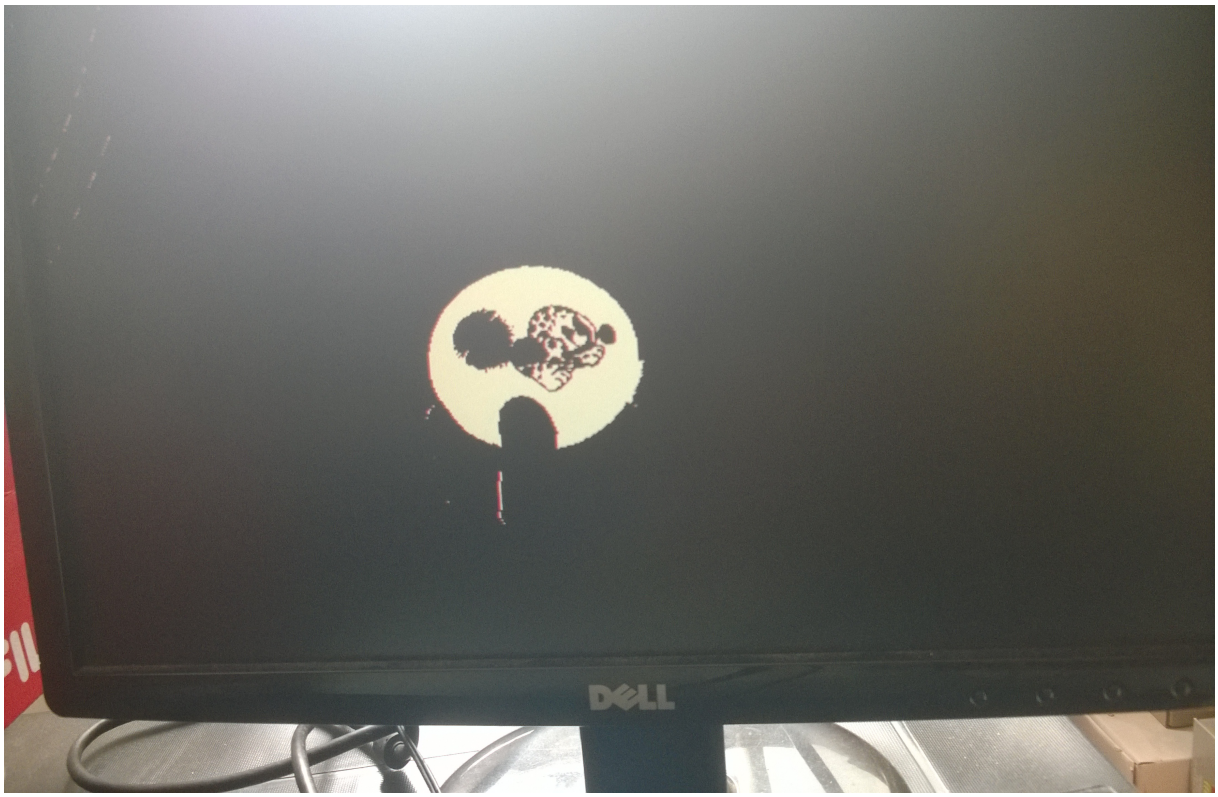

Figura 47: Imagem da bola vermelha depois do processamento de imagem com limiar T = 85. (Fonte: Própria)

Figura 48: Imagem da bola rosada depois do processamento de imagem com limiar T = 85. (Fonte: Própria)

Alterando o valor do limiar para T = 30, a bola rosa é melhor reconhecida. Contudo, há um aumento considerável de ruído na imagem (ver figura 49).

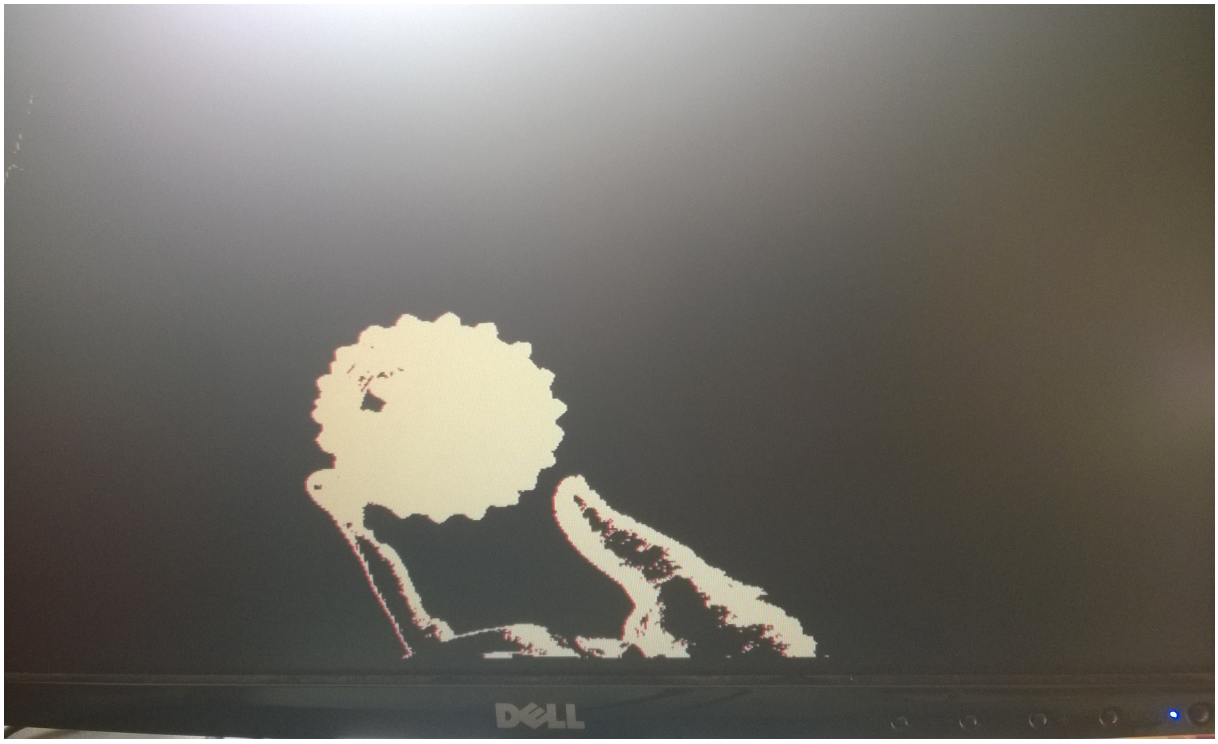

Figura 49: Imagem da bola rosada depois do processamento de imagem com limiar T = 30. (Fonte: Própria)

Com a realização deste teste, os requisitos restantes foram obedecidos. Eles estão descritos na tabela 6.

Tabela 6: Requisitos cumpridos pelo teste pós-processamento. (Fonte: Própria)

Requisito	**Nota**
RF3	O sistema processa a imagem inteiramente por *hardware*.
RF4	O sistema é capaz de reconhecer objetos de cor vermelha na imagem.
RF6	O sistema exibe a imagem processada no monitor VGA.
RNF2	O sistema utiliza a segmentação por limiarização no processamento de imagem.
RNF3	A imagem processada esta binarizada, ou seja, toda em preto e branco.
RNF4	O processamento de imagem foi implementado em VHDL utilizando uma FPGA.

10 CONCLUSÃO

Neste documento foi apresentada a implementação de um sistema embarcado em FPGA dedicado ao reconhecimento de objetos vermelhos, aplicando-se o algoritmo de segmentação por limiarização como processamento de imagem.

Para implementação do sistema, foram utilizados vários componentes disponibilizados pela Altera e também criado o componente responsável pelo processamento das imagens utilizando a ferramenta Qsys. Um dos componentes mais importantes no sistema foi o processador Nios II, pois através dele foi possível a comunicação dos componentes por meio do barramento Avalon, assim como criar um ou mais *softwares* em linguagem C para que ele execute.

Para tornar possível este projeto foi necessário um vasto conhecimento das ferramentas e dispositivos utilizados, além de um embasamento teórico em sistemas embarcados, processamento de imagens, protocolos de comunicação, linguagens de descrição de *hardware* e FPGAs. A maioria destes conhecimentos foram adquiridos no decorrer do curso de Engenharia de Computação. Outros, no entanto, tiveram de ser adquiridos através de extensa leitura de livros, manuais e tutoriais disponíveis na internet, como por exemplo: VHDL avançado, Verilog, Qsys e o barramento Avalon. Além disso, foi necessária a obtenção da plataforma de desenvolvimento Altera DE2 e do módulo de câmera D5M. Como a equipe apenas tinha um dispositivo de cada, foi muito difícil o trabalho conjunto na parte de implementação do *hardware*. Somando-se estes obstáculos, a equipe perdeu muito mais tempo do que o esperado para implementação do projeto.

Ultrapassando as dificuldades encontradas durante a fase de projeto, o sistema cumpre o seu objetivo principal que é o reconhecimento de objetos vermelhos em imagens adquiridas por uma câmera, como discutido no capítulo 9. O processamento foi inteiramente realizado por *hardware*, obtendo-se uma taxa de atualização de quadros relativamente boa. A taxa de atualização de quadros chegou a 28-29 FPS, quando em um ambiente de ótima iluminação e chegou aos 7 FPS, quando necessário aumentar muito o tempo de exposição à luz dos sensores.

O resultado do processamento de imagem para reconhecimento dos objetos vermelhos realizado no teste do capítulo anterior demonstrou que o algoritmo utilizado é muito sensível à iluminação do ambiente e à reflexão da luz pelos objetos, tornando-se fraco para sistemas que exigem exatidão ou reconhecimento de objetos com mais de uma cor e/ou em ambientes em que há muita variação na iluminação. Este é um dos problemas encontrados na classificação de objetos apenas pela cor dos *pixels*. Além disso, há vários outros aspectos que influenciam na qualidade da imagem: distância focal, distorções radiais da lente, resolução da imagem, tempo de exposição, etc.

São várias as contribuições que este trabalho proporciona: um sistema inteiro implementado em uma FPGA, a criação de um componente de processamento de imagem, a implementação de comunicação por fluxo de dados utilizando o barramento Avalon, a interface com o módulo de câmera TRDB-D5M, entre outros. Sendo assim, este projeto pode servir de base no desenvolvimento de sistemas semelhantes e, também, auxiliar no aprendizado dos dispositivos e ferramentas da Altera aplicados ao processamento de imagens.

10.1 TRABALHOS FUTUROS

Seguindo o que foi discutido neste capítulo, uma ideia proposta para trabalhos futuros é a de integrar ao sistema, o processamento de imagem em *software*. Ou seja, implementar exatamente o mesmo algoritmo desenvolvido em *hardware* e fazer a comparação de *performance* entre os dois.

O presente trabalho fornece a base para tal, uma vez que o processador Nios II já está implementado e dispõe de uma memória de programa relativamente grande(512KB). Além disso, a Altera fornece uma vasta coleção de materiais em seu site para auxiliar na criação de sistemas embarcados, desde os mais simples até os mais complexos.

REFERÊNCIAS

ALTERA. **Introduction to Quartus II Software**. 2011. Disponível em: <http://www.altera.com/literature/manual/quartus2-introduction.pdf>. Acesso em: 16 de janeiro de 2015.

ALTERA. **DE2 Development and Education Board**: User manual. [S.l.], 2012. 71 p.

ALTERA. **Introduction to the Altera Nios II Soft Processor**. [S.l.], 2013. 27 p.

ALTERA. **Video IP Cores for Altera DE-Series Boards**. [S.l.], 2013. 46 p.

ALTERA. **About Qsys**. 2014. Disponível em: <http://quartushelp.altera.com/current/mergedProje about-qsys.htm>. Acesso em: 16 de janeiro de 2015.

ALTERA. **Nios II Processor Reference Handbook**. [S.l.], 2014. 288 p.

ALTERA. **Nios II Software Developer's Handbook**. [S.l.], 2014. 498 p.

ALTERA. **Altera FPGAs**. 2015. Disponível em: <http://www.altera.com/devices/fpga/fpga-index.html>. Acesso em: 16 de janeiro de 2015.

ALTERA. **Altera Megafunctions**. 2015. Disponível em: <http://www.altera.com/products/ip/altera/mega.html/design-flow>. Acesso em: 16 de janeiro de 2015.

ALTERA. **Avalon Interface Specifications**. [S.l.], 2015. 58 p.

ALTERA. **Embedded Software Development**. 2015. Disponível em: <http://www.altera.com/devices/processor/nios2/tools/ni2-development-tools.html>. Acesso em: 16 de janeiro de 2015.

ALTERA. **Frequently Asked Questions Embedded Processing**. 2015. Disponível em: <http://www.altera.com/devices/processor/faq/emb-faq.html>. Acesso em: 16 de janeiro de 2015.

CHO, J. U. et al. **A Survey of AI-based meta-heuristics for Dealing with Local Optima in Local Search**. 2007. International Conference on Control, Automation and Systems 2007.

CHU, P. P. **Embedded SOPC Design with Nios II Processor and VHDL Examples**. Hoboken, New Jersey: John Wiley and Sons, 2011.

DOTCOM, V. **Verilog Resources**. 2012. Disponível em: <http://www.verilog.com/>. Acesso em: 17 de Fevereiro de 2015.

ELETRONICS, F. **What is an Embedded Processor?** 2015. Disponível em: <http://www.futureelectronics.com/en/microprocessors/embedded-processors.aspx>. Acesso em: 17 de Fevereiro de 2015.

G.SHAPIRO, L.; STOCKMAN, G. C. **Computer Vision**. 1st. ed. Seatle, Washington: Prentice Hall, 2001.

HEMATIAN, A. et al. **Field Programmable Gate Array System for Real-time IRIS Recognition**. 2011. Advanced Informatics School, Universiti Teknologi Malaysia.

MIKLOS, P. **Image Interpolation Techniques**. 2004. Serbia and Montenegro.

NXP SEMICONDUCTORS. **I²C-bus specification and user manual**. [S.l.], 2014. 64 p.

OSHANA, R. **Introduction to Embedded and Real-Time Systems**. [S.l.]: Texas Instruments, 2006.

PEDRONI, V. A. **Circuit Design and Simulation with VHDL**. [S.l.]: The MIT Press; second edition edition, 2010.

SOURCETECH411. **Top FPGA Companies For 2013**. 2013. Disponível em: <http://sourcetech411.com/2013/04/top-fpga-companies-for-2013>. Acesso em: 16 de janeiro de 2015.

TERASIC. **TRDB-D5M**: Terasic trdb-d5m hardware specification. [S.l.], 2009. 55 p.

TERASIC. **DE Series Introduction**. 2015. Disponível em: <http://www.terasic.com>. Acesso em: 16 de janeiro de 2015.

WIKIPEDIA. **Bayer Filter**. 2015. Disponível em: <http://en.wikipedia.org/wiki/bayer-filter>. Acesso em: 16 de janeiro de 2015.

XILINX. **All Programmable FPGAs**. 2015. Disponível em: <http://www.xilinx.com/products/silicon-devices/fpga.html>. Acesso em: 16 de janeiro de 2015.