UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO DE ELETRÔNICA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MAURO ACRAS

ANÁLISE DE REQUISITOS TEMPORAIS PARA SISTEMAS EMBARCADOS AUTOMOTIVOS

DISSERTAÇÃO

PONTA GROSSA

2016

MAURO ACRAS

ANÁLISE DE REQUISITOS TEMPORAIS PARA SISTEMAS EMBARCADOS AUTOMOTIVOS

Dissertação apresentada como requisito para obtenção do grau de Mestre em Engenharia Elétrica do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná – Campus Ponta Grossa. Área de Concentração: Instrumentação e Controle.

Orientador: Prof. Dr. Max Mauro Dias Santos

PONTA GROSSA

2016

Ficha catalográfica elaborada pelo Departamento de Biblioteca
da Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa
n.13/17

A187 Acras, Mauro

Análise de requisitos temporais para sistemas embarcados automotivos / Mauro Acras. -- 2017.
109 f. : il. ; 30 cm.

Orientador: Prof. Dr. Max Mauro Dias Santos

Dissertação (Mestrado em Engenharia Elétrica) - Programa de Pós-Graduação em Engenharia Elétrica. Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2017.

1. Sistemas embarcados (Computadores). 2. Automóveis - Software. 3. Processamento eletrônico de dados em tempo real. I. Santos, Max Mauro Dias. II. Universidade Tecnológica Federal do Paraná. III. Título.

CDD 621.3

Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FOLHA DE APROVAÇÃO

Título da Dissertação Nº 22/2016

ANÁLISE DE REQUISITOS TEMPORAIS PARA SISTEMAS EMBARCADOS AUTOMOTIVOS

por

Mauro Acras

Esta dissertação foi apresentada às **15 horas e 30 minutos** do dia **14 de dezembro de 2016** como requisito parcial para a obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA, com área de concentração em Controle e Processamento de Energia, Programa de Pós-Graduação em Engenharia Elétrica. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Dr. Angelo Marcelo Tusset – UTFPR-PG

Prof. Dr. Frederic Conrad Janzen – UTFPR-PG

Prof. Dr. José Manoel Balthazar – UNICAMP

Prof. Dr. Max Mauro Dias Santos (UTFPR)
Orientador

Prof. Dr. Claudinor Bitencourt Nascimento (UTFPR)
Coordenador do PPGEE

A FOLHA DE APROVAÇÃO ASSINADA ENCONTRA-SE NO DEPARTAMENTO DE REGISTROS ACADÊMICOS DA UTFPR – CÂMPUS PONTA GROSSA

Dedico este trabalho aos meus pais Jorge e Adélia, meus maiores exemplos e incentivadores, à minha esposa Gilmara e aos meus filhos Isabela e Murilo, pelos quais tudo vale a pena.

AGRADECIMENTOS

Agradeço ao meu pai, exemplo de vida e superação, sempre pronto a ajudar quando necessário. Agradeço à minha mãe que, como professora, desde recém-graduado há quase 25 anos, sempre me aconselhou a continuar os estudos. Demorei mas segui o conselho, obrigado mãe.

À minha esposa e meus filhos que sempre estiveram ao meu lado nos momentos de maior dificuldade.

Ao meu orientador Prof. Dr. Max Mauro Dias Santos, pelo estímulo ao desenvolvimento desta dissertação e fundamentais ensinamentos que contribuíram de forma incomensurável ao longo da pesquisa desenvolvida.

Ao Prof. Ângelo Marcelo Tusset que desde os primeiros dias desta trajetória sempre ofereceu importantes conselhos e palavras de apoio e incentivo, tanto em sala de aula quanto em particular.

À toda a equipe da Gliwa GmbH, em especial ao Sr. Peter Gliwa que não só forneceu os equipamentos e ferramentas necessárias para o desenvolvimento deste trabalho, como ofereceu todo o suporte necessário.

Aos engenheiros Alfredo Baratta e Jerry Park da Infineon Technologies AG, que apesar da distância ofereceram suporte fundamental para a realização do projeto.

À todos os meus amigos e colegas que direta ou indiretamente ajudaram e torceram por mim.

À UTFPR, seus professores e colaboradores que oportunizaram o apoio e estrutura para adquirir os conhecimentos necessários para desenvolver este trabalho.

E a Deus por ter me dado saúde e força para superar as dificuldades.

RESUMO

ACRAS, Mauro. **Análise De Requisitos Temporais Para Sistemas Embarcados Automotivos**. 2016. XX folhas. Dissertação (Mestrado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2016.

Os sistemas embarcados automotivos são caracterizados por sistemas computacionais que suportam funcionalidades na forma de softwares embarcados para proporcionar aos usuários maior conforto, segurança e desempenho. Entretanto, existe uma grande quantidade de funções integradas que elevam o nível de complexidade de forma que se deve utilizar métodos e ferramentas de projetos adequados para garantir os requisitos funcionais e não funcionais do sistema. Todo o projeto de software embarcado automotivo deve iniciar com a definição de requisitos funcionais e de acordo com a dinâmica do subsistema que uma ECU (*Electronic Control Unit*) irá controlar e/ou gerenciar, deve-se ainda definir os requisitos temporais. Uma função automotiva pode ter requisitos temporais do tipo, período de ativação, atraso fim-a-fim, deadline entre outras que por sua vez estão estritamente relacionadas com as características da arquitetura de hardware utilizada. Em um sistema automotivo, tem-se uma arquitetura de computação embarcada distribuída em que existem tarefas e mensagens que trocam sinais entre si e podem ter requisitos temporais que devam ser atendidos. A análise temporal para verificação e validação dos requisitos temporais pode ser realizada ao nível de arquitetura distribuída, tarefas e instruções sendo que a utilização adequada de métodos e ferramentas é uma condição necessária para sua verificação. Desta forma, apresenta-se uma descrição do estado da arte de análise temporal em sistemas embarcados automotivos, suas propriedades e a utilização das ferramentas da Gliwa para avaliar se os requisitos temporais são atendidos. Um exemplo ilustrativo foi implementado com o propósito de apresentar como os métodos, processos e ferramentas devem ser aplicados para verificar se os requisitos temporais definidos previamente no início do projeto foram atendidos e para que em um sistema já existente possam suportar funções adicionais com requisitos temporais a serem garantidos. É importante verificar que as ferramentas de análise temporal, tem o propósito ainda de verificar se os recursos computacionais estão sendo utilizados de acordo com o especificado no início do projeto.

Palavras-chave: Sistemas Embarcados, Software Automotivo, Sistemas de Tempo Real.

ABSTRACT

ACRAS, Mauro. **Timing Analysis of Automotive Embedded Systems**. 2016. XX pages. Dissertação (Mestrado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2016.

Automotive embedded systems are characterized by computer systems that support embedded software functionalities to provide users with greater comfort, security and performance. However, there are a number of integrated functions that raise the level of complexity so that appropriate design methods and tools must be used to guarantee the functional and non-functional requirements of the system. All automotive embedded software design must begin with the definition of functional requirements and according to the dynamics of the subsystem that an ECU (Electronic Control Unit) will control and/or manage, it is necessary to define the time requirements. An automotive function may have time requirements of type, activation period, end-to-end delay and deadline among others which in turn are strictly related to the characteristics of the hardware architecture used. In an automotive system there is a distributed embedded computing architecture in which there are tasks and messages that exchange signals between them and may have timing requirements that must be met. The timing analysis for verification and validation of timing constrains can be carried out at the level of distributed architecture, tasks and instructions, and the proper use of methods and tools is a necessary condition for their verification. In this way, a description of the state of the art of timing analysis in automotive embedded systems, their properties and the use of the tools of Gliwa to evaluate if the timing constrains are met. An illustrative example has been implemented with the purpose of presenting how the methods, processes and tools should be applied to verify that the time requirements defined at the beginning of the project are met and that in an existing system can support additional functions with requirements to be guaranteed. It is important to note that timing analysis tools are still intended to verify that computational resources are being used as specified at the beginning of the project.

Keywords: Embedded Systems, Automotive Software, Real Time Systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ACRÔNIMOS

ATdemo	*Automotive Tools Demo*
AURIX	*Automotive Real-time Integrated next generation architeture*
AVR	*Adaptative Variable-Rate Task*
BCET	*Best-Case Execution Time*
BDC	*Bottom Dead Center*
BSF	*Basic Scheduling Frame*
CAN	*Controller Area Network*
CET	*Core Execution Time*
CKP	*Crankshaft Position Sensor*
CMU	*Clock Management Unit*
CPS	*Cyber Physical System*
CPU	*Central Processing Unit*
DPLL	*Digital Phase Locked Loop Module*
E/E	Eletro-Eletrônico
ECM	*Engine Control Module*
ECU	*Electronic Control Unit*
GPIO	*General Purpose Input/Output*
GTM	*Generic Timer Module*
ICE	*Internal Combustion Engine*
IHM	Interface Homem Máquina
ISR	*Interrupt Service Routine*
LFBGA	*Low-Profile Fine Pitch Ball Grid Array)*
LQFP	*Low-Profile Quad Flat Package*
OT1	*Open Timing Standard*
PMI	Ponto Morto Inferior
PMS	Ponto Morto Superior
RPM	Rotações Por Minuto
RTOS	*Real Time Operating System*
SW-C	*Software Components (Autosar)*
TBU	*Time Base Unit*
TDC	*Top Dead Center*
TIM	*Timer Input Module*
WCET	*Worst-Case Execution Time*

SUMÁRIO

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

Sistemas embarcados estão cada vez mais presentes na vida moderna. Desde uma simples central de alarme até o controle e gerenciamento de diversos sistemas de um avião utilizam sistemas microprocessados desenvolvidos para determinada aplicação específica. Porém, é bastante trivial notar que os requisitos de tempo necessários no projeto de cada um desses sistemas variam muito de acordo com o seu objetivo. Uma falha causada por uma sobrecarga no processador de uma central de alarme tem consequências muito menos drásticas do que uma falha semelhante no sistema que determina a velocidade relativa de um avião comercial através da leitura das pressões dos tubos de Pitot instalados na sua fuselagem.

No caso dos sistemas embarcados automotivos, muitos deles necessitam que sejam atendidos requisitos de tempo específicos, de forma que se uma tarefa precisa ser executada num determinado instante e este momento é perdido, já não há mais sentido em executá-la. São os chamados “Sistemas de Tempo Real”, onde o momento de executar as tarefas e o tempo que essas tarefas levarão para serem executadas são determinados, ou seja, são sistemas determinísticos. Para que tais requisitos possam ser atendidos são utilizados sistemas operacionais de tempo real (RTOS) onde as tarefas mais críticas podem ser priorizadas.

Muito embora os requisitos de tempo sejam alguns dos mais importantes requisitos de segurança a serem observados em projetos de aplicações críticas, ainda não existe um padrão específico que trate deste tema. O padrão ISO-26262, uma adaptação da IEC-61508 específica para o setor automotivo, trata da identificação de perigos funcionais e não funcionais e da demonstração que o software não viola objetivos relevantes de segurança, porém não menciona procedimentos e práticas especificas para atestar a segurança temporal de determinado projeto, sendo assim, os procedimentos normalmente adotados para atestar a segurança temporal no desenvolvimento de um projeto onde o comportamento temporal é crítico é a chamada “pratica corrente”, ou seja, a prática adotada repetidamente pelos principais desenvolvedores na produção de projetos.

1.2 MOTIVAÇÃO

O comportamento de produtos automotivos hoje em dia é frequentemente controlado por sistemas eletrônicos embarcados, e estes sistemas embarcados estão crescendo em número e complexidade com a adição de novas funcionalidades e recursos nos automóveis modernos.

A concepção e desenvolvimento de tais sistemas embarcados automotivos requer uma abordagem estruturada e bem definida de orientações que facilitem este processo. Abordagens baseadas em modelos têm sido amplamente utilizadas na indústria automotiva para gerenciar a modelagem de sistemas complexos. Linguagens de modelagem são a chave para implementar tais conceitos de modelagem, possibilitando o desenvolvimento com maior agilidade e qualidade, no entanto, questões temporais tem sido sempre uma grande preocupação durante a modelagem. O tempo é importante em quase todos os sistemas cyber físicos de tempo real, especialmente em sistemas embarcados automotivos, onde os resultados ao se negligenciar esta questão podem variar desde a simples perda de conforto até a situações que ameaçam a vida. Ainda assim, algumas das linguagens mais bem sucedidas em sistemas de modelagem ainda enfrentam o desafio de lidar com requisitos de tempo. Assim sendo, com a complexidade crescente e necessidade de se adicionar cada vez mais novas funcionalidades e inovações nos automóveis modernos, o estudo de ferramentas de analise temporal torna-se indispensável de forma a facilitar a detecção e correção dos problemas de temporização nos sistemas Cyber Físicos.

1.3 OBJETIVOS

O objetivo do presente trabalho é realizar a analise temporal de um sistema embarcado automotivo com múltiplas funções sendo executadas e gerenciadas por um sistema operacional de tempo real. Através desta analise temporal determinar a confiabilidade do sistema e o impacto que tarefas não periódicas podem causar nesta confiabilidade em altas rotações. Por confiabilidade do sistema entende-se a carga na CPU, carga no barramento, escalonamento das tarefas de forma que sejam

eliminadas ou minimizadas colisões, preempções, falhas de ativação e conflitos de recursos.

São apresentados neste trabalho os procedimentos adotados, bem como os resultados e conclusões obtidos através da analise temporal de uma central de gerenciamento de motor a combustão (ECM) que utiliza um microcontrolador multi-core com três núcleos rodando um sistema operacional de tempo real (RTOS) que gerencia a execução de tarefas periódicas com escalonamento fixo, tarefas periódicas com escalonamento dinâmico e tarefas adaptativas de taxa variável (AVR).

Para a realização do trabalho foi utilizado um kit “Atdemo” fornecido pela Gliwa Embedded Systems. O kit ATdemo (“Automotive Tools demo”) é composto por uma placa de desenvolvimento ShieldBuddy TC275 e uma placa acessória “Gliwa Shield” desenvolvida pela também alemã Gliwa Embedded Systems especialmente para compor o Kit ATdemo.

A placa ShieldBuddy TC275 conta com o microcontrolador infineon AURIX TC275, que é um poderoso microcontrolador que, além de possuir três núcleos de processamento, possui avançados módulos internos desenvolvidos especificamente para o gerenciamento de motores, sendo assim a ECM ideal para o desenvolvimento do presente trabalho.

Outro componente do Kit ATdemo é o sistema operacional de tempo real (RTOS) GliwOS, fornecido com seu código fonte, possibilitando assim a inclusão dos módulos desenvolvidos especificamente para este trabalho.

Para realizar a analise temporal foi utilizado o “Gliwa T1 Suite”, aplicativo desenvolvido pela Gliwa para se comunicar com a unidade de controle eletrônico (ECU) alvo e obter todas as informações necessárias para a analise temporal em tempo real.

1.4 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está dividido em oito capítulos. O primeiro capítulo discorre sobre generalidades a respeito do trabalho, contêm as considerações iniciais, motivação, objetivos e esta breve descrição sobre a organização do mesmo.

Os quatro capítulos seguintes contêm a fundamentação teórica pertinente ao trabalho em uma sequência lógica onde cada capítulo complementa o anterior, ou seja, o capítulo 2 apresenta conceitos de sistemas embarcados, o capitulo 3 discute as restrições temporais às quais certos sistemas embarcados estão sujeitos e os sistemas operacionais de tempo real (RTOS) que possibilitam a organização das tarefas de forma que estas restrições temporais sejam atendidas, o capitulo 4 discorre sobre a analise temporal em sistemas embarcados e mostra que as restrições temporais podem e devem ser mensuradas, comentando sobre padrões, técnicas e ferramentas para realizar tal analise. Finalizando os capítulos referentes à fundamentação teórica, o capitulo 5 apresenta as tarefas angulares adaptativas (AVR) e seu principais conceitos.

Além da fundamentação teórica, os capítulos 2 a 5 também apresentam a ferramenta utilizada no trabalho referente ao conceito apresentado, ou seja, no capitulo 2 foi apresentado o hardware do sistema embarcado utilizado no trabalho, no capitulo 3 o sistema operacional de tempo real GliwOS, no capitulo 4 se apresenta a poderosa ferramenta de analise temporal Gliwa T1 Suite e no capitulo 5 comenta-se sobre a tarefa AVR desenvolvida para este trabalho.

No capitulo 6 apresenta-se a metodologia utilizada para realizar o trabalho. Desde o projeto e execução da plataforma de emulação de sistema de gerenciamento de moto com AVR, até a metodologia para coleta dos resultados medidos.

Os resultados são apresentados no capitulo 7, seguido pelas conclusões no capítulo 8 e as sugestões de trabalhos futuros no capítulo 9.

2 SISTEMAS EMBARCADOS E SISTEMAS CIBER-FÍSICOS

Um sistema embarcado é um sistema de computação que é completamente dedicado a executar uma ou mais tarefas ou operações, sendo normalmente encontrado incorporado a um sistema completo. O uso dos sistemas embarcados está crescendo dia a dia nos mais diversos campos, especialmente nas áreas militares, aeroespacial, automotiva e médica, mas encontramos também sistemas embarcados até mesmo em simples eletrodomésticos. A quantidade de aplicações de tais sistemas ainda está em pleno crescimento, sendo que existe uma enorme quantidade de novos ensaios e experimentos sendo realizados para a sua utilização.

O crescimento da utilização de sistemas embarcados deve-se também à possibilidade de se ter flexibilidade e controle total na fase de projeto, onde se pode definir o tamanho do equipamento desejado, recursos de segurança, requisitos de confiabilidade, medidas de desempenho e até mesmo o custo do produto final. Exemplos de tais sistemas podem variar de um simples leitor de MP3 a um sistema maior, como os diversos módulos do sistema de controle de uma usina de energia nuclear.

Sistemas Ciber-Físicos (CPS) são sistemas que promovem a integração entre um sistema computacional e o ambiente físico. São sistemas que se valem de sensores para obter informações do ambiente externo, processam estas informações e, com base nos resultados obtidos, acionam atuadores externos de modo obter o resultado físico desejado. Diversos sistemas anteriormente denominados apenas por “Sistemas Embarcados” se enquadram na categoria de sistema Ciber-Físico, sendo este o caso da Central de Gerenciamento do Motor (ECM), pois ela realiza a aquisição de informações das condições do ambiente interno e externo ao motor através dos diversos sensores disponíveis, processa estas informações e determina o acionamento dos atuadores que controlaram os fenômenos físicos necessários para o funcionamento adequado do motor.

2.1 O KIT ATdemo

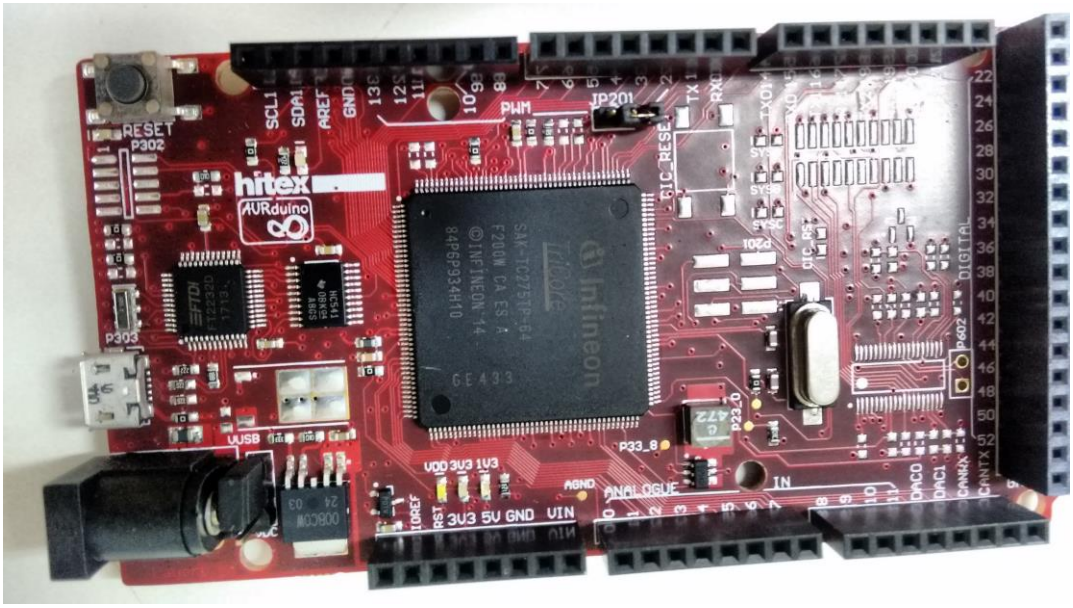
O sistema embarcado utilizado neste trabalho é o kit ATdemo fornecido pela Gliwa Embedded Systems. O kit ATdemo (“Automotive Tools demo”) é uma

plataforma de avaliação concebida para demonstrar a utilização de diversas ferramentas como por exemplo, o compilador, o debbuger , o RTOS e as ferramentas de analise temporal. O kit é composto por uma placa de desenvolvimento ShieldBuddy TC275 fabricada pela empresa alemã hitex e uma placa acessória “Gliwa Shield” desenvolvida pela também alemã Gliwa Embedded Systems especialmente para compor o Kit ATdemo.

2.1.1 HITEX ShieldBuddy TC275

A placa ShieldBuddy TC275 é uma placa com formato e conexões similares à uma placa arduino convencional, porém com o poderoso microcontrolador infineon AURIX TC275.

Figura 1 – Shield Buddy TC275



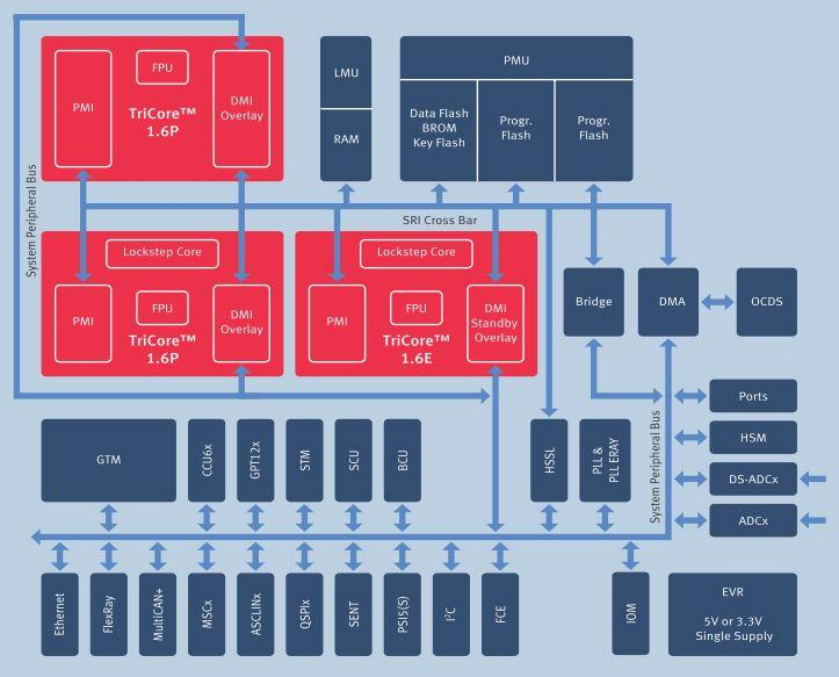
Fonte: Elaborado pelo Autor(2016)

2.1.1.1 Microcontrolador Infineon Aurix TC275

O cérebro da placa de desenvolvimento Shield Buddy TC275 é o poderoso microcontrolador Infineon TC275 da família AURIX (AUtomotive Real-time Integrated

neXt generation architeture). A família de microcontroladores AURIX foi especialmente projetada para aplicações automotivas e caracteriza-se por contar com avançados módulos internos desenvolvidos especificamente para o gerenciamento de motores, como por exemplo, o módulo GTM (Generic Timer Module) que é especialmente útil para reduzir a carga do processador com atividades pertinentes ao gerenciamento de um motor e que são altamente consumidoras de recursos. O TC275 conta com três núcleos de 32 bits rodando à 200MHz.

Figura 2 - arquitetura interna do microcontrolador TC275

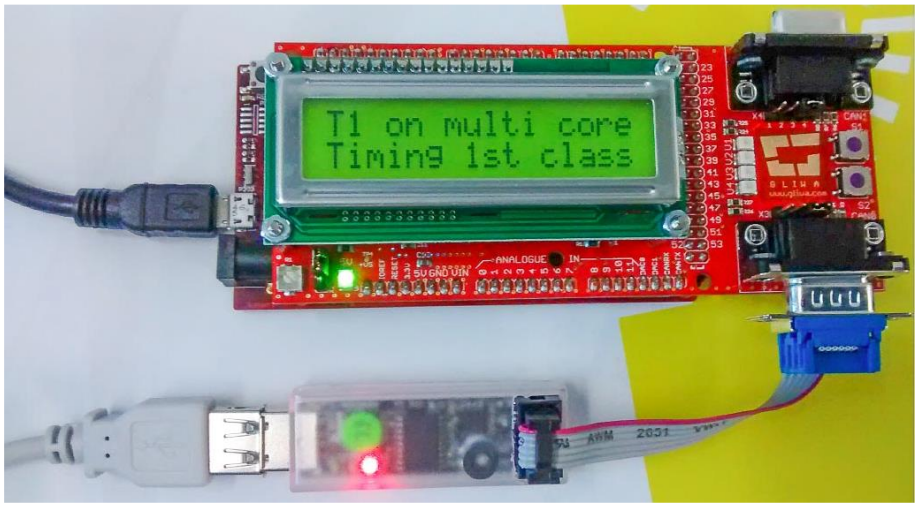


Fonte: Infineon(2016)

2.1.2 Gliwa Shield

Juntamente com a placa ShieldBuddy TC275, faz parte do kit ATdemo, o Gliwa Shield que oferece duas interfaces CAN, quatro leds e dois botões, além de um display com duas linhas de dezesseis caracteres. Entre as principais funções do *shield* estão a conexão entre o Kit e o computador rodando o software T1 que exige uma interface CAN para ser executado e a possibilidade de interação entre o usuário e a placa principal ShieldBuddy através dos botões e do display, tornando-se assim uma interface homem-máquina básica.

Figura 3 – Gliwa Shield



Fonte: Elaborado pelo Autor(2016)

3 RESTRIÇÕES TEMPORAIS EM SISTEMAS CIBER-FÍSICOS E O SISTEMA OPERACIONAL DE TEMPO REAL (RTOS)

3.1 RESTRIÇÕES TEMPORAIS

Sistemas embarcados são na sua maioria associados a restrições de tempo. Um exemplo muito simples das restrições temporais enfrentadas por um sistema embarcado é a *deadline*, ou seja, um período máximo de tempo desejado no qual é necessário que determinada tarefa do sistema seja concluída. Muitas outras restrições de tempo podem surgir durante a execução de uma tarefa ou até mesmo serem associadas a outras tarefas. Devido à estreita associação com o tempo, esses sistemas são conhecidos como sistemas embarcados em tempo real. Para um sistema embarcado em tempo real, não apenas a confiabilidade e o desempenho do sistema são importantes, mas também a conclusão das tarefas dentro dos requisitos de tempo definidos em projeto. Esses sistemas de tempo real podem ser classificados em duas categorias principais com base na importância da satisfação de requisitos de tempo na execução de tarefas. São os sistemas de tempo real rígidos ou críticos (*Hard*) e os sistemas de tempo real moderados ou não-críticos (*Soft*). Para um sistema crítico o controle temporal é o fator mais importante uma vez que as consequências da perda de um *deadline* poderiam ser desde prejuízos financeiros até mesmo danos físicos com risco a vida, enquanto que no caso de sistemas em tempo real moderados as consequências da perda do prazo de execução de uma tarefa não é tão crítica em comparação um sistema de tempo real rígido. Um sistema de controle de voo é um exemplo de sistema de tempo real rígido, uma vez que um único erro de voo pode causar a perda de vidas, por outro lado um sistema de reservas pode ser citado como um exemplo de um sistema de tempo real não-crítico, já que perder uma reserva aérea raramente é catastrófico.

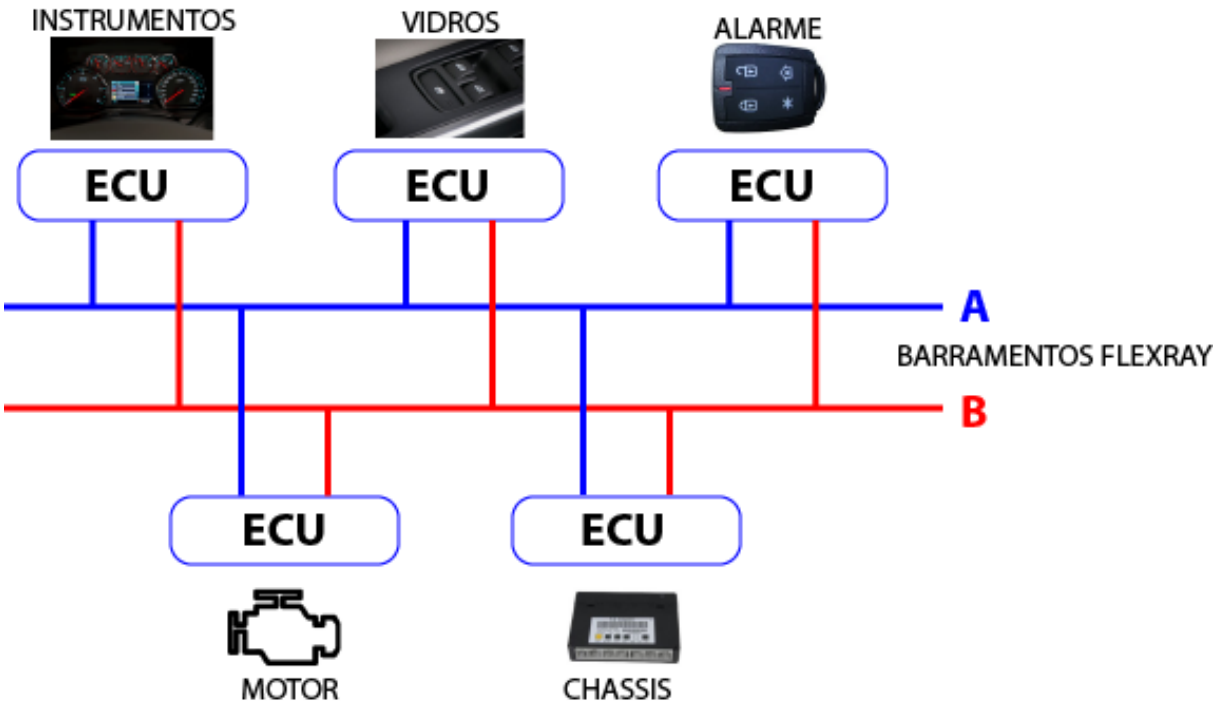
Sistemas Embarcados tem encontrado grande importância em aplicações automotivas. Um automóvel moderno tem diversos sistemas embarcados distribuídos no seu interior. Sistemas de freio ABS, de navegação, de controle de emissões, de ar condicionado, vidros, travas, entre outros, são alguns dos exemplos de tais sistemas cada vez mais presentes nos automóveis atuais. A grande maioria destes sistemas embarcados automotivos são sistemas embarcados em tempo real rígidos.

Um olhar atento sobre estes sistemas revela que cada um deles é uma combinação de sensores, atuadores e Unidade de Controle Eletrônico (ECU). Sensores adquirem os sinais elétricos ou mecânicos necessários e os enviam para a respectiva ECU. A ECU processa estes sinais e envia os sinais necessários para os atuadores realizarem as ações necessárias. A comunicação entre as ECU's é viabilizada por redes de comunicação de padrões específicos para ambientes automotivos. Dentre os principais tecnologias de rede de comunicação automotiva destacam-se os padrões CAN, LIN e FlexRay.

Os sistemas embarcados automotivos são realmente bastante diversificados em termos de funcionalidade, desenvolvimento e construção. Alguns desses sistemas funcionam de forma independente, outros dependem em maior ou menor nível de um ou vários sistemas. Cada um destes sistemas é dedicado a diferentes funcionalidades. Por exemplo, o sistema de controle de travamento das portas recebe e processa as informações dos sensores pertinentes e atua sobre os mecanismos de travamento das portas, um Módulo de Gerenciamento do Motor (ECM) fará o mesmo para controlar o processo de combustão interna no motor e assim por diante. Estes sistemas estão todos distribuídos no interior de um automóvel. Existem várias justificativas para tal distribuição, desde a instalação de cada módulo em um local mais próximo dos eventos que ele irá controlar, até possibilitar que o processo de desenvolvimento dos módulos seja realizado por equipes diferentes e até mesmo em locais diferentes e por empresas diferentes.

A Figura 4 é um exemplo da presença de tais sistemas dentro de um automóvel. O sistema é formado por uma combinação de várias ECU's diferentes, com diferentes funções, conectadas entre si por uma rede de comunicação FlexRay. Estes sistemas automotivos são, portanto, conhecidos como sistemas embarcados distribuídos de tempo real.

Figura 4 – Sistemas embarcados automotivos distribuídos e conectados entre si através de uma rede FlexRay.



Fonte: Elaborado pelo Autor(2016)

Nestes sistemas embarcados distribuídos de tempo real o tempo torna-se um dos fatores mais importantes a se considerar. Ao se modelar tais sistemas é necessária especial atenção às restrições temporais. Embora existam vários padrões de modelagem em prática atualmente, a manipulação das restrições temporais ainda é uma área a ser aprimorada na indústria automotiva.

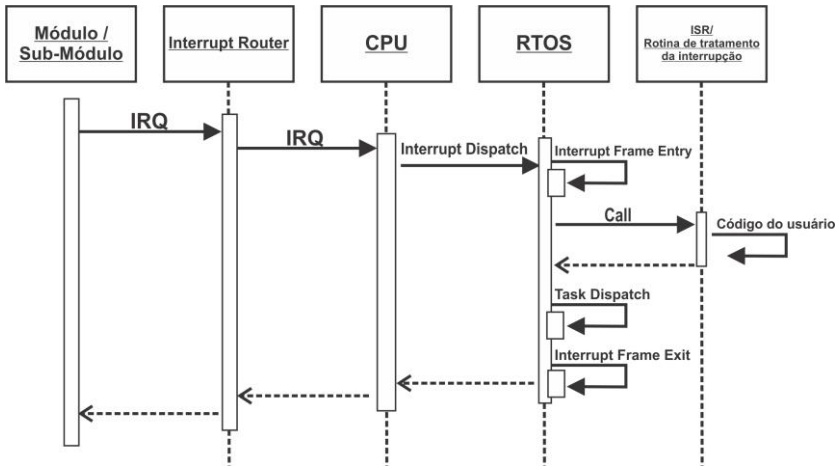
3.2 O SISTEMA OPERACIONAL DE TEMPO REAL (RTOS)

Um sistema operacional de tempo real (RTOS) é um sistema operacional destinado a garantir a execução de processos dentro de parâmetros específicos de tempo. Estes parâmetros específicos de tempo são os chamados “requisitos temporais”. Quando do projeto de um sistema embarcado de tempo real os projetistas determinam diversos requisitos temporais que deverão ser respeitados de forma a garantir a funcionalidade e segurança do sistema quando da execução destes processos.

3.2.1 O Tratamento de Interrupções em um RTOS

Quanto ao tratamento de Interrupções, os sistemas operacionais de tempo real são divididos em duas categorias. Os sistemas operacionais CAT 2 (Figura 5) tem recursos para receber e tratar solicitações de interrupção.

Figura 5 - Tratamento de interrupção em um RTOS CAT2

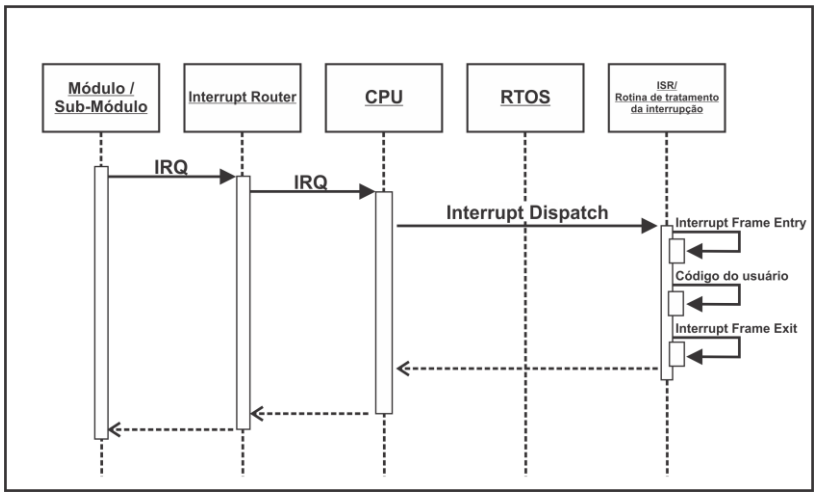


Fonte: AUTOSAR(2011): Adaptado pelo Autor

As solicitações de interrupção são recebidas pelo RTOS CAT2 (Figura 5) que pode ativar tarefas e executáveis para o tratamento da mesma.

Já um RTOS CAT1 (Figura 6) não tem acesso às interrupções do sistema e, portanto, as mesmas são tratadas diretamente pela CPU, sem a intervenção do sistema operacional.

Figura 6 - Tratamento de interrupção em um RTOS CAT1



Fonte: AUTOSAR(2011): Adaptado pelo Autor

Esta característica, por um lado torna o sistema operacional mais leve, mas por outro lado reduz a possibilidade de utilização de rotinas mais elaboradas que

poderiam ser utilizadas em um sistema AVR para ativar seletivamente determinadas tarefas ou executáveis em função do modo AVR atual por exemplo.

3.2.2 GliwOS

O sistema operacional GliwOS é um sistema operacional CAT1 simples e eficiente. O GliwOS suporta microcontroladores multicore, possui múltiplas opções de escalonamento e é fornecido com seu código fonte, o que se mostrou essencial para o desenvolvimento deste trabalho.

4 ANALISE TEMPORAL EM SISTEMAS CIBER-FÍSICOS AUTOMOTIVOS

A análise temporal ajuda a planejar, entender, aperfeiçoar e garantir sistemas embarcados em relação ao seu escalonamento e é considerado um passo essencial durante o desenvolvimento de sistemas embarcados automotivos.

A análise temporal pode ser realizada em diferentes níveis. Ao nível da rede, o objetivo é verificar a troca de informações entre ECU's, pois existem várias ECU's integradas e conectadas por diversas redes em uma plataforma E/E de um veículo. Ao nível RTOS, o objetivo é verificar os efeitos do escalonamento de tarefas em uma ECU. No nível de código, o objetivo é analisar apenas fragmentos de código, por exemplo, determinar o tempo de execução de uma função que executa certa parte do código.

A análise temporal pode também ser realizada em diferentes estágios de desenvolvimento do sistema. Deve-se determinar qual análise será executada em determinada fase do processo de desenvolvimento, uma vez que não é possível aplicar todos os níveis de analise temporal em todos os estágios de desenvolvimento.

4.1 ANALISE TEMPORAL: NÍVEIS E FASES DE DESENVOLVIMENTO

Qualquer atividade, problema ou caso de uso relacionado a tempo pode ser inserido em um diagrama bidimensional contendo em uma dimensão o nível e em outra dimensão a fase de desenvolvimento conforme descrito na Figura 7.

4.1.1 Análise ao nível de código, tarefas e rede

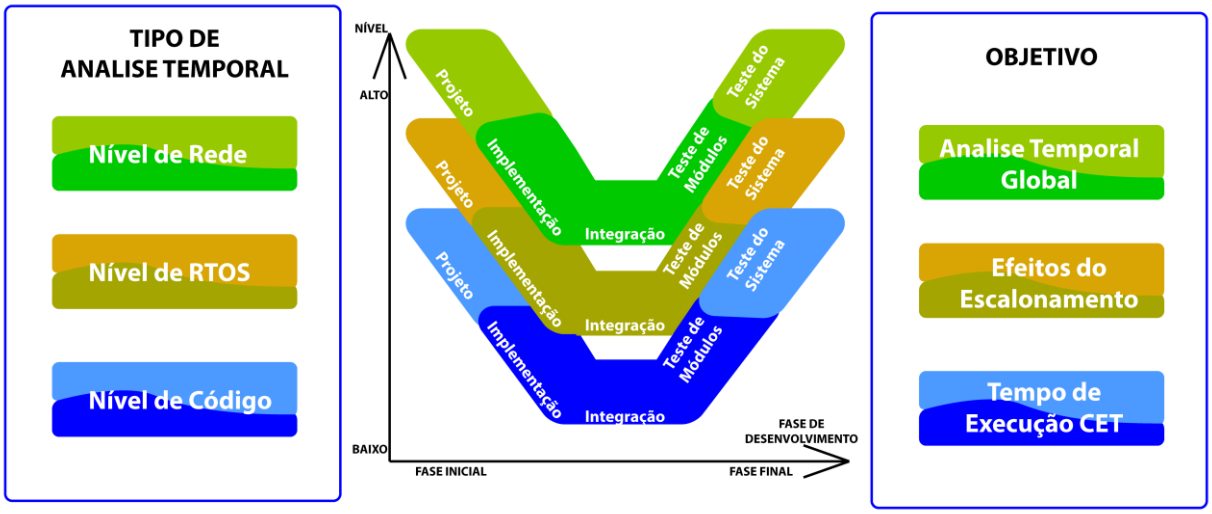
Ao nível de código foca em um fragmento de código (por exemplo, uma única função) independentemente de escalonamento. O tempo de execução é o resultado mais importante do nível de código.

O nível RTOS (sistema operacional de tempo real) considera apenas uma entidade de escalonamento (uma ECU com um processador de núcleo simples, por exemplo) e foca nos efeitos do escalonamento. A maioria dos peritos em temporização no nível do RTOS é encontrada nos fornecedores tier-1 (que fornecem materiais para

os integradores), os fornecedores tier-1 geralmente integram todos os componentes de software à ECU e também configuram o sistema operacional.

Ao nível de rede trabalha com aspectos referentes à comunicação entre ECU's. A maioria dos peritos em temporização no nível de rede são encontrados nos integradores, eles integram várias ECU's conectadas a várias redes em uma plataforma E/E (Elétrica/Eletrônica) para o veículo.

Figura 7 – Tipos de analise temporal para cada fase de desenvolvimento no modelo V



Fonte: Elaborado pelo Autor(2016)

Casos de uso são bem diferentes dependendo de onde eles estão localizados no processo de desenvolvimento. Na fase inicial do projeto por exemplo, a maioria do código fonte ainda não está disponível, tornando impossível medir, traçar ou realizar analises de código estático. As principais tarefas para a analise temporal nas várias fases são:

- **Fase inicial** - determinar requisitos de tempo e projetar um diagrama de tempos que atenda os requisitos, definir o hardware apropriado (selecionar o microcontrolador, por exemplo), iniciar a implementação.
- **Fase de integração** - finalizar a implementação, integrar os componentes em um ambiente operacional, depurar e otimizar a temporização, medir os tempos e relacionar os resultados com os respectivos requisitos, validar os modelos.
- **Fase final** - medir e supervisionar os tempos, realizar testes de temporização (pode ser feito em paralelo com testes funcionais), usar abordagens

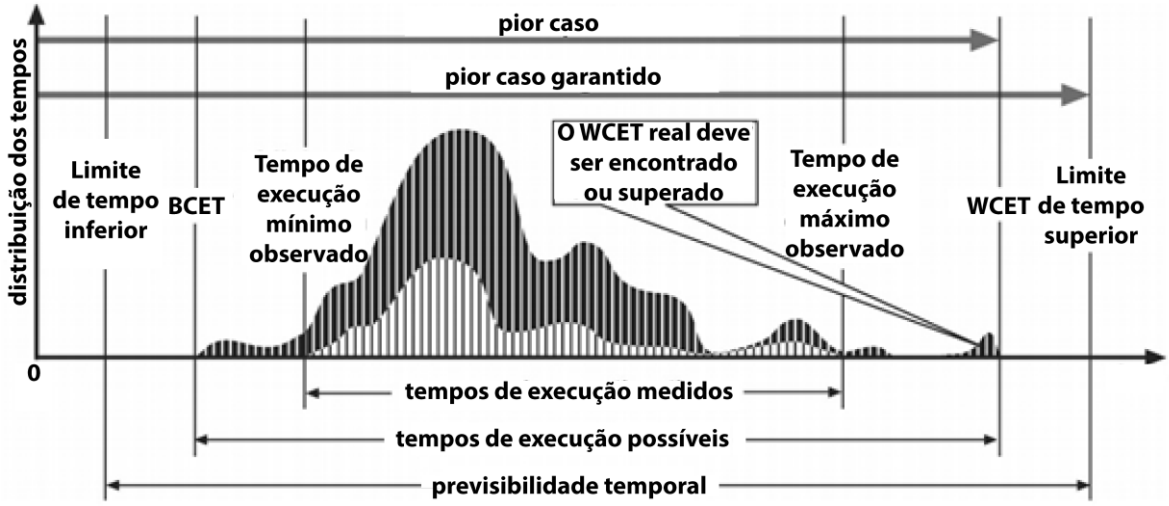
baseadas em modelo para localizar casos não previstos (corner cases) e realizar a verificação formal.

4.2 TÉCNICAS DE ANÁLISE DE TEMPO

Uma maneira de se realizar a análise temporal de um sistema embarcado pode ser utilizando-se de ferramentas de análise temporal já existentes.

O tempo de execução de um determinado código pode variar de acordo com diversos fatores, por exemplo, a concorrência por recursos de rede e processamento. O pior tempo de execução possível é denominado WCET (Worst-Case Execution Time) e o melhor tempo de execução possível é denominado BCET (Best-Case Execution Time). Para a determinação destes valores devem ser consideradas todas as entradas possíveis, inclusive entradas que possam violar as especificações de projeto.

Figura 8 – Diagrama dos tempos de execução de uma tarefa



Fonte: Wilhelm(2008): Adaptado pelo Autor

O diagrama dos tempos de execução possíveis para determinada tarefa (Figura 8) mostra que na maioria das vezes que um mesmo código for executado o seu tempo de execução CET sofrerá pouca variação, porém, algumas poucas vezes o código poderá ser executado em um tempo menor que o normal (sendo o menor tempo o BCET) e algumas vezes poderá ser executado em um tempo maior do que o

normal (sendo o maior tempo o WCET), o que determinará tais situações será a disponibilidade da CPU e dos recursos de hardware.

Para a realização da analise temporal pode-se dispor de diversas técnicas (Tabela 1), cada uma mais apropriada para determinadas fases do desenvolvimento do sistema e para o nível de analise temporal desejada.

Tabela 1- Técnicas de analise temporal

TÉCNICAS DE ANÁLISE TEMPORAL	ENTRADA (DADOS)	ENTRADA (MODELO) OU MECANISMO
Analise de código estático	Código fonte e/ou binário	Modelo do processador
Simulação de código	Binário	Modelo do processador
Rastreamento (*tracing*)/Medição	Software instrumentado ou hardware adaptado	Eventos são gravados numa memória de rastreamento
Simulação de escalonamento	CETs, modelo das aplicações (configuração do escalonador)	Modelo do escalonador
Analise de escalonamento	BCET/WCET, modelo das aplicações (configuração do escalonador)	Modelo do escalonador

Fonte: Gliwa(2016)

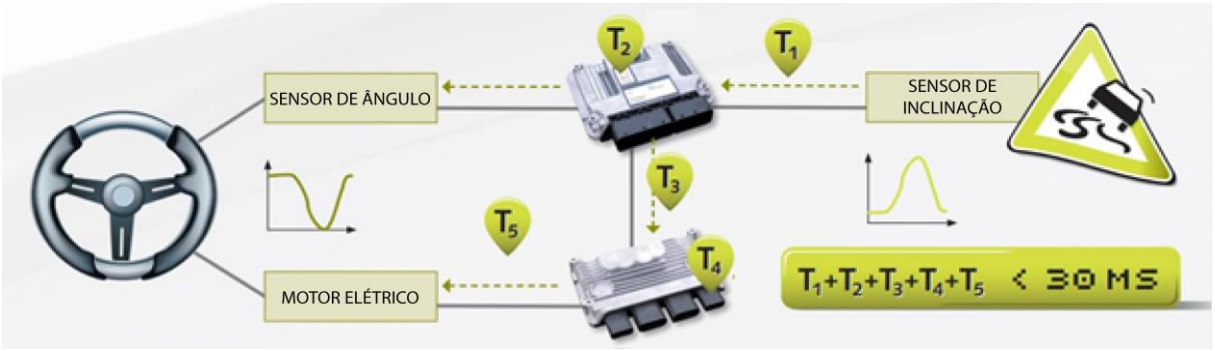
- Análise do Código Estático - As ferramentas de analise do código estático leem o código fonte e/ou o código binário de uma aplicação ou parte dela. Elas calculam um limite inferior para o BCET e um limite superior para o WCET para um dado fragmento de código, uma função por exemplo. Qualquer tempo de execução real possível estará certamente entre este intervalo, desde que este fragmento não seja interrompido. Quaisquer dados presentes somente em tempo de execução (por exemplo, limites superiores de iterações de loops e o conteúdo de ponteiros de funções dinâmicas) devem ser informados manualmente na forma de anotações adicionais.
- Simulação de Códigos - Simuladores de códigos simulam a execução de um determinado código binário para um certo processador. Existe uma vasta variedade de simuladores de códigos. Simuladores com conjuntos de instruções simples fornecem informações muito limitadas sobre o tempo de execução enquanto simuladores complexos consideram também os efeitos de pipeline e cache. Para obter-se valores confiáveis de WCET utilizando um simulador de código, este deve

estar embarcado em um ambiente de teste que realmente cause o pior caso a ser simulado.

- Análise de Escalonamento - Baseado no modelo de um certo escalonador (por exemplo, um determinado sistema operacional de tempo real RTOS), ferramentas de analise de escalonamento tomam CET's mínimos e máximos como entrada e fornecem, por exemplo, o WCET garantido. Isto possibilita verificar se algum limite de tempo será violado dentro de certas condições. Para o pior caso de cada tarefa e interrupção, um diagrama é gerado possibilitando analisar a situação na qual ele ocorre. Os tempos de execução fornecidos para analise podem ser estimativas ou saídas de outras ferramentas, como por exemplo, BCET e WCET analisados estatisticamente ou dados medidos.
- Medição - O sistema real é analisado e a informação de tempo observada é fornecida. A medição de tempo é frequentemente baseada em rotinas hook (interceptam funções antes de serem executadas) do sistema operacional.
- Rastreamento - O método do rastreamento observa o sistema real. Para eventos dedicados, registros de tempo juntamente com informações de evento são colocados em um buffer de rastreamento. A seleção de eventos pode ser tão detalhada a ponto de rastrear fluxos que permitem reconstruir a execução de cada instrução de máquina ou tão superficial a ponto de rastrear somente eventos relacionados ao escalonamento. O rastreamento pode ser baseado em instrumentação (por exemplo, modificação de software) ou em equipamentos especiais de rastreamento. Rastreamentos podem ser visualizados e analisados off-line, para depuração por exemplo. Todos os tipos de informação de tempo podem ser retirados de um rastreamento.
- Simulação de Escalonamento - Simuladores de escalonamento proporcionam funcionalidade similar à analise de escalonamento, porém, ao invés de calcular os resultados eles simulam comportamento de tempo de execução. As informações temporais observadas e os rastreamentos gerados são as principais saídas. Se os cenários de piores casos são simulados, os tempos de resposta observados serão iguais aos WCET's. Alguns simuladores permitem definições de tarefas em linguagem C de forma que modelos de aplicações complexas são suportados ao oferecer uma linguagem de especificação bem conhecida pelos engenheiros automotivos.

4.3 EXEMPLO – SISTEMA DE DIREÇÃO ATIVA

Figura 9 - Diagrama analise temporal



Fonte: Gliwa(2016): Adaptado pelo Autor

Com o modelo dinâmico do carro e a função da direção ativa em mente, o programador definiu um tempo mínimo de reação para a cadeia completa, 30ms no caso. Este se torna um requisito de tempo global de alto nível (nível de rede) para o sistema. Este requisito de tempo então é decomposto, ou seja, é segmentado em porções menores T1, T2...,T5, uma porção para cada componente do sistema. Obviamente, ECU's e barramentos manipulam simultaneamente muito mais elementos com muitos outros requisitos de tempo, todos competindo por recursos computacionais e de rede. Em uma ECU com tarefas, interrupções e seus executáveis, os requisitos de tempo de alto nível são divididos em requisitos mais fragmentados e a competição por recursos continua em um nível mais baixo.

4.4 PADRÕES

4.4.1 Extensões de temporização autosar

Com AUTOSAR V4.0 as extensões de temporização foram introduzidas permitindo a definição de requisitos de tempo. Em um primeiro momento, eventos como "inicio de executável R" ou "recepção de dados D" são definidos. Em um segundo momento, requisitos relacionados aos eventos são formulados.

Exemplo 1: Após o inicio do executável A, a recepção de dados D deve ocorrer até no máximo 2,5ms.

Exemplo 2: Eventos E1, E2, E4 e E7 devem sempre ocorrer nesta sequencia.

4.4.2 Padrão temporal aberto - OT1

Desde o início de 2013 não existe nenhum padrão para troca de informações temporais entre ferramentas de analise temporal. *Open Timing Standard* (OT1) é um formato de troca de dados novo e unificado que se propõe a ser utilizado por todos os tipos de ferramentas relacionadas a temporização. OT1 está no formato XML e permite a troca de:

-Configurações de sistema (tarefas, prioridades, executáveis, etc.)

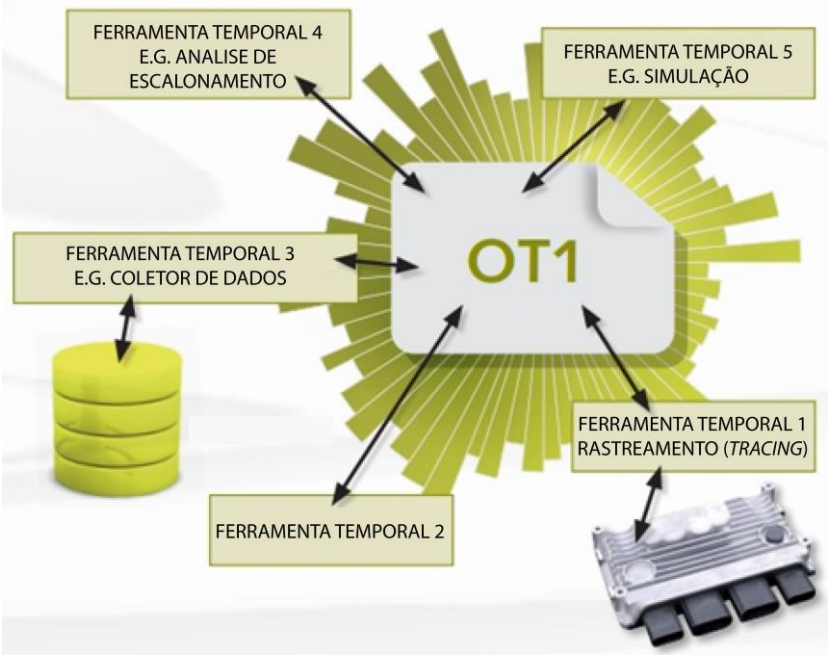
-Rastreamentos (registros de eventos relacionados a escalonamento por exemplo)

-Informações de temporização (tempo de execução, tempos de resposta, etc), também conhecidos como “Garantias de temporização”.

-Requisitos de tempo (tempos de resposta máximos permitidos, por exemplo)

Todas as informações de tempo relacionadas a um projeto são mantidas em um grande container OT1 e qualquer ferramenta de analise temporal pode fornecer ou recuperar informação. Inclusive é possível solicitar informação ausente. Uma ferramenta de analise de escalonamento, por exemplo, pode requisitar o CET de determinado executável e esta solicitação pode ser respondida tanto por uma ferramenta de rastreamento como por uma ferramenta de analise de código. Uma vez que toda informação é identificada com a sua fonte, gerenciar diversas fontes para o mesmo tipo de informação se torna muito fácil.

Figura 10 - Padrão OT1

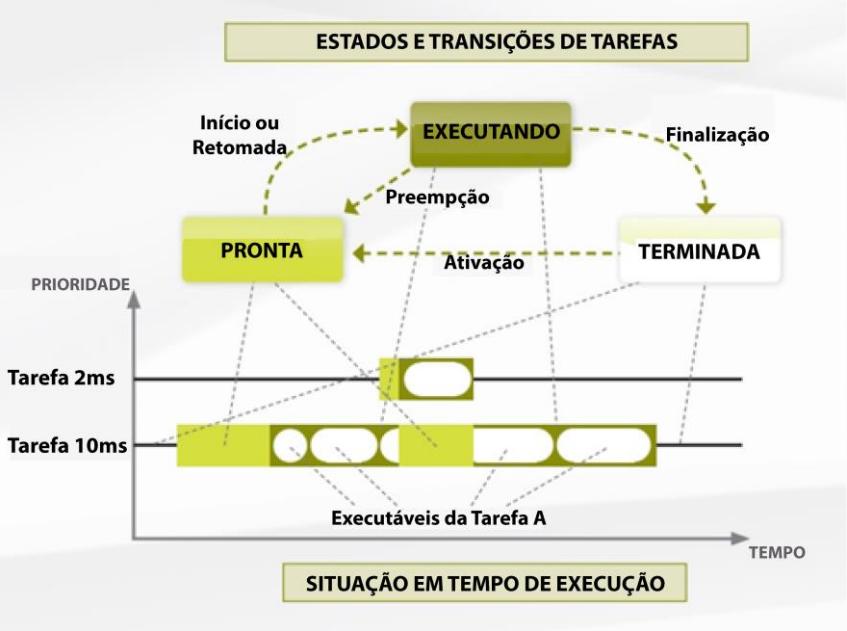


Fonte: Gliwa(2016): Adaptado pelo Autor

4.5 DE COMPONENTES DE SOFTWARE A EXECUTÁVEIS, A TAREFAS E A EXECUÇÃO

Componentes de software AUTOSAR (SWC) encapsulam uma determinada funcionalidade, por exemplo, o controle da marcha lenta de uma ECU de gerenciamento do motor. Uma SWC é implementada com executáveis que tem certos requisitos de escalonamento, segurança e temporização. O controle de marcha lenta por exemplo pode ser codificado em três executáveis: IdleSpeedInit, IdleSpeed10ms e IdleSpeed50ms. Como parte da configuração do sistema operacional, todos os executáveis são mapeados a tarefas ou interrupções que atendem seus requisitos. Em uma situação em tempo de execução mostrada na figura abaixo (Figura 11), Task10ms contém quatro executáveis, portanto o executável IdleSpeed10ms tem que compartilhar o seu container com três executáveis de outras SWC's. Em tempo de execução o sistema operacional escalona tarefas e interrupções de acordo com seus atributos (mais importante: período e prioridade).

Figura 11 - Estados e transições de tarefas



Fonte: Gliwa(2016): Adaptado pelo Autor

4.6 CARGA DA CPU – CARGA DO BARRAMENTO

A carga na CPU e a carga no barramento são as características mais importantes na analise temporal. Elas unem o complexo tema da analise temporal em um único numero que é perfeito para relatórios de gerenciamento, entretanto, eles são muito simples para capturar todas as características de temporização. Por exemplo, uma ECU com carga de 40% pode facilmente violar exigências de tempo.

Quando declaramos a carga da CPU e do barramento, devemos tornar claro como elas foram definidas:

-Qual é o frame de referencia?

-Como foi considerada a tarefa de background (se existir)?

-O overhead do sistema operacional foi considerado corretamente?

Acima de tudo, a carga da CPU / carga do barramento não pode substituir um conjunto completo de requisitos de tempo detalhados.

4.7 SEGURANÇA E DISPONIBILIDADE

A segurança e disponibilidade de um sistema são requisitos concorrentes. Um sistema pode entrar em um estado seguro contra falhas abandonando a disponibilidade. Seria lógico concluir que o sistema mais seguro é o que simplesmente não faz nada, entretanto, tal sistema não será comercialmente bem sucedido.

Uma falha em atender requisitos rígidos de tempo fará um sistema seguro e bem protegido entrar em um estado livre de falhas, atingindo seus requisitos de segurança, mas oferecendo pouca ou nenhuma funcionalidade. Uma tarefa que se exceda levemente pode fazer com que todas as funções da sua ECU sejam descartadas. No contexto de sistemas seguros, bom comportamento temporal é portanto essencial para se manter a disponibilidade.

Durante o processo de desenvolvimento de tal sistema seguro, erros de temporização podem ser muito difíceis de analisar e depurar, uma vez que a proteção de temporização assume o controle e para as operações. Somente com ferramentas adequadas podemos analisar e reconstruir o comportamento temporal antes de tais paradas.

4.8 PADRÕES ABORDANDO ASPECTOS DE SEGURANÇA

Requisitos Funcionais: São aqueles relacionados às entradas e saídas desejadas do sistema, ou seja, é o comportamento desejado nas saídas do sistema quando submetido à determinadas entradas. São as funções para as quais o sistema está sendo desenvolvido.

Requisitos Não Funcionais: São aqueles relativos à aspectos internos do sistema, ou seja, não são relativos às funções desejadas no sistema, mas devem ser atendidos da mesma forma.

Não existe nenhum padrão de segurança especifico para temporização embarcada, entretanto, os padrões listados abaixo exigem a identificação de perigos funcionais e não funcionais e a demonstração de que o software não viola objetivos relevantes de segurança. Estes padrões mencionam explicitamente três importantes características de software não funcionais e relevantes para a segurança:

- Ausência de erros em tempo de execução
- Tempo de execução

- Consumo de memória

Tabela 2 - Padrões de segurança

PADRÃO	NÍVEL DE SEGURANÇA		COMENTÁRIOS
	Mais Baixo	Mais Alto	
IEC-61508	SIL1	SIL4	Padrão geral (não especificamente automotivo) para segurança funcional já obsoleto
IEC-61508 Edição 2.0	SIL1	SIL4	Sucessor do IEC-61508
ISO-26262	ASIL-A	ASIL-D	Adaptação do IEC-61508 especificamente automotiva
DO-178B	Level E	Level A	Padrão de Segurança para Aviônica
DO-178C	Level E	Level A	Sucessor do DO-178B
CENELEC prEN 50128	SIL1	SIL4	Padrão de Segurança Ferroviario

Fonte: Gliwa(2016)

4.9 ASPECTOS LEGAIS

No pior caso, pessoas podem morrer como resultado de um problema de temporização. Não está claramente definido como tais casos são tratados em um tribunal, mas o fabricante da ECU que causou o acidente será questionado se o software foi testado e verificado de acordo com a prática corrente. A prática corrente é definida por pesquisa e se torna prática corrente quando aplicada repetidamente na produção de projetos, portanto, no que diz respeito a responsabilidade, projetos deveriam pelo menos fazer uso da prática corrente.

4.10 ANALISE TEMPORAL EM PROCESSADORES *MULTICORE*

Desenvolvimento multi núcleos apresenta desafios significativamente maiores. A lista abaixo destaca os aspectos principais:

Nível de código

-Conflitos de memória/barramento compartilhados. (por exemplo, núcleos buscando código da mesma flash)

-CET’s diferentes para a mesma função em diferentes núcleos. (isto se torna relevante quando utilizando alocação dinâmica de tarefas)

Nível de sistema operacional

-Alocação dinâmica de tarefas com crescente overhead de escalonamento devido a custos de migração.

-Muito uso dos serviços mutex do sistema operacional provavelmente resultará e defeitos funcionais relativos a temporização.

-Alocação estática de tarefas pode levar a baixa performance devido ao mau uso de alguns núcleos.

4.11 A FERRAMENTA DE ANALISE TEMPORAL GLIWA T1 SUITE

O pacote de software T1 consiste em dois módulos distintos, o módulo alvo T1 (T1-TARGET-SW) que é executado no dispositivo alvo, e o módulo principal T1 (T1-HOST-SW) que é executado em um computador padrão Windows (PC).

O módulo T1-TARGET-SW se vale de instrumentação altamente eficiente para obter informações relevantes de temporização dentro do próprio dispositivo alvo e as entrega para o módulo T1-HOST-SW no PC. Por “instrumentação altamente eficiente” entende-se que são porções de software desenvolvidas para monitorar as informações desejadas no dispositivo alvo sem causar impacto significativo no sistema e nas próprias informações que estão sendo monitoradas. Esta eficiência pode ser quantificada e numa aplicação automotiva típica de 32 bits ela causa um acréscimo de carga na CPU entre 0,2% e 0,8%.

Figura 12 – Conexão entre o PC executando o módulo principal T1 e o dispositivo alvo (ECU) executando o módulo alvo T1

Fonte: Gliwa(2016): Adaptado pelo Autor

A interface com o dispositivo alvo pode ser dos padrões CAN, FlexRay™, JTAG ou Nexus. O pacote T1 requer muito pouca largura de banda para o fluxo de dados entre o PC e o dispositivo alvo, possibilitando sua utilização em projetos de produção em larga escala, onde a largura de banda é tipicamente limitada.

Quando os dados são transmitidos para o PC eles são visualizados e analisados. A geração de relatórios e a facilidade para automatizar as funções do pacote T1 proporcionam perfeita integração aos procedimentos de teste.

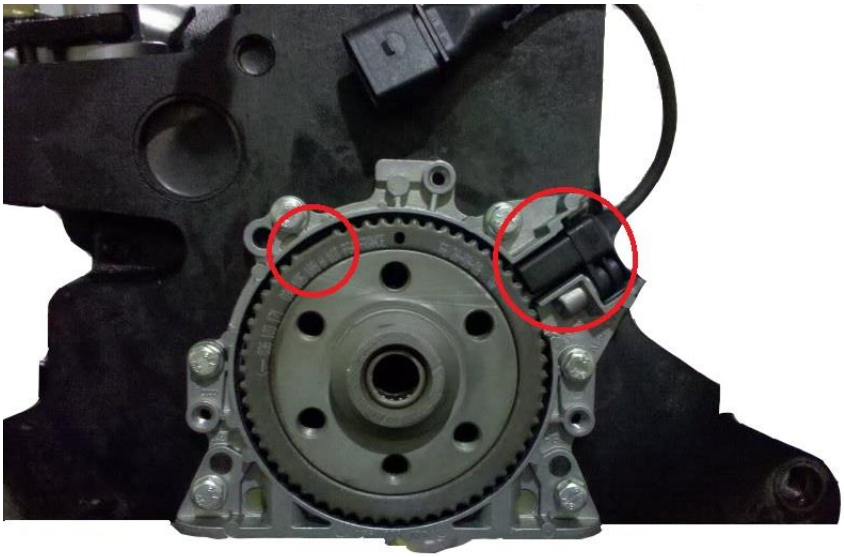
O modulo alvo T1 (T1-TARGET-SW), além de coletar dados, pode realizar a analise em tempo real dos dados coletados e detectar se restrições temporais foram violadas. Nestas situações ele pode, por exemplo, salvar informações de ambiente de forma a permitir uma analise mais apurada da situação de erro e das causas que levaram a tal erro. Desta forma até mesmo as mais complexas causas de erro de execução podem ser diagnosticadas.

5 AS TAREFAS ANGULARES E AS TAREFAS ADAPTATIVAS DE TAXA DE ATIVAÇÃO VARIÁVEL (AVR)

5.1 TAREFAS ANGULARES

A central de gerenciamento (ECM) de um motor a combustão interna (ICE) precisa de diversas informações sobre o estado atual do motor como, por exemplo, a temperatura do líquido de arrefecimento, a pressão na linha de combustível, a temperatura do ar na admissão, etc. Uma das principais informações necessárias é a posição do virabrequim, pois sem esta informação não há como a ECM saber o momento correto para comandar diversas atividades como a injeção de combustível ou o acionamento do circuito de ignição que gera a faísca para que a combustão ocorra.

Figura 13 – Roda fônica com 60 menos 2 dentes. Destaque para a região dos dois dentes ausentes e para o sensor de rotação CKP



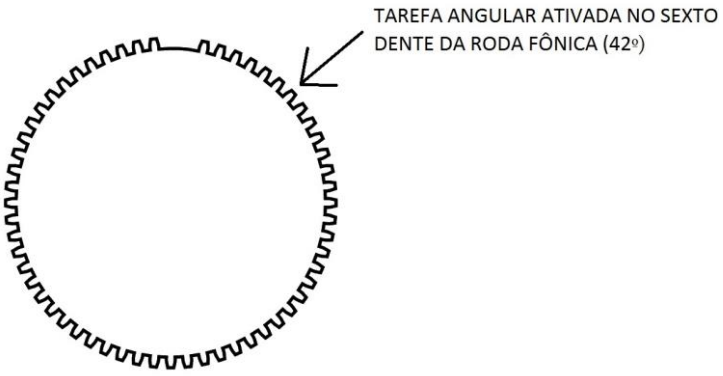
Fonte: Adaptado pelo Autor(2016)

Para fornecer esta informação geralmente utiliza-se a roda fônica. A roda fônica (Figura 13) nada mais é do que de um disco dentado acoplado ao eixo do virabrequim do motor que, em conjunto com um sensor, gera o sinal elétrico que possibilita à ECM identificar a posição e a velocidade do motor. Desta forma torna-se possível a ativação de tarefas em ângulos específicos da roda fônica. São as tarefas

angulares, descritas por Guzzella e Onder (2010) como tarefas relacionadas à rotação do virabrequim e ativadas em ângulos de rotação específicos.

As tarefas angulares geralmente são tarefas ativadas por interrupção (ISR) e ocorrem em posições angulares especificas do virabrequim. São tarefas que devem ser ativadas no momento exato em que o virabrequim atinge a posição angular (Figura 14) determinada pela CPU em função das informações e valores recebidos através dos diversos sensores que compõem o sistema ciber-físico de gerenciamento do motor.

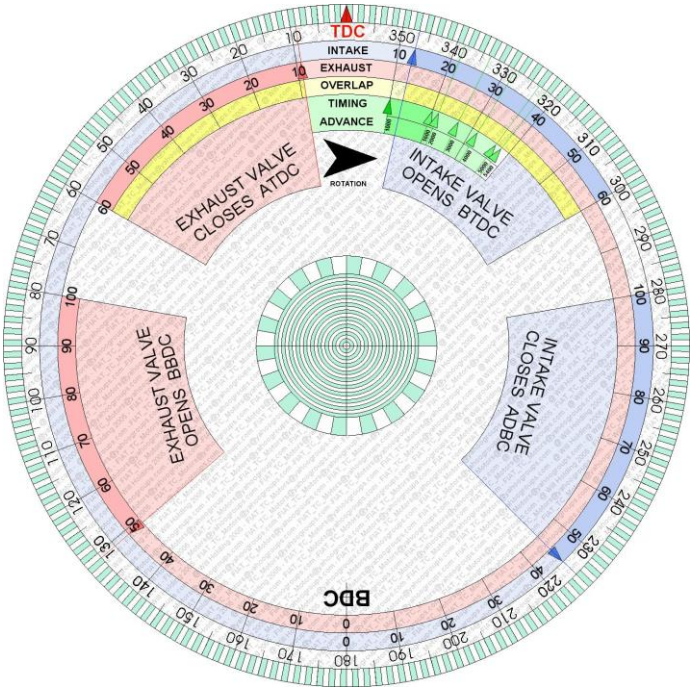
Figura 14 – Exemplo de tarefa angular ativada quando o motor estiver na posição de 42º



Fonte: Elaborado pelo Autor(2016)

Estas tarefas geralmente são criticas porque seu período de ativação é variável e por isso elas podem causar efeitos inesperados na carga da CPU em altas rotações caso o sistema não esteja bem escalonado e dimensionado.

Figura 15 – Roda fônica didática



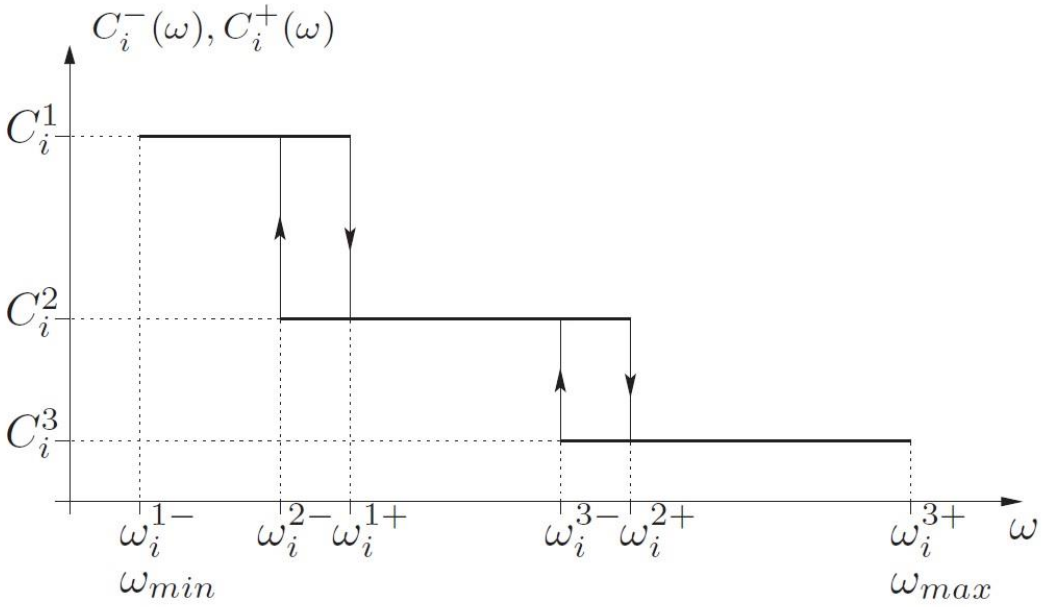
Fonte: Fiat_TC_motors(2016)

Através da roda fônica didática (Figura 15) nota-se o ponto morto superior (TDC), ponto morto inferior (BDC) e os valores típicos para os ângulos onde deverão ocorrer a abertura e fechamento das válvulas de admissão e escape do motor.

5.2 TAREFAS ADAPTATIVAS DE TAXA DE ATIVAÇÃO VARIÁVEL (AVR)

As tarefas adaptativas de taxa de ativação variável (AVR) são tarefas que geralmente são ativadas por interrupção e não possuem um período determinado. Nem toda tarefa ativada por interrupção é uma tarefa AVR, porém algumas tarefas que são ativadas por interrupção e, além de possuírem uma variação muito grande no seu período de ativação, possuem um tempo de execução (CET) significativo, podem levar a uma sobrecarga do microcontrolador. A utilização da tecnologia AVR pode minimizar este problema fazendo com que algumas funcionalidades destas tarefas sejam desabilitadas na medida em que a sua frequência de ativação aumenta.

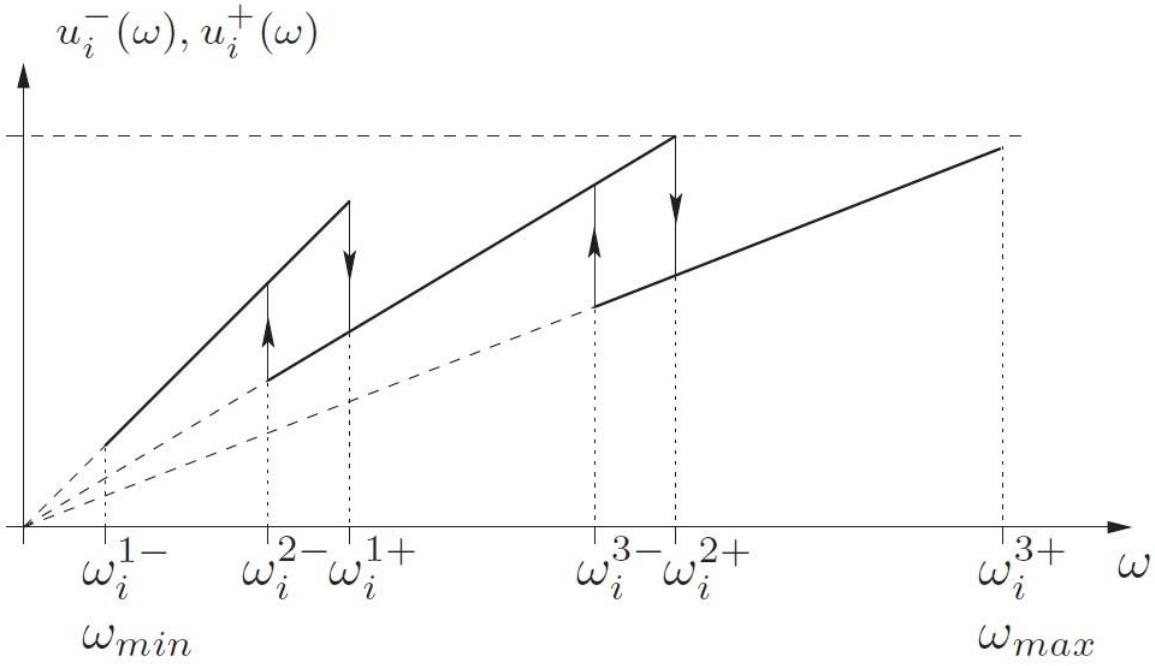
Figura 16 – WCET de uma tarefa AVR em função da velocidade de rotação do motor



Fonte: Biondi E Buttazzo(2015)

Uma forma de se implementar esta solução seria a variação contínua da funcionalidade das tarefas em função da velocidade de rotação do motor, porém não é viável o desenvolvimento de uma tarefa que tenha a sua funcionalidade sendo alterada continuamente. Portanto, o que se faz é definir um conjunto de modos de operação e se estabelecer um destes modos de operação para cada faixa de rotação do motor. Desta forma basta haver uma versão da tarefa para cada modo de operação, garantindo que a funcionalidade desta tarefa seja reduzida a cada incremento no modo de operação. Para que não ocorra instabilidade no sistema nas velocidades de rotação limítrofes entre modos de operação adjacentes, é determinada uma histerese (Figura 16) onde a velocidade de rotação necessária para a alteração de um modo de operação para o modo subsequente é diferente da velocidade de rotação necessária para a alteração contrária.

Figura 17 – Utilização da CPU pela tarefa AVR em função da velocidade de rotação do motor



Fonte: Biondi E Buttazzo(2015)

Sendo assim, um conjunto de modos de operação é definido em função da velocidade de rotação do motor e as tarefas AVR reduzem sua demanda computacional a medida que esses modos Desta forma o seu tempo de execução (CET) é reduzido quando o motor atinge faixas maiores de velocidade de rotação (Figura 17) e a carga na CPU não sofre um aumento tão expressivo ou até mesmo sofre uma redução em algumas situações limítrofes.

5.3 MODELO DO SISTEMA COM AVR

O modelo matemático proposto por Biondi e Buttazzo(2015) é subdividido em modelo da fonte de rotação e modelo das tarefas conforme descrito a seguir.

5.3.1 Modelo da fonte de rotação

Considerando-se uma fonte única de rotação (o motor) caracterizada pelas seguintes variáveis:

θ – Posição angular instantânea do virabrequim

ω – Velocidade angular instantânea do virabrequim

α – Aceleração angular instantânea do virabrequim

Considera-se que a velocidade ω é limitada dentro de uma faixa [ω_{min} , ω_{max}], que a aceleração α é limitada dentro de uma faixa [α^- , α^+] e que a variação da aceleração α durante uma rotação completa do motor é desprezível.

5.3.2 Modelo das tarefas

As atividades computacionais consideradas são representadas por um conjunto de *n* tarefas de tempo real preemptivas $\Gamma = \{ \tau_1, \tau_2, \tau_3, \ldots, \tau_n \}$. Cada tarefa pode ser uma tarefa periódica clássica ativada a intervalos fixos de tempo ou uma tarefa angular ativada em posições angulares específicas do virabrequim. As tarefas angulares possuem um período de ativação variável (inversamente proporcional à velocidade de rotação do motor ω) e adaptam a sua funcionalidade para velocidades de rotação diferentes, portanto são chamadas de tarefas adaptativas de taxa de ativação variável (AVR). O subconjunto das tarefas periódicas clássicas é representado por Γ_P e o subconjunto das tarefas AVR é representado por Γ_A, portanto $\Gamma = \Gamma_P \cup \Gamma_A$ e $\Gamma_P \cap \Gamma_A = \emptyset$. Quando necessário uma tarefa adaptativa AVR também poderá ser representada por τ_i^*.

Ambos os tipos de tarefas são caracterizados por um tempo de execução no pior caso (worst case execution time WCET) C_i, um período T_i e um tempo limite (*deadline*) D_i. Entretanto, enquanto tais parâmetros são fixos para as tarefas periódicas regulares, eles dependem da velocidade de rotação do motor para as tarefas angulares.

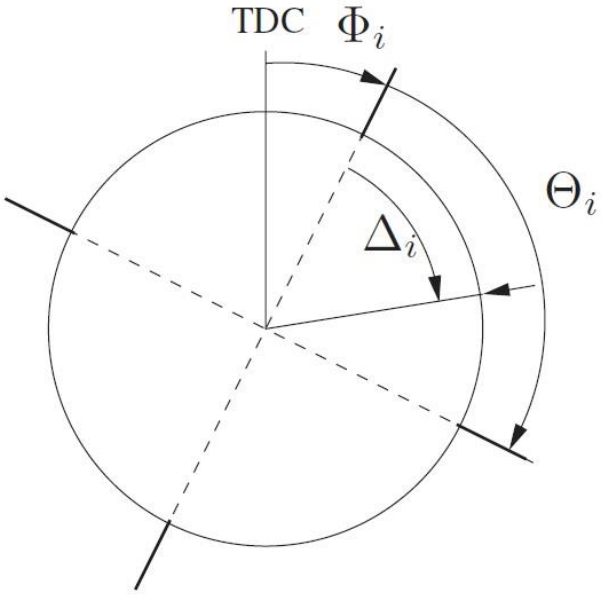
Uma tarefa angular τ_i^* é caracterizada por um período angular Θ_i e por uma fase angular Φ_i, de forma que ela é ativada nos seguintes ângulos:

$$\theta_i = \Phi_i + k\Theta_i, \text{ para } k = 0,1,2,\ldots \quad (5.1)$$

Isto significa que o período de uma tarefa adaptativa AVR é inversamente proporcional à velocidade de rotação do motor ω e pode ser expressa por:

$$\boldsymbol{T_i(\omega) = \frac{\Theta_i}{\omega}} \tag{5.2}$$

Figura 18 – Parâmetros de uma tarefa angular representada em uma rotação do motor



Fonte: Biondi E Buttazzo(2015)

Uma tarefa angular τ_i^* também é caracterizada por uma *deadline* Δ_i expressa como uma fração δ_i do período angular ($\delta_i \in [0,1]$). Sendo assim, $\Delta_i = \delta_i \Theta_i$ representa a *deadline* angular relativa.

A Figura 18 ilustra graficamente os parâmetros de uma tarefa angular representada um uma rotação do motor. Note-se que a fase angular é relativa a posição de referencia chamada ponto morto superior (TDC – *Top dead center*) que corresponde a posição do virabrequim na qual o pistão atingiu sua posição mais alta dentro do cilindro.

Quando do agendamento da tarefa no sistema operacional de tempo real (RTOS), uma *deadline* absoluta deve ser atribuída a cada tarefa quando da sua ativação para que ela seja agendada. Embora a *deadline* angular seja constante, a *deadline* temporal é uma função de ω e, para uma velocidade de rotação constante, é igual a:

$$\boldsymbol{D_i(\omega) = \frac{\Delta_i}{\omega} = \frac{\delta_i \Theta_i}{\omega} = \delta_i T_i(\omega)} \tag{5.3}$$

Entretanto o valor do próximo período é desconhecido, uma vez que a velocidade angular do motor ω não é constante.

O tempo de execução de uma tarefa angular τ_i^* também é função da velocidade de rotação do motor, uma vez que as tarefas adaptativas AVR adaptam sua funcionalidade para reduzir a utilização da CPU em velocidades mais altas. Estas tarefas são implementadas como um conjunto de modos, cada um deles operando dentro de faixas específicas de rotação, além da histerese introduzida para evitar instabilidades do sistema em rotações próximas a faixas de rotação limítrofes. Portanto, o tempo de processamento de uma tarefa AVR pode ser descrito por uma função degrau composta por M_i modos de execução, onde cada modo m $(m = 1, \ldots, M_i)$ é definido por um tempo computacional C_i^m e uma faixa de rotações $[\omega_i^{m-}, \omega_i^{m+}]$.

Note-se que devido à histerese nas comutações de modos, o tempo computacional de uma tarefa AVR depende não somente do valor de ω, mas também do modo atual m, levando a duas alternativas de comportamento caracterizadas pelas funções a seguir:

$$C_i^+(\omega) = C_i^m, \qquad \forall \omega \in (\omega_i^{(m-1)+}, \omega_i^{m+}] \tag{5.4}$$

$$C_i^-(\omega) = C_i^m, \qquad \forall \omega \in [\omega_i^{m-}, \omega_i^{(m+1)-}) \tag{5.5}$$

Similarmente, a utilização da CPU por uma tarefa AVR com histerese pode ser definida pelas seguintes funções:

$$u_i^+(\omega) = \frac{C_i^+(\omega)}{T_i(\omega)} = \frac{\omega C_i^+(\omega)}{\Theta_i} \tag{5.6}$$

$$u_i^-(\omega) = \frac{C_i^-(\omega)}{T_i(\omega)} = \frac{\omega C_i^-(\omega)}{\Theta_i} \quad (5.7)$$

5.3.3 Exemplo

A tabela (Tabela 3) ilustra um exemplo de uma tarefa AVR com três modos especificados para três intervalos de rotação diferentes, mas que se sobrepõem. Cada modo m executa uma função diferente caracterizada por um tempo computacional C^m e as transições de modo tem uma histerese de 500 rpm, ou seja, a transição do modo 1 para o modo 2 ocorre a 2000 rpm mas a transição contraria só ocorre a 1500 rpm.

Tabela 3 – Exemplo de uma tarefa AVR com três modos

Modo	C^m	$[\omega^{m-}, \omega^{m+}]$ (rpm)	Funcionalidade
1	C^1	[500, 2000]	f1()
2	C^2	[1500, 4000]	f2()
3	C^3	[3500, 6000]	f3()

Fonte: Biondi E Buttazzo(2015)

Aplicando-se os valores da tabela (Tabela 3) nas equações do custo computacional de uma tarefa AVR temos:

$$C_i^+(\omega) = C_i^m, \qquad \forall\omega \in (\omega_i^{(m-1)+}, \omega_i^{m+}]$$

$$C_i^+(\omega) = C_i^1, \qquad \forall\omega \in (\omega_i^{0+}, \omega_i^{1+}]$$
$$C_i^+(\omega) = C_i^1, \qquad \forall\omega \in (-, 2000]$$

$$C_i^+(\omega) = C_i^2, \qquad \forall\omega \in (\omega_i^{1+}, \omega_i^{2+}]$$
$$C_i^+(\omega) = C_i^2, \qquad \forall\omega \in (2000, 4000]$$

$$C_i^+(\omega) = C_i^3, \qquad \forall\omega \in (\omega_i^{2+}, \omega_i^{3+}]$$
$$C_i^+(\omega) = C_i^3, \qquad \forall\omega \in (4000, 6000]$$

$$C_i^-(\omega) = C_i^m, \qquad \forall\omega \in [\omega_i^{m-}, \omega_i^{(m+1)-})$$

$$C_i^-(\omega) = C_i^1, \qquad \forall\omega \in [\omega_i^{1-}, \omega_i^{2-})$$
$$C_i^-(\omega) = C_i^1, \qquad \forall\omega \in [500, 1500)$$

$$C_i^-(\omega) = C_i^2, \qquad \forall\omega \in [\omega_i^{2-}, \omega_i^{3-})$$
$$C_i^-(\omega) = C_i^2, \qquad \forall\omega \in [1500, 3500)$$

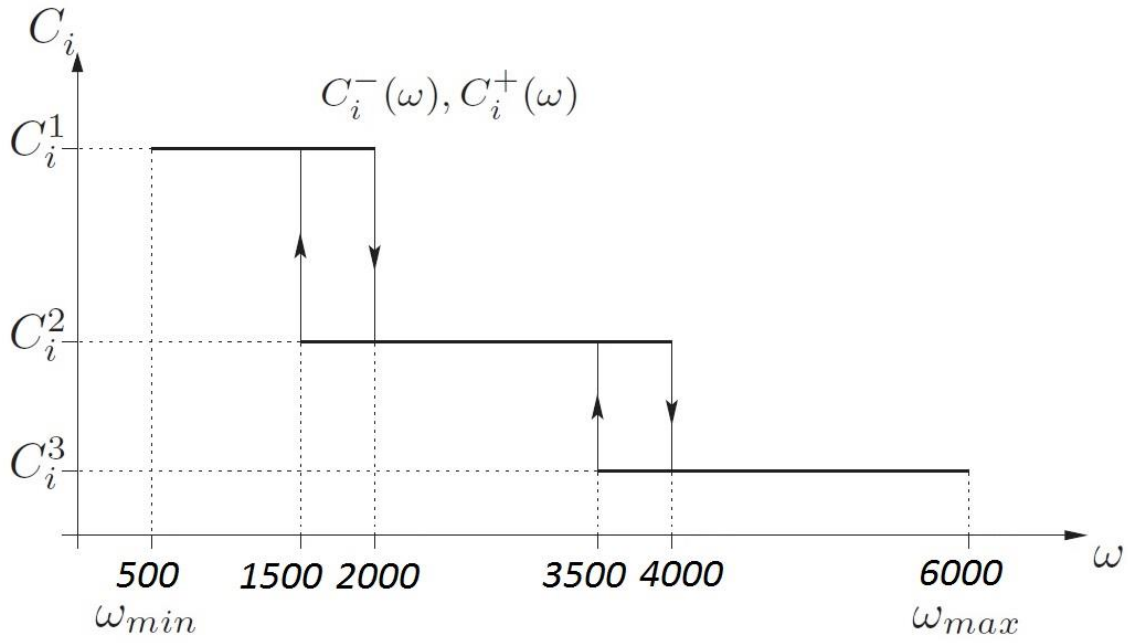
$$C_i^-(\omega) = C_i^3, \qquad \forall\omega \in [\omega_i^{3-}, \omega_i^{4-})$$
$$C_i^-(\omega) = C_i^3, \qquad \forall\omega \in [3500, -)$$

Portanto tem-se:

$$\begin{cases} C_i^+(\omega) = C_i^1, & \forall\omega \in (-, 2000] \\ C_i^+(\omega) = C_i^2, & \forall\omega \in (2000, 4000] \\ C_i^+(\omega) = C_i^3, & \forall\omega \in (4000, 6000] \end{cases}$$

$$\begin{cases} C_i^-(\omega) = C_i^1, & \forall\omega \in [500, 1500) \\ C_i^-(\omega) = C_i^2, & \forall\omega \in [1500, 3500) \\ C_i^-(\omega) = C_i^3, & \forall\omega \in [3500, -) \end{cases}$$

Figura 19 – Custo computacional em função da velocidade de rotação do motor ω



Fonte: Biondi E Buttazzo: Adaptado pelo Autor(2016)

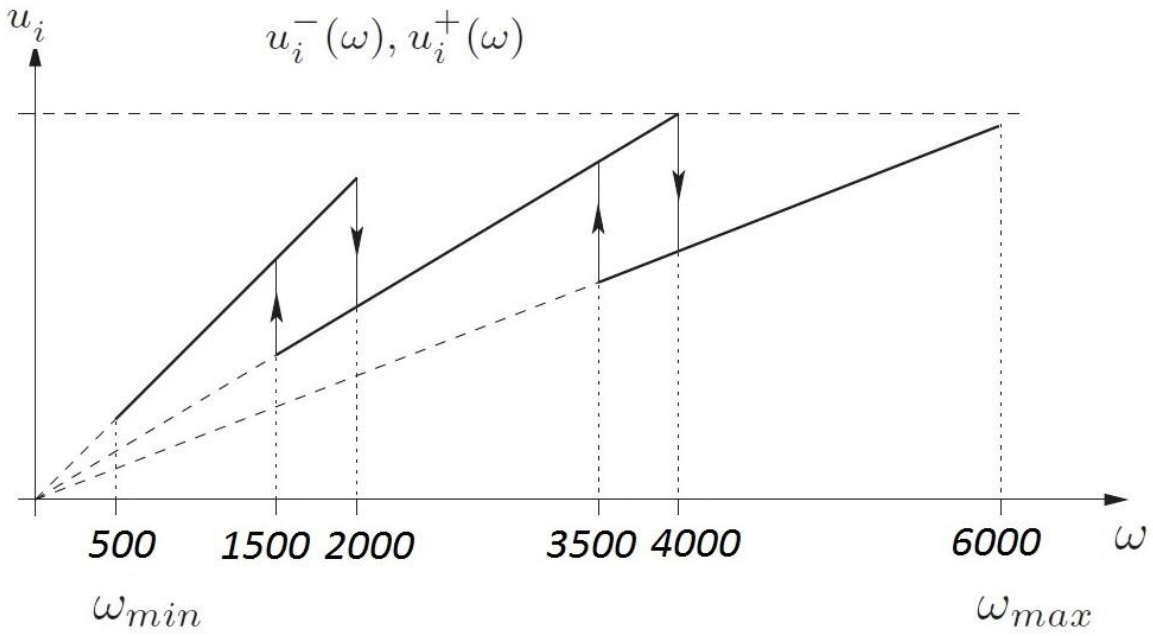
A utilização da CPU para o exemplo citado também pode ser obtida através das formulas:

$$u_i^+(\omega) = \frac{C_i^+(\omega)}{T_i(\omega)} = \frac{\omega C_i^+(\omega)}{\Theta_i}$$

$$u_i^-(\omega) = \frac{C_i^-(\omega)}{T_i(\omega)} = \frac{\omega C_i^-(\omega)}{\Theta_i}$$

Uma vez que o período angular Θ_i é constante temos que a utilização da CPU nada mais é do que o custo computacional multiplicado por um fator constante e pela velocidade angular de rotação do motor ω, portanto pode-se traçar o seguinte gráfico do custo em função da velocidade:

Figura 20 – Utilização computacional em função da velocidade de rotação do moto ω



Fonte: Biondi E Buttazzo: Adaptado pelo Autor(2016)

5.4 ATIVANDO A TAREFA AVR UTILIZADA

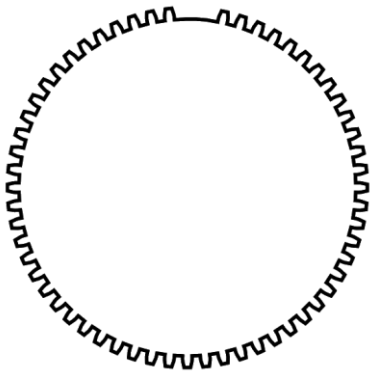
Diversas tarefas controladas pela ECM de um motor a combustão devem ser ativadas em posições angulares bem determinadas, especialmente as tarefas que controlam a injeção do combustível na quantidade e momento exatos para cada um dos cilindros do motor e também o instante e a quantidade de energia com a qual deve ser acionado o circuito de ignição que irá gerar a faísca que resultará no processo de combustão.

São as tarefas angulares, descritas anteriormente neste capítulo. Tais tarefas não são tarefas agendadas pelo sistema operacional de tempo real RTOS, elas são ativadas por posição angular e, para tanto, torna-se necessário o sinal de um sensor que envie à central de gerenciamento do motor (ECM) informações que possibilitem a determinação do valor de θ (posição angular instantânea do motor). Assim a ECM pode determinar a ativação da tarefa AVR no instante em que o motor estiver na posição θ_i para a qual esta tarefa foi programada.

O sensor que realiza tal tarefa é o sensor CKP. O sensor CKP (crankshaft position sensor) é o sensor de posição do virabrequim. O sinal do sensor CKP é gerado através de uma roda dentada fixada ao eixo do virabrequim (roda fônica) que

ao movimentar-se induz no sensor um sinal compatível com a sua forma geométrica. Neste trabalho foi utilizado o sinal equivalente ao de uma roda fônica com 60 menos 2 dentes (Figura 21), ou seja, ela é dividida em 60 setores porém dois setores consecutivos não possuem dente de forma que o sinal gerado (Figura 22) possa ser utilizado para determinar a posição do ponto morto superior (PMS ou Top Dead Center - TDC), possibilitando assim a determinação da posição do virabrequim, além da sua velocidade de rotação.

Figura 21 – Roda fônica com 60 menos 2 dentes



Fonte: Elaborado pelo Autor(2016)

Desta forma podemos determinar a posição do virabrequim com uma resolução de 6 graus utilizando o sinal do sensor CKP.

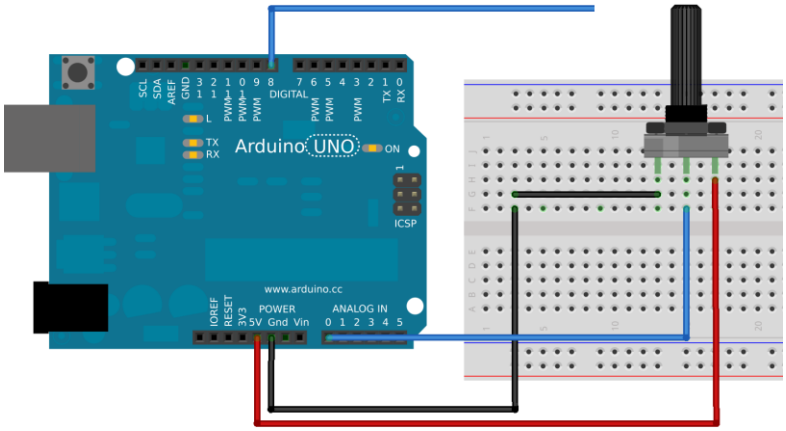
Figura 22 – Sinal gerado pelo sensor de posição do virabrequim – CKP

Fonte: Elaborado pelo Autor(2016)

Com o objetivo de simular o sinal do sensor CKP neste trabalho, foi utilizada a plataforma Arduino programada de forma a gerar o sinal 60-2 do CKP em sua saída digital D8. Para variar a frequência do sinal foi conectado um potenciômetro linear à sua entrada analógica A0 e o Arduino foi programado para variar o período do sinal de forma proporcional ao valor inverso da tensão lida em sua entrada analógica. Desta

forma a frequência do sinal de saída varia linearmente com a posição do potenciômetro, atuando assim de forma similar a uma roda fônica variando sua velocidade de rotação.

Figura 23 – Arduino UNO utilizado para simular sinal do sensor CKP



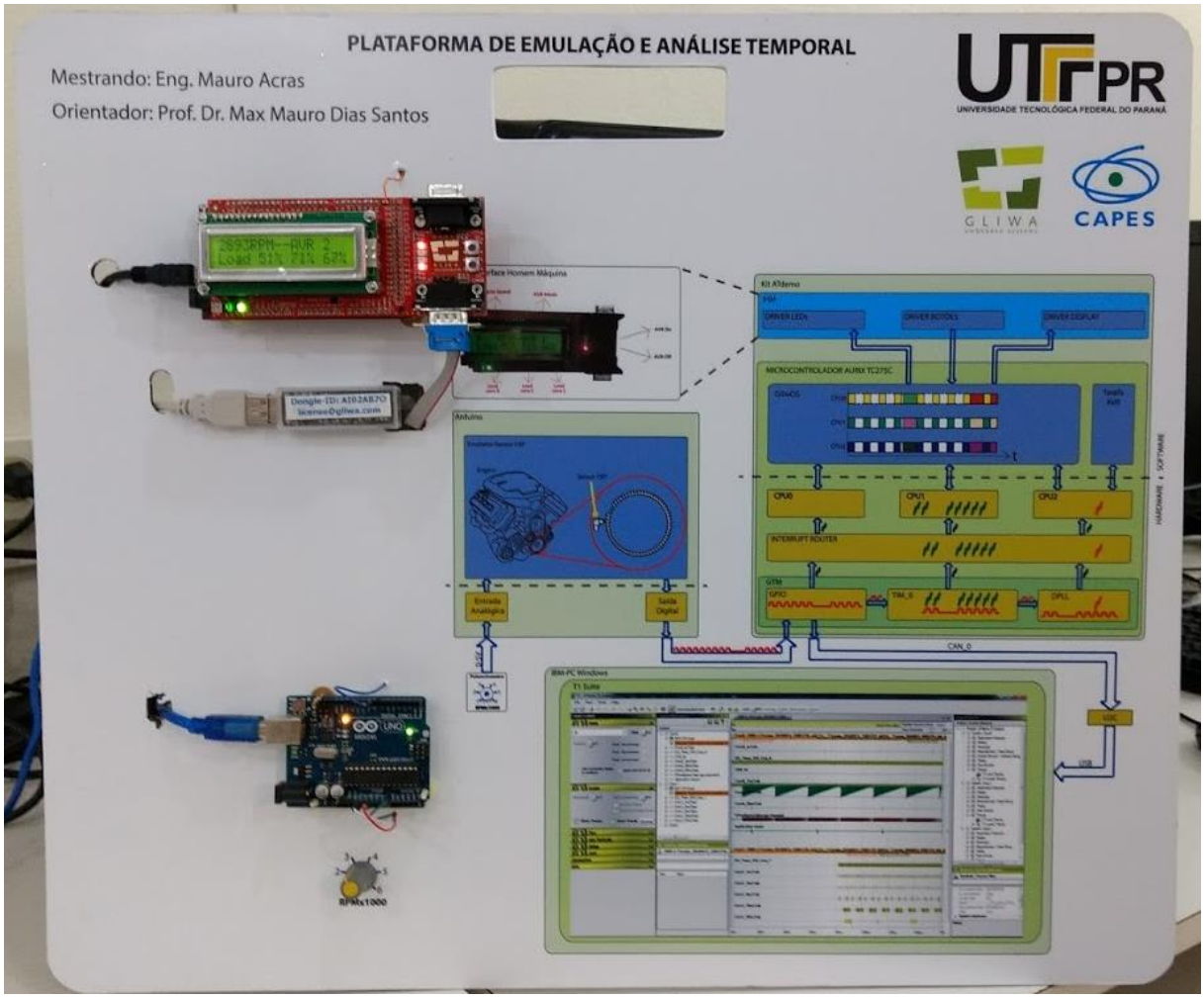
Fonte: Elaborado pelo Autor(2016)

6 METODOLOGIA

6.1 UMA PLATAFORMA DE EMULAÇÃO DE SISTEMAS DE GERENCIAMENTO MOTOR COM AVR

De forma a emular um sistema ciber-físico de gerenciamento de motor a combustão interna (ICE) com suporte a tarefas AVR foi utilizado o kit ATdemo descrito no capitulo 2, rodando o sistema operacional de tempo real RTOS GliwOS descrito no capítulo 3.

Figura 24 - Painel contendo o Sistema completo – Sensor CKP simulado + Kit ATdemo (ECM)



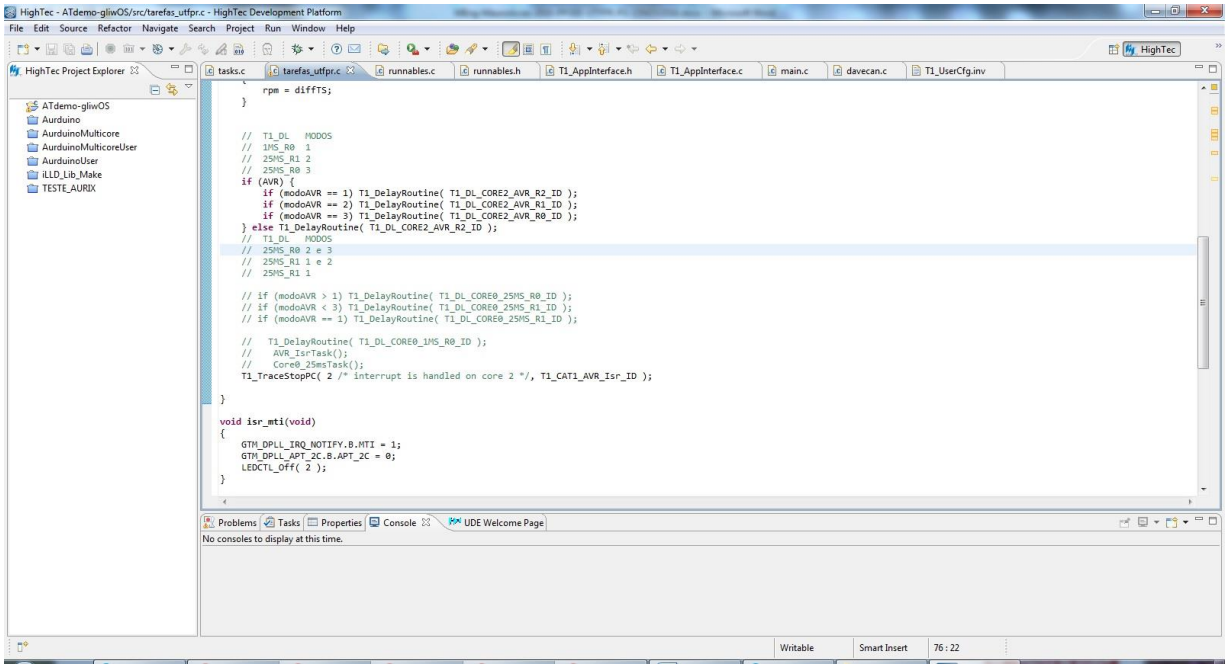
Fonte: Elaborado pelo Autor(2016)

Ao kit ATdemo foi conectado o Arduino emulando o sinal do sensor CKP de modo que as tarefas angulares AVR pudessem ser ativadas e posteriormente analisadas pela ferramenta de analise temporal T1. Todos os componentes foram montados em um painel (Figura 24) tornando o sistema mais didático e as medições e analises mais ágeis.

6.1.1 Ambiente de desenvolvimento

Foi utilizado o ambiente de desenvolvimento “Eclipse for tricore tool chain”. O ambiente eclipse é uma ferramenta de código aberto largamente utilizada para desenvolvimento de projetos, podendo ser personalizada para utilização nas mais diversas plataformas. A distribuição utilizada é fornecida pela HighTec EDV-Systeme GmbH e foi desenvolvida para plataformas Tricore e Aurix.

Figura 25 – Ambiente Eclipse utilizado para realizar alterações diretamente no código fonte do sistema operacional de tempo real RTOS



Fonte: Eclipse, Adaptado pelo Autor(2016)

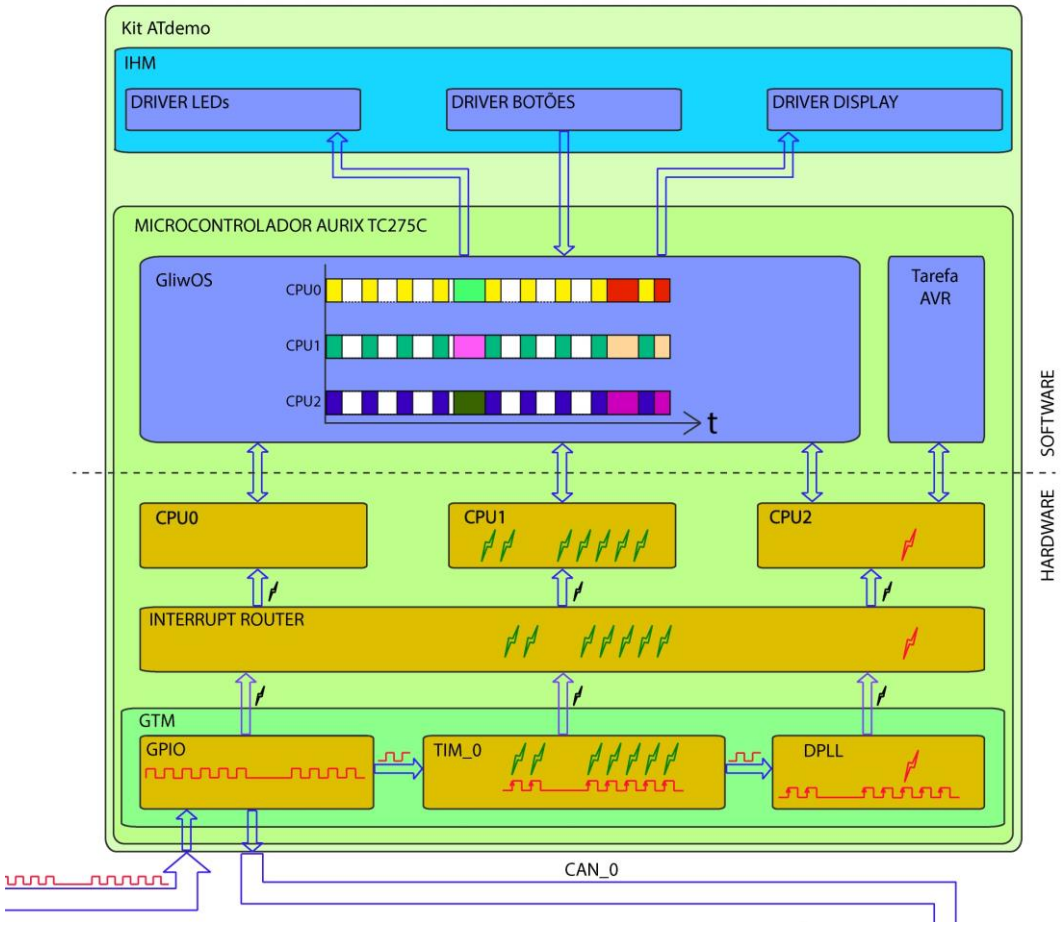
Várias rotinas de software foram desenvolvidos no decorrer deste trabalho na linguagem C++ no ambiente de desenvolvimento Eclipse e integrados ao sistema operacional de tempo real GliwOS. Tais rotinas foram necessárias para a

programação e configuração de diversos módulos internos do microcontrolador que receberam e trataram o sinal fornecido pelo emulador do sensor CKP e também a rotina de tratamento de interrupção da tarefa angular AVR ativada a partir do sinal recebido.

6.1.2 Configuração e programação do microcontrolador Infineon Aurix TC275

A estratégia utilizada para se obter as tarefas angulares foi a de se programar o microcontrolador para receber o sinal de posição do virabrequim (CKP), a partir do sinal recebido determinar a posição angular instantânea θ e então gerar uma ou mais interrupções nas posições angulares θ_i nas quais devem ser ativadas as i tarefas angulares existentes no sistema.

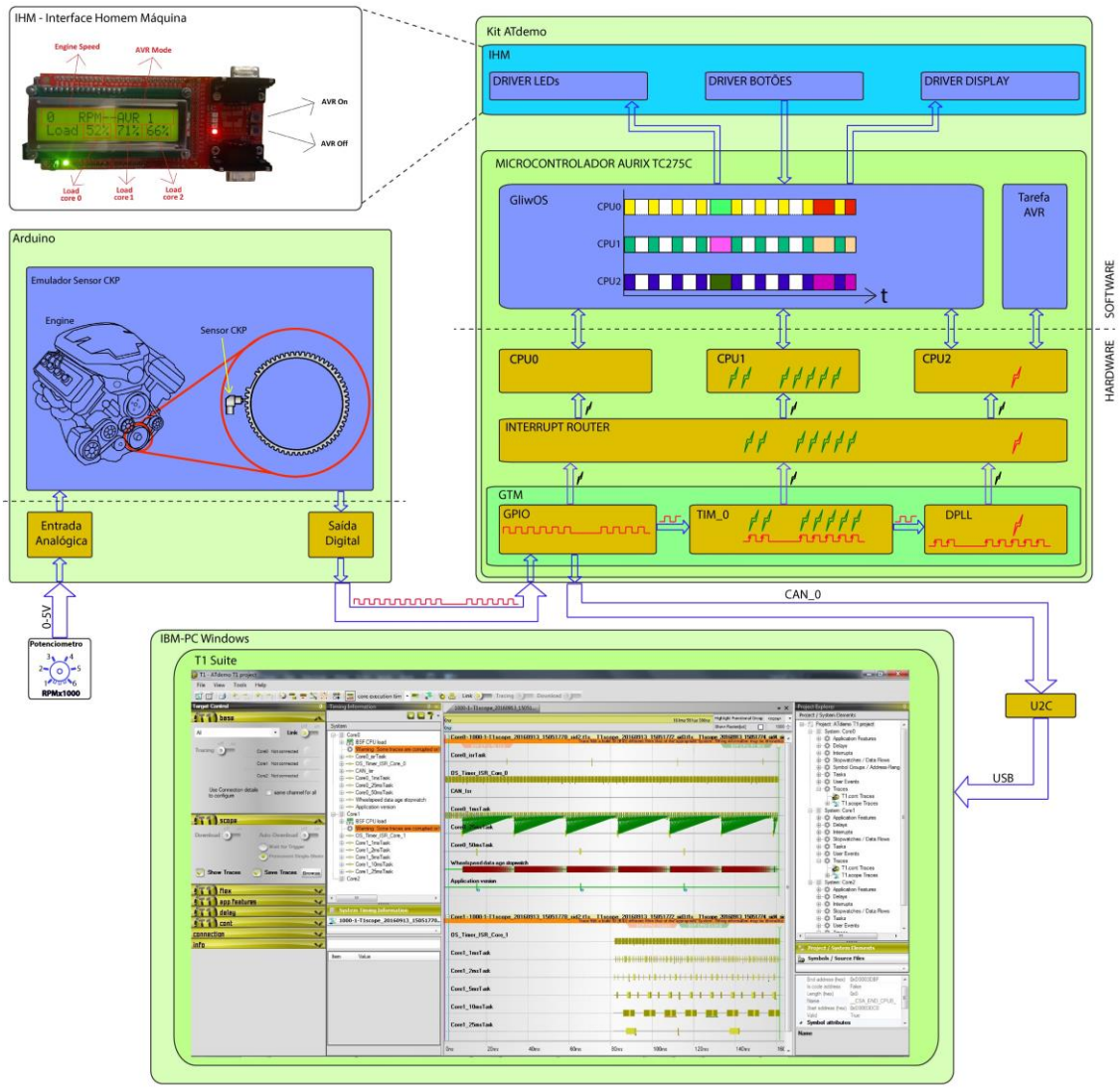
Figura 26 - Diagrama em blocos do Kit ATdemo



Fonte: Elaborado pelo Autor(2016)

Através do diagrama em blocos do kit ATdemo (Figura 26) pode-se notar o sinal do sensor CKP entrando no submódulo GPIO do microcontrolador AURIX, seguindo para o módulo TIM_0 e em seguida para o módulo DPLL, este sim responsável por gerar a interrupção que dará origem à tarefa AVR na posição angular desejada do virabrequim. O *interrupt router* então direciona esta interrupção para a CPU programada, a qual tratará esta interrupção executando a rotina de tratamento de interrupção que neste caso nada mais é do que a rotina AVR.

Figura 27 - 6.1 Diagrama em blocos completo da plataforma de emulação de sistemas de gerenciamento motor com avr

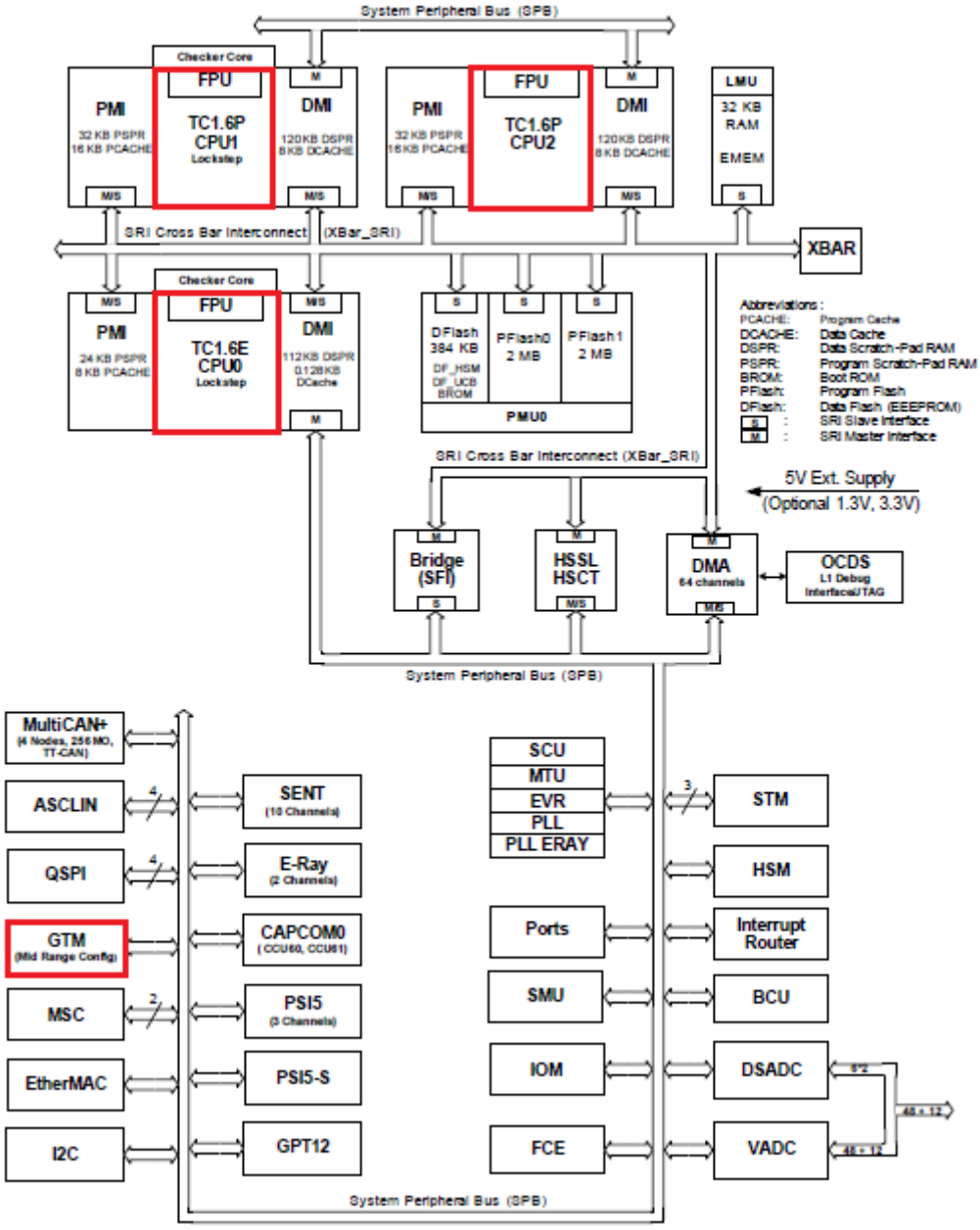


Fonte: Elaborado pelo Autor

Para se atingir tal objetivo foi necessário programar e configurar o microcontrolador, seus módulos, submódulos e a sua GPIO. O principal módulo

utilizado foi o módulo GTM do microcontrolador (Figura 28), que foi programado para que seus submódulos realizassem as operações necessárias e também para que houvesse a interação entre os módulos utilizados e os três núcleos de processamento.

Figura 28 – Arquitetura interna AURIX TC275 – em destaque os três núcleos e o módulo GTM



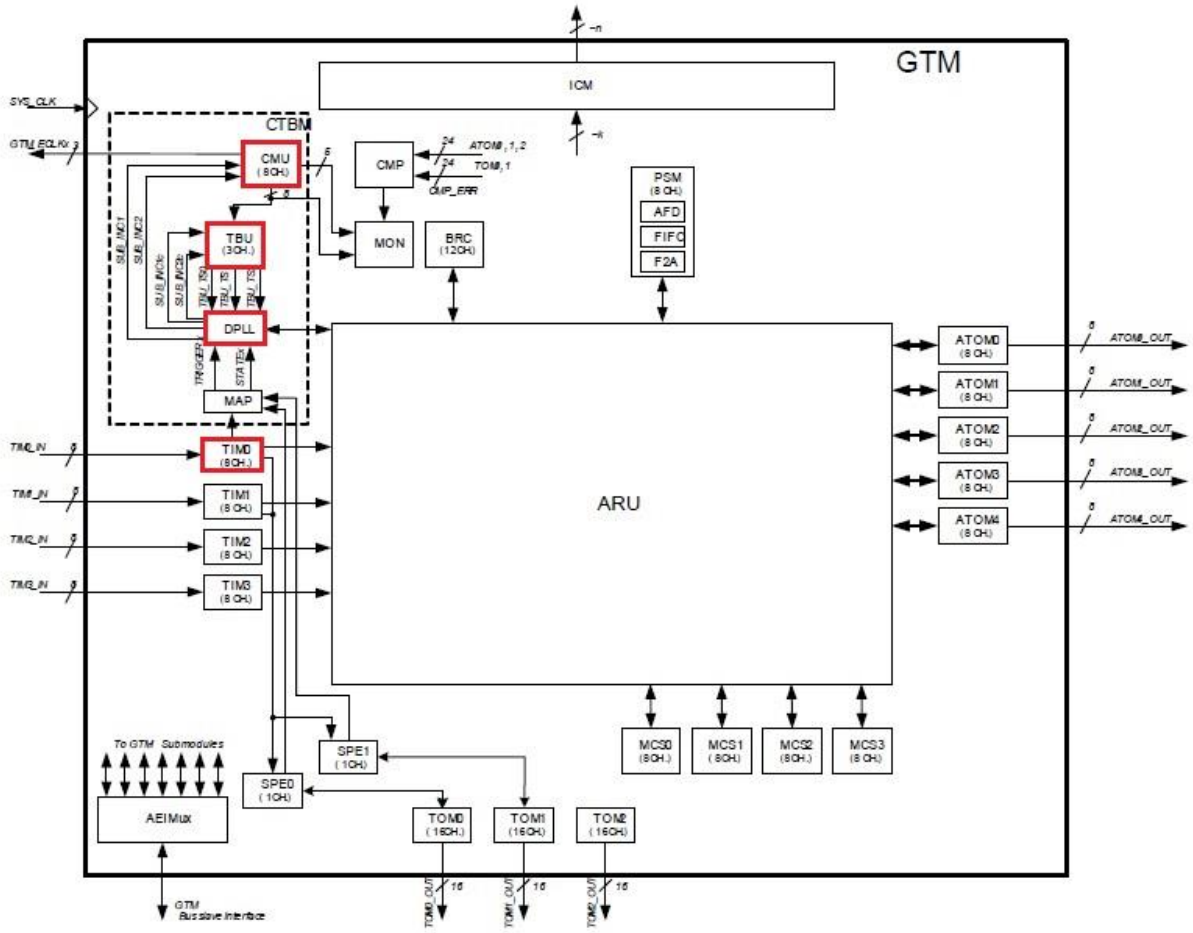
Fonte: Infineon, Adaptado pelo Autor(2016)

O módulo GTM, assim como o próprio microcontrolador, também contém uma estrutura modular com submódulos (Figura 29) que possuem funcionalidades diversas. A configuração e a combinação dos recursos oferecidos por estes

submódulos possibilitam desde a simples geração de sinais de clock personalizados para outros módulos ou submódulos, até a aquisição, processamento e geração de sinais necessários para o funcionamento de um motor a combustão interna (ICE) com reduzida necessidade de intervenções da CPU.

Para a determinação da posição angular instantânea θ utilizaram-se os submódulos TIM0, DPLL, CMU e TBU do módulo GTM.

Figura 29 – Módulo GTM – destaque para os submódulos utilizados para aquisição e tratamento do sinal CKP

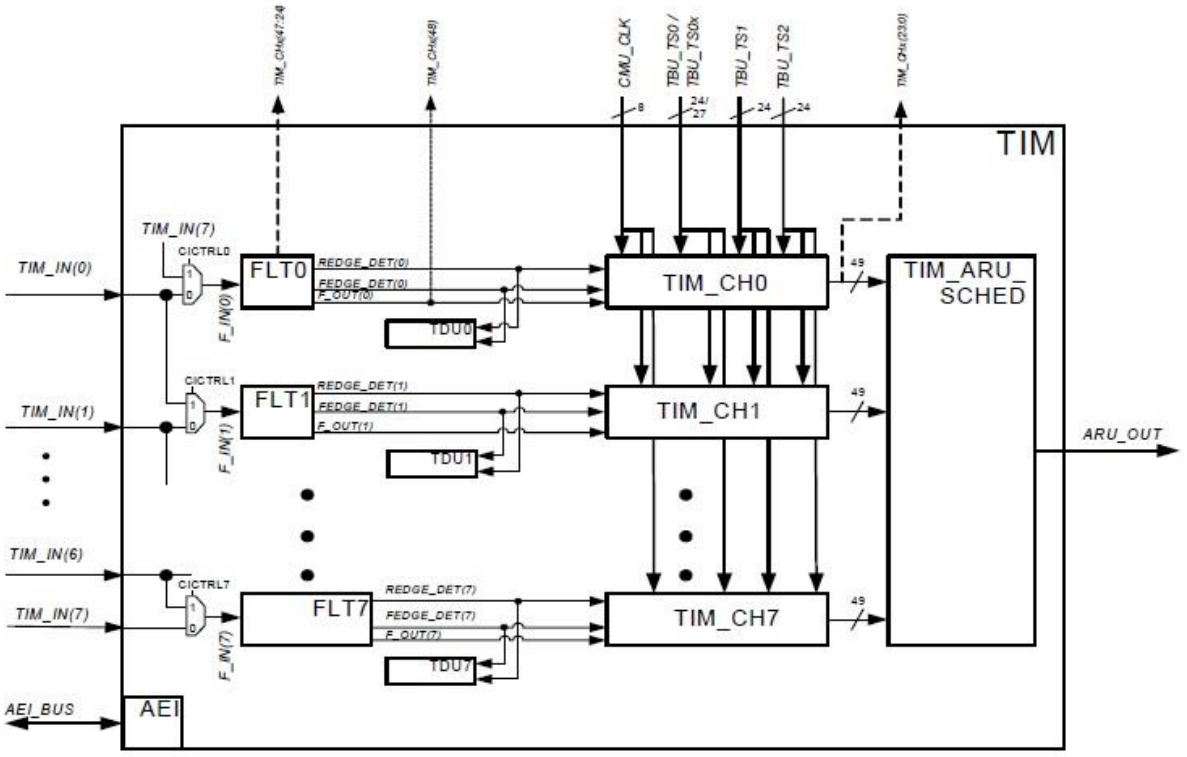


Fonte: Infineon, Adaptado pelo Autor(2016)

O submódulo TIM (Timer Input Module) é responsável pela aquisição e condicionamento de sinais de entrada do módulo GTM. Algumas características do sinal de entrada podem ser obtidas dentro do próprio submódulo TIM (Figura 30),

porém para aplicações mais avançadas os sinais podem ser direcionados para outros submódulos com recursos específicos para a aplicação desejada.

Figura 30 – Arquitetura interna do modulo TIM



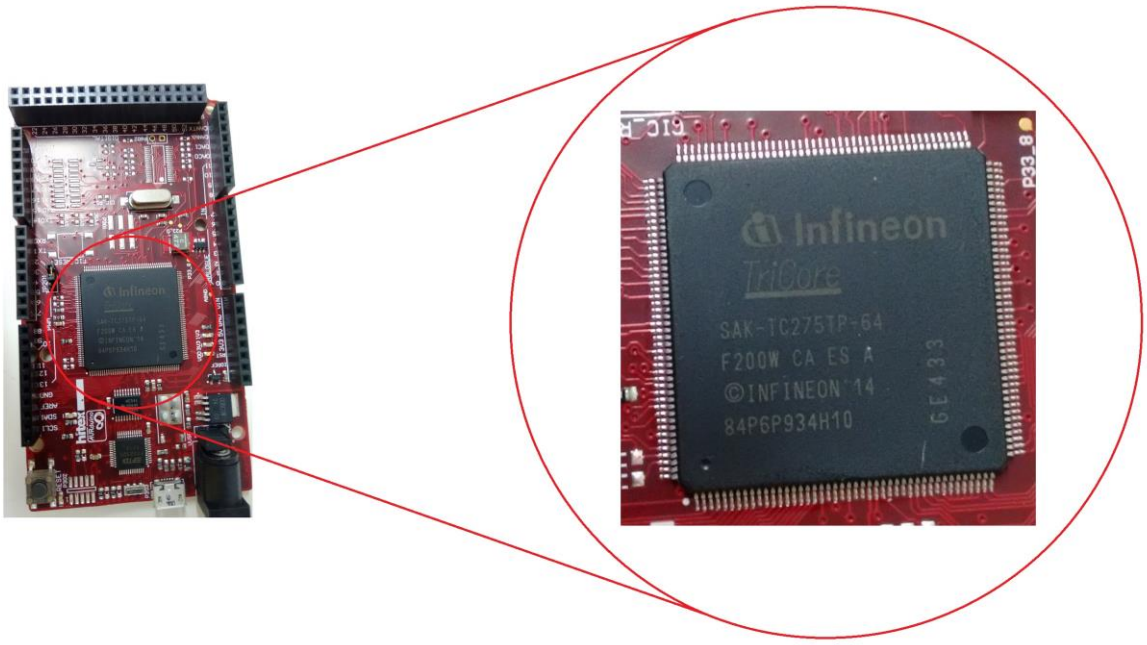
Fonte: Infineon, Adaptado pelo Autor(2016)

No caso deste trabalho é necessário que o submódulo TIM direcione o sinal recebido para o submódulo DPLL (Digital PLL Module) para que este realize a geração de interrupções nas posições angulares determinadas. Para que o submódulo DPLL consiga determinar a posição angular instantânea θ do virabrequim é necessário armazenar em seus registradores uma imagem do sinal CKP que deverá ser recebido, ou seja, a quantidade de rampas de subida e de descida que o sinal deve apresentar a cada rotação completa do virabrequim. Esta informação representa a geometria da roda fônica, ou seja, a quantidade de dentes presentes e dentes faltantes na sua construção e é necessária para que o submódulo DPLL sincronize seus circuitos internos com o sinal gerado pelo sensor CKP e assim possa determinar a posição angular θ. É o modulo DPLL que gera a interrupção na posição angular θ_i programada por um dos três núcleos de processamento do microcontrolador e tradada por uma rotina de tratamento de interrupção (ISR) que nada mais é do que a tarefa AVR sendo executada.

Para que o sinal do sensor CKP chegue aos módulos e submódulos internos mencionados anteriormente é necessário programar a GPIO (General Purpose Input/Output) do microcontrolador para direcionar para estes módulos alguma porta de entrada do microcontrolador que esteja disponível.

O microcontrolador Infineon AURIX TC27X é produzido em diversas versões com variações que vão desde o encapsulamento até a velocidade de clock.

Figura 31 – Detalhe do microcontrolador SAK-TC275TP-64F200W CA utilizado neste trabalho.



Fonte: Elaborado pelo Autor(2016)

A versão do microcontrolador instalada no kit ATdemo utilizado neste trabalho é a SAK-TC275TP-64F200W CA.

As principais especificações deste microcontrolador constam no documento “AURIX™ TC27x variants - Data Sheet Addendum” versão 1.1 de 10 de dezembro de 2014.

Figura 32 – Reprodução parcial "AURIX™ TC27x variants - Data Sheet Addendum" indicando os prefixos e as funcionalidades integradas ao dispositivo

Prefix

- SAK: $T_{ambient}$ Temperature Range from -40 °C up to +125 °C
- SAL: $T_{ambient}$ Temperature Range from -40 °C up to +150 °C (packaged device)

Feature Package

- T – Standard type without HSM
- TP – Standard type with HSM enabled
- TC – customer specific feature set

Fonte: Infineon, Adaptado pelo Autor(2016)

Tabela 4 – Principais características do microcontrolador utilizado

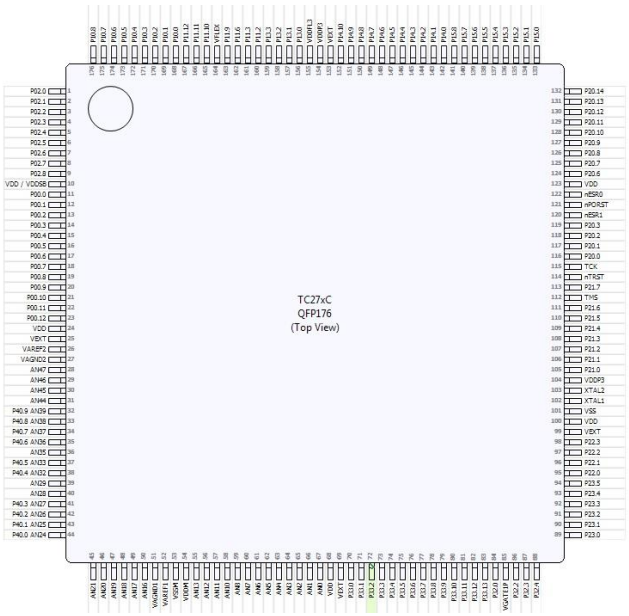
Versão	Status de produção	Encapsulamento
SAK-TC275TP-64F200W CA	STANDARD	PG-LQFP-176-22

Temperatura de operação	Chip ID	Freq.(MHz)	PFlash(MB)	
-40°C – +125°C	4742 7060H	200	4	
EEPROM(KB)	**Total SRAM(KB)**	**Core 1&2 TC16P DSPR (KB)**	**Core 1&2 TC16P PSPR (KB)**	**Core 0 TC16E DSPR (KB)**	**Core 0 TC16E PSPR (KB)**
64 @ 500k	472	120	32	112	24
LMU(KB)	**Canais ADC**	**FlexRay(#/ch.)**	**ETH**	**HSM**	**CAN FD**
32	48	½	Sim	Sim	Sim

Fonte: Infineon, Adaptado pelo Autor(2016)

Através destes dados do fabricante cumpre-se destacar o significado dos principais dados registrados na superfície do microcontrolador (Figura 31) como o prefixo SAK (Figura 32) que define a sua faixa de temperatura de operação, o sufixo TP que determina que esta versão tem habilitado o Módulo de Segurança por Hardware (HSM – Hardware Security Module) e que o encapsulamento do microcontrolador utilizado é o LQFP (Low Profile Quad Flat Package) com 176 pinos (Figura 33).

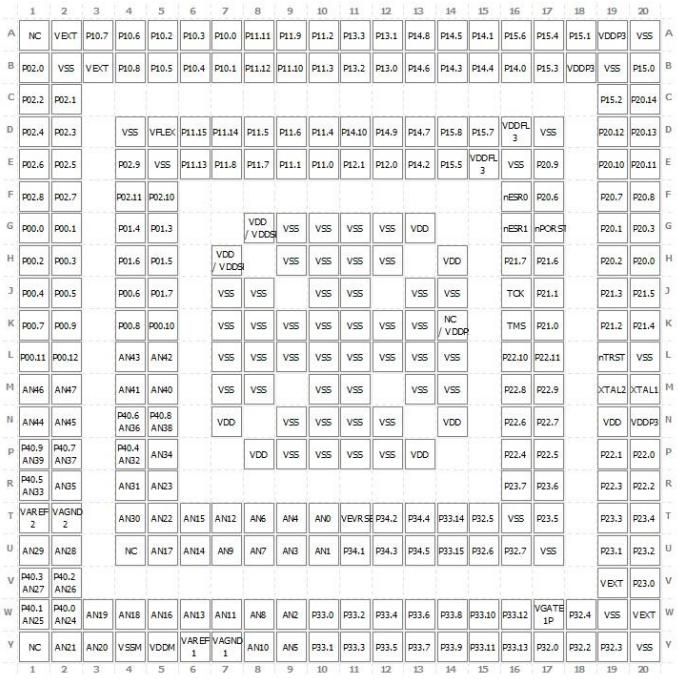
Figura 33 – Vista superior da estrutura física de um microcontrolador TC27xC com encapsulamento LQFP de 176 pinos



Fonte: Infineon(2016)

Algumas versões deste microcontrolador são encapsuladas no padrão LFBGA (Low Profile Fine Pitch Ball Grid Array) com 292 pinos (Figura 34) devido ao grande número de recursos e de módulos internos. A utilização do encapsulamento no padrão LQFP com apenas 176 pinos reduz significativamente a quantidade de portas disponíveis e pode dificultar a localização de uma entrada disponível para o sinal de entrada fornecido pelo sensor CKP, pois grande parte destas portas são utilizadas pela própria placa ShieldBuddy e também pelo GliwaShield com seus LEDs, botões, interfaces CAN, display e interface SDcard.

Figura 34 - Vista superior da estrutura física de um microcontrolador TC27xC com encapsulamento LFBGA de 292 pinos

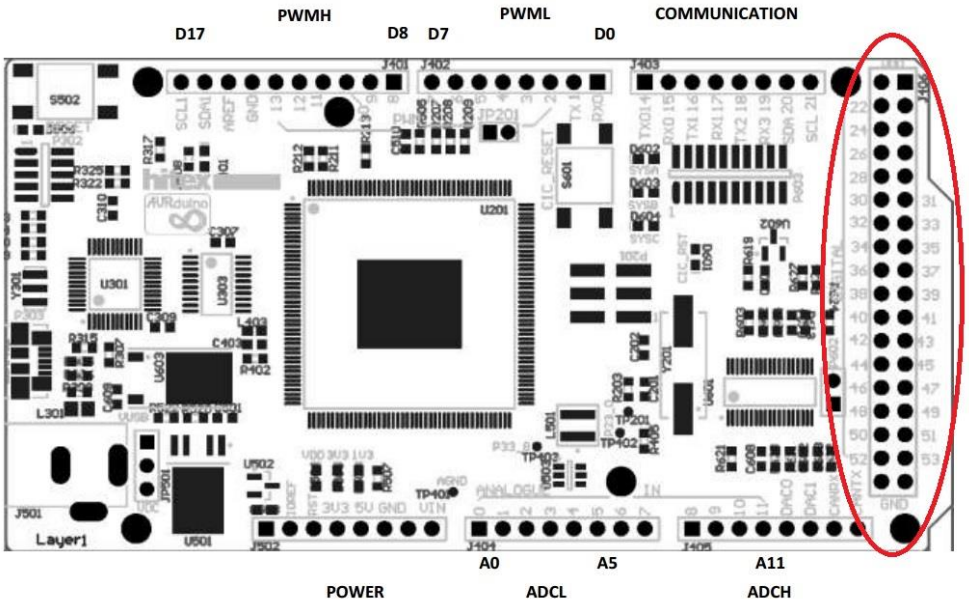


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
A	NC	VEXT	P10.7	P10.6	P10.2	P10.3	P10.0	P11.11	P11.9	P11.2	P13.3	P13.1	P14.8	P14.5	P14.1	P15.6	P15.4	P15.1	VDDP3	VSS	A
B	P02.0	VSS	VEXT	P10.8	P10.5	P10.4	P10.1	P11.12	P11.10	P11.3	P13.2	P13.0	P14.6	P14.3	P14.4	P14.0	P15.3	VDDP3	VSS	P15.0	B
C	P02.2	P02.1																	P15.2	P20.14	C
D	P02.4	P02.3		VSS	VFLEX	P11.15	P11.14	P11.5	P11.6	P11.4	P14.10	P14.9	P14.7	P15.8	P15.7	VDDFL 3	VSS		P20.12	P20.13	D
E	P02.6	P02.5		P02.9	VSS	P11.13	P11.8	P11.7	P11.1	P11.0	P12.1	P12.0	P14.2	P15.5	VDDFL 3	VSS	P20.9		P20.10	P20.11	E
F	P02.8	P02.7		P02.11	P02.10											nESR0	P20.6		P20.7	P20.8	F
G	P00.0	P00.1		P01.4	P01.3			VDD / VDDSB	VSS	VSS	VSS	VSS	VDD			nESR1	nPORST		P20.1	P20.3	G
H	P00.2	P00.3		P01.6	P01.5		VDD / VDDSB		VSS	VSS	VSS	VSS		VDD		P21.7	P21.6		P20.2	P20.0	H
J	P00.4	P00.5		P00.6	P01.7		VSS	VSS		VSS	VSS		VSS	VSS		TCK	P21.1		P21.3	P21.5	J
K	P00.7	P00.9		P00.8	P00.10		VSS	VSS	VSS	VSS	VSS	VSS	VSS	NC / VDDP		TMS	P21.0		P21.2	P21.4	K
L	P00.11	P00.12		AN43	AN42		VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS		P22.10	P22.11		nTRST	VSS	L
M	AN46	AN47		AN41	AN40		VSS	VSS		VSS	VSS		VSS	VSS		P22.8	P22.9		XTAL2	XTAL1	M
N	AN44	AN45		P40.6 AN36	P40.8 AN38		VDD		VSS	VSS	VSS	VSS		VDD		P22.6	P22.7		VDD	VDDP3	N
P	P40.9 AN39	P40.7 AN37		P40.4 AN32	AN34			VDD	VSS	VSS	VSS	VSS	VDD			P22.4	P22.5		P22.1	P22.0	P
R	P40.5 AN33	AN35		AN31	AN23											P23.7	P23.6		P22.3	P22.2	R
T	VAREF 2	VAGND 2		AN30	AN22	AN15	AN12	AN6	AN4	AN0	VEVRSE	P34.2	P34.4	P33.14	P32.5	VSS	P23.5		P23.3	P23.4	T
U	AN29	AN28		NC	AN17	AN14	AN9	AN7	AN3	AN1	P34.1	P34.3	P34.5	P33.15	P32.6	P32.7	VSS		P23.1	P23.2	U
V	P40.3 AN27	P40.2 AN26																	VEXT	P23.0	V
W	P40.1 AN25	P40.0 AN24	AN19	AN18	AN16	AN13	AN11	AN8	AN2	P33.0	P33.2	P33.4	P33.6	P33.8	P33.10	P33.12	VGATE 1P	P32.4	VSS	VEXT	W
Y	NC	AN21	AN20	VSSM	VDDM	VAREF 1	VAGND 1	AN10	AN5	P33.1	P33.3	P33.5	P33.7	P33.9	P33.11	P33.13	P32.0	P32.2	P32.3	VSS	Y
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	

Fonte: Infineon(2016)

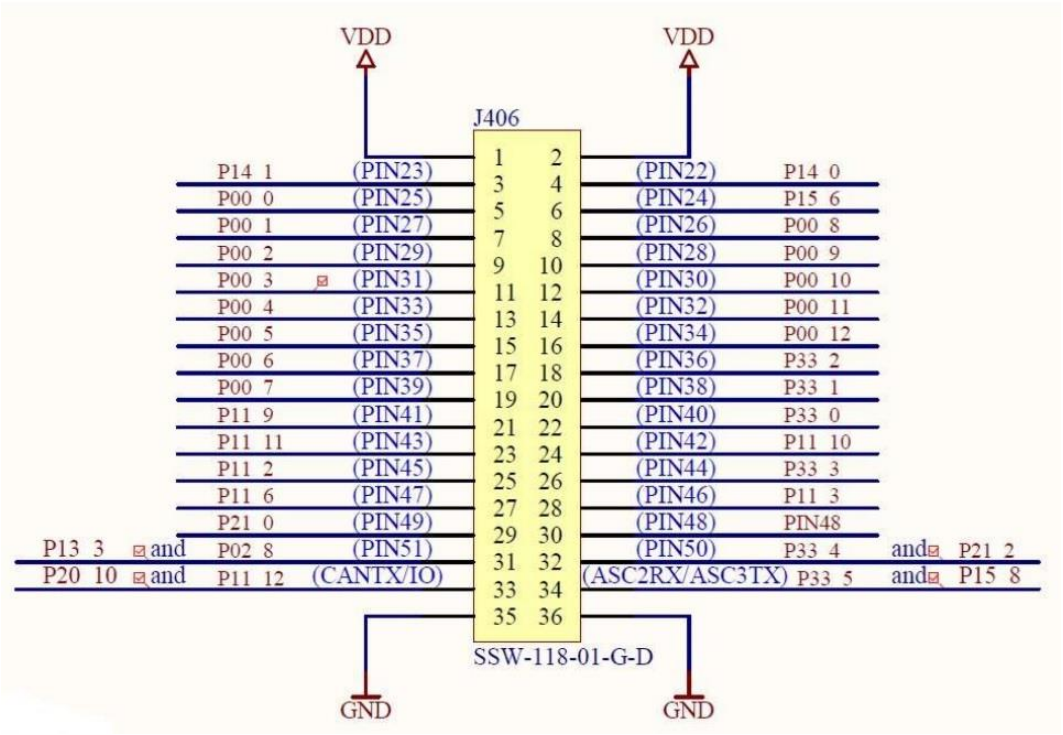
Através da documentação da placa ShieldBuddy foram localizados no conector de I/O (Figura 35) alguns pinos que poderiam dar acesso ao módulo GTM do microcontrolador.

Figura 35 - Layout da placa ShieldBuddy com destaque para o conector I/O utilizado para o sinal de entrada CKP



Fonte: Hitex(2016)

Figura 36 – Detalhe do conector de I/O e das portas do microcontrolador acessíveis através do conector.



Fonte: Hitex(2016)

Boa parte dos pinos disponíveis no conector de I/O da placa oferecem acesso direto às portas do microcontrolador, outros oferecem acesso a circuitos da própria placa ShieldBuddy. A partir da tabela de equivalência (Tabela 5) dos pinos do conector de I/O da placa ShieldBuddy com as portas do microcontrolador TC275 pode-se localizar os pinos que tem acesso ao módulo GTM.

Tabela 5 - Tabela de equivalência dos pinos do conector de I/O da placa ShieldBuddy com as portas do microcontrolador TC275

Arduino Signal Name	ShieldBuddy Connector Name	TC275T Pin Assignment
Digital pin 22	XIO.3	P14.0
Digital pin 23	XIO.4	P14.1
Digital pin 24	XIO.5	P15.6
Digital pin 25	XIO.6	P00.0
Digital pin 26	XIO.7	P00.8
Digital pin 27	XIO.8	P00.1
Digital pin 28	XIO.9	P00.9
Digital pin 29	XIO.10	P00.2
Digital pin 30	XIO.11	P00.10
Digital pin 31	XIO.12	P00.3
Digital pin 32	XIO.13	P00.11
Digital pin 33	XIO.14	P00.4
Digital pin 34	XIO.15	P00.12
Digital pin 35	XIO.16	P00.5
Digital pin 36	XIO.17	P33.2
Digital pin 37	XIO.18	P00.6
Digital pin 38	XIO.19	P33.1
Digital pin 39	XIO.20	P00.7
Digital pin 40	XIO.21	P33.2
Digital pin 41	XIO.22	P11.9
Digital pin 42	XIO.23	P11.10
Digital pin 43	XIO.24	P11.11
Digital pin 44 (PWM)	XIO.25	P33.3
Digital pin 45 (PWM)	XIO.26	P11.2
Digital pin 46 (PWM)	XIO.27	P11.3
Digital pin 47	XIO.28	P11.6
Digital pin 48	XIO.29	P21.3 (B-step) P21.1 (A-step) via link
Digital pin 49	XIO.30	P21.0
Digital pin 50 (MISO)	XIO.31	P33.4 + P21.3
Digital pin 51 (MOSI)	XIO.32	P2.8 + P13.3
Digital pin 52 (SCK)	XIO.33	P33.5 + P15.8
Digital pin 53 (SS)	XIO.34	P11.12 + P20.10

Fonte: Hitex(2016)

Devido a grande quantidade de módulos e recursos oferecidos pelo microcontrolador, a GPIO do microcontrolador possibilita a cada porta acessar apenas alguns destes módulos e submódulos. Além disso, a placa acessória GliwaShield é conectada ao conector de I/O da placa ShieldBuddy e utiliza boa parte dos sinais que ele oferece (Tabela 6, Tabela 7, Tabela 8, Tabela 9 e Tabela 10).

Tabela 6 – Portas utilizadas pelo Display da placa GliwaShield

Pin on LCD header	Description	AURduino Pin	Controller Pin
1	GND	-	-
2	VDD / +5V	-	-
3	VO / Contrast	-	-
4	RS (Register Select)	PIN26	P00.8
5	R/W (Read/Write)	PIN28	P00.9
6	EN (Enable) GND	PIN30	P00.10
7	DB0	PIN25	P00.0
8	DB1	PIN27	P00.1
9	DB2	PIN29	P00.2
10	DB3	PIN31	P00.3
11	DB4	PIN33	P00.4
12	DB5	PIN35	P00.5
13	DB6	PIN37	P00.6
14	DB7	PIN39	P00.7
15	Background light supply	-	-
16	Background light GND	-	-

Fonte: Gliwa(2016)

Tabela 7 – Portas utilizadas pelas duas interfaces CAN da placa GliwaShield

CAN Channel	Name	AURduino Pin	Controller Pin
CAN0	CAN TX	CANTX0	P20.8
CAN0	CAN RX	CANRX0	P20.7
CAN1	CAN TX	PIN22	P14.0
CAN1	CAN RX	PIN23	P14.1

Fonte: Gliwa(2016)

Tabela 8 – Portas utilizadas pelos LEDs da placa GliwaShield

LED	AURduino Pin	Controller Pin
V1	PIN47	P11.6
V2	PIN41	P11.9
V3	PIN42	P11.10
V4	PIN43	P11.11

Fonte: Gliwa(2016)

Tabela 9 – Portas utilizadas pelos botões da placa GliwaShield

Button	AURduino Pin	Controller Pin
S1	PIN40	P33.0
S2	PIN38	P33.1

Fonte: Gliwa(2016)

Tabela 10 – Portas utilizadas pelos controlador SDcard da interface GliwaShield

SD Pin	Description	AURduino Pin	Controller Pin	Pin Function
1	CS (Chip select)	CANTX/IO	P11.12 + P20.10	SLSO27
2	MOSI / MTSR	PIN51	PIN02.8 + P13.3	MTSR2P
4	VDD / +3.3V	-	-	-
5	SCK / Serial Clock	AD14/RXD3	P33.5 + P15.8	SCLK2
6	GND	-	-	-
7	MISO / MRST	PIN50	P33.4 + P21.3	MRST2CP
K2	Card Detect	PIN49	P21.0	-

Fonte: Gliwa(2016)

Através do cruzamento das informações foi constatado que a porta 15.6 do microcontrolador estava disponível no conector de I/O da placa ShieldBuddy através do seu pino de número 24, que não estava sendo utilizada por nenhum periférico da placa ShieldBuddy e nem da placa GliwaShield e que a mesma poderia ser direcionada (Tabela 11) para a entrada zero do submódulo TIM zero (TIM0_0) através da configuração apropriada dos registradores da GPIO.

Tabela 11 – Reprodução parcial da tabela de mapeamento das portas disponíveis para o módulo GTM do microcontrolador TC275 com encapsulamento padrão LQFP-176

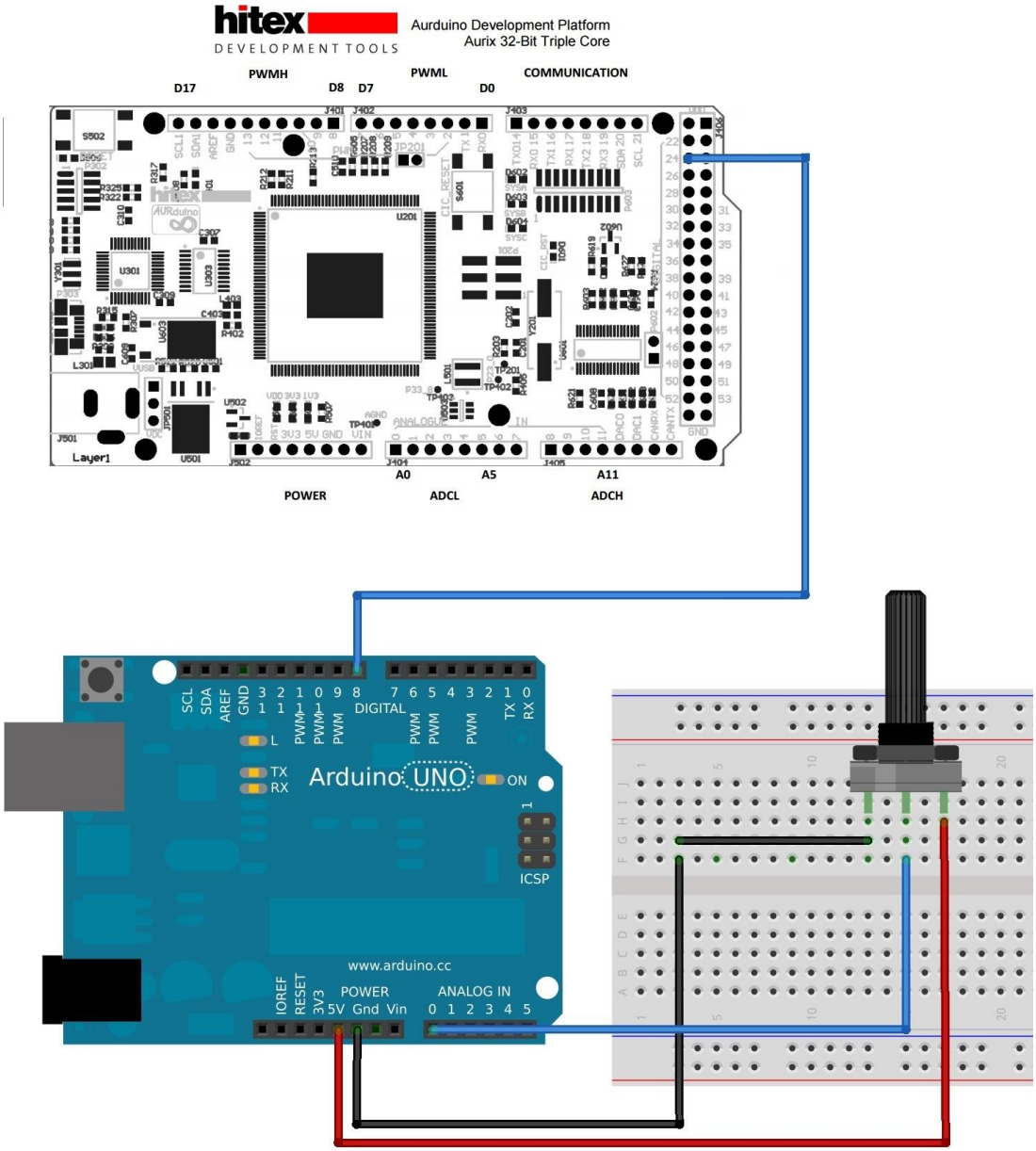
GTM to Port Mapping for QFP-176

Port	Input	Output	Input Timer Mapped		Output Timer Mapped			
			A	B	A	B	C	D
P15.0	TIN71	TOUT71	TIM2_3	TIM3_3	TOM1_3	TOM2_11	ATOM 1_3	ATOM 4_3
P15.1	TIN72	TOUT72	TIM2_4	TIM3_4	TOM1_4	TOM2_12	ATOM 1_4	ATOM 4_4
P15.2	TIN73	TOUT73	TIM2_5	TIM3_5	TOM1_5	TOM2_13	ATOM 1_5	ATOM 4_5
P15.3	TIN74	TOUT74	TIM2_6	TIM3_6	TOM1_6	TOM2_14	ATOM 1_6	ATOM 4_6
P15.4	TIN75	TOUT75	TIM2_7	TIM3_7	TOM1_7	TOM2_15	ATOM 1_7	ATOM 4_7
P15.5	TIN76	TOUT76	TIM2_0	TIM3_0	TOM0_0	TOM1_0	ATOM 1_0	ATOM 4_0
P15.6	TIN77	TOUT77	TIM0_0	TIM1_0	TOM0_0	TOM1_0	ATOM 1_0	ATOM 4_0
P15.7	TIN78	TOUT78	TIM0_1	TIM1_1	TOM0_1	TOM1_1	ATOM 1_1	ATOM 4_1

Fonte: Infineon, Adaptado pelo Autor(2016)

Sendo assim foi realizada a conexão (Figura 37) entre o Arduino utilizado para emular o sinal do sensor de rotação do motor CKP e o kit ATdemo utilizado como central de gerenciamento de motor (ECM).

Figura 37 – Conexão entre Arduino Uno e ShieldBuddy TC275 Hitex

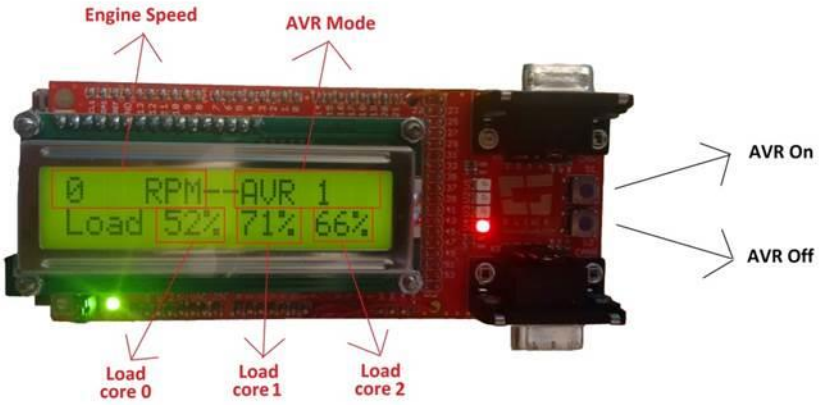


Fonte: Elaborado pelo Autor(2016)

Uma vez recebido o sinal do sensor CKP com sucesso, foi criada a rotina de tratamento de interrupção ISR e configurados os botões, LEDs e display.

Utilizando os recursos do GliwaShield, indispensável para realizar a analise temporal do sistema através da comunicação entre o host T1 e a placa ShieldBuddy através de sua interface CAN, foi possível desenvolver uma interface homem máquina (IHM) básica (Figura 38) mas extremamente funcional.

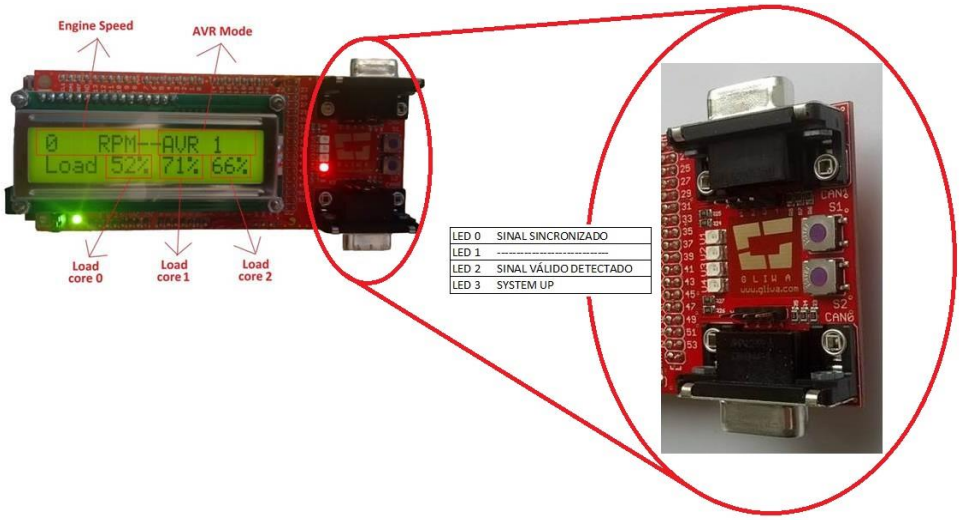
Figura 38 – Interface Gliwa Shield



Fonte: Elaborado pelo Autor(2016)

Para facilitar a operação do sistema e visualização do seu estado foram utilizados os recursos de entrada e saída da placa GliwaShield de forma que o display foi configurado para exibir as informações de velocidade do motor (RPM), modo AVR e carga nos três núcleos do microcontrolador, os LEDs foram programados para indicar a situação da aquisição do sinal do sensor CKP (Figura 39) e os botões foram configurados para ativar e desativar a função AVR, de forma que fosse possível realizar a analise temporal com AVR e sem AVR de forma mais prática.

Figura 39 – Detalhe do painel de LEDs, Botões S1 e S2 e Interfaces CAN0 e CAN1



Fonte: Elaborado pelo Autor(2016)

O LED 3 é acionado (Tabela 12) assim que o sistema operacional de tempo real (RTOS) é carregado e os módulos e submódulos internos do microcontrolador estão prontos para receber o sinal de entrada.

Assim que um sinal válido é detectado na entrada canal 0 do submódulo TIM_0 o LED 2 é acionado, indicando que o sinal do sensor CKP está sendo recebido.

Após a sincronização do sinal do sensor CKP com os circuitos internos do submódulo DPLL a tarefa AVR é habilitada e o LED 0 é acionado.

A partir deste momento a tarefa AVR é ativada sempre que a roda fônica atinge a posição programada.

Tabela 12 – Sinalização LEDs

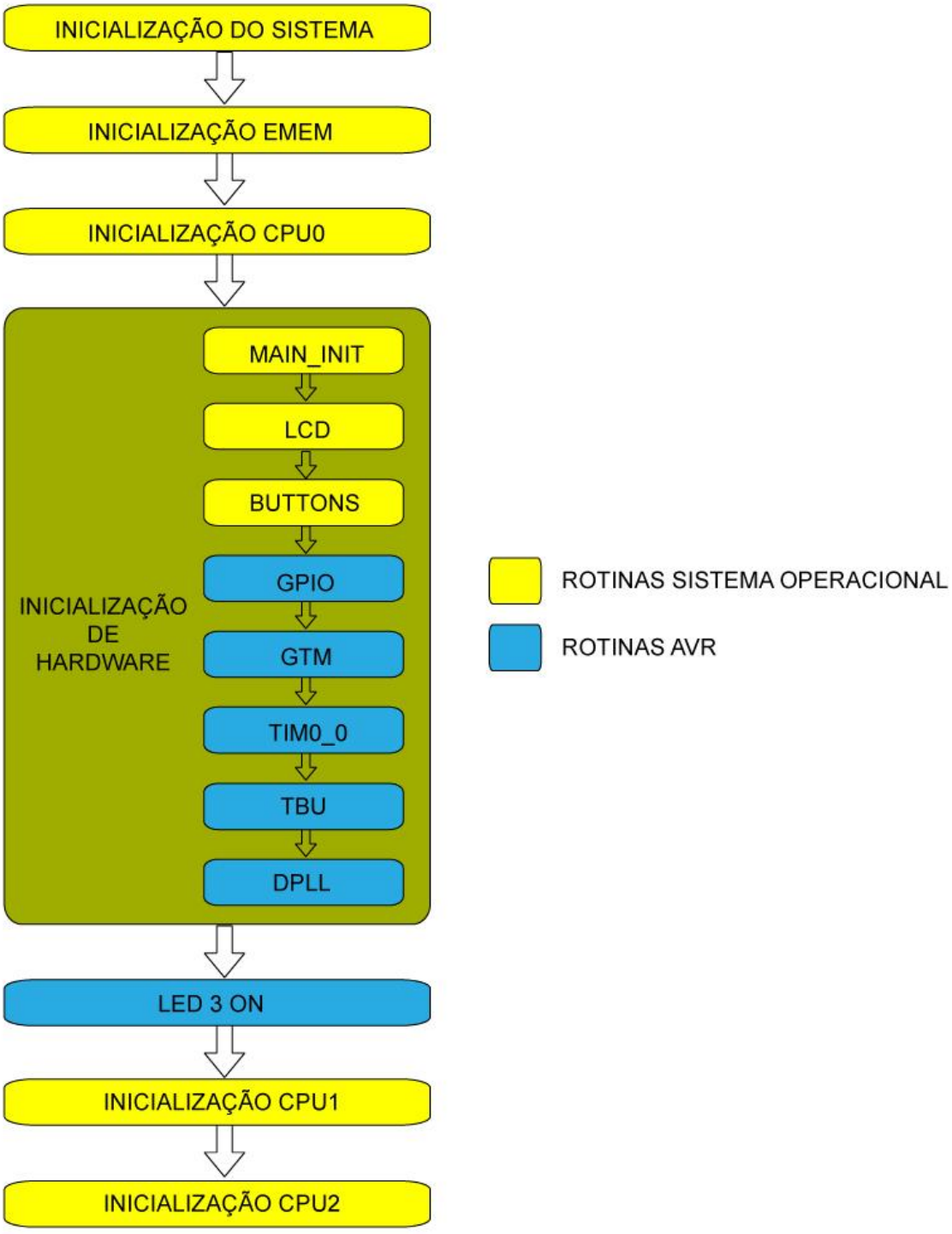
LED 0	SINAL SINCRONIZADO
LED 1	----------
LED 2	SINAL VÁLIDO DETECTADO
LED 3	SYSTEM UP

Fonte: Elaborado pelo Autor(2016)

6.1.3 Rotinas de configuração e funções desenvolvidas e integradas ao RTOS

Com o objetivo de configurar os módulos e submódulos do microcontrolador utilizados neste trabalho foram desenvolvidas rotinas de configuração que foram integradas ao código fonte do sistema operacional de tempo real GliwOS. No momento em que o microcontrolador é energizado o sistema operacional executa rotinas de inicialização especificas para cada elemento de hardware interno ou externo ao microcontrolador e também para os três núcleos de processamento. As rotinas necessárias para o funcionamento de tarefas AVR foram incluídas logo após a inicialização dos periféricos utilizados para o funcionamento da placa GliwaShield (Figura 40) e antes da inicialização dos núcleos de processamento 1 e 2.

Figura 40 - Sequência de inicialização do microcontrolador



Fonte: Elaborado pelo Autor(2016)

Após a sequência de inicialização o sistema já está apto a receber o sinal do sensor CKP e gerar as interrupções necessárias.

A rotina de tratamento de interrupções (ISR) definida para a tarefa AVR foi instalada na CPU2 () de forma que não houvesse sobrecarga nos núcleos de processamento, uma vez que a CPU0 foi programada para atender as interrupções das interfaces CAN e a instalação da tarefa AVR neste núcleo poderia afetar a comunicação entre o módulo alvo T1 (T1-TARGET-SW) e o módulo T1-HOST-SW no PC.

Para que o sinal gerado internamente no submódulo DPLL fosse sincronizado com o sinal do sensor CKP, o módulo DPLL precisa comparar o sinal de entrada com a imagem do sinal padrão armazenado em seus registradores.

Tabela 13 – Interrupções instaladas para viabilizar a tarefa AVR

Core 1 (CPU1)			
ORIGEM DA INTERRUPÇÃO	ISR	OBJETIVO	
SRC_ID_GTMTIM00	isr_toothdet	detectar sinal de entrada	
SRC_ID_GTMDPLL13	isr_lockdet	detectar sincronismo	GL1I
SRC_ID_GTMDPLL14	isr_locklost	detectar perda de sincronismo	LL1I
SRC_ID_GTMDPLL7	isr_mti	detectar dente faltante	MTI

Core 2 (CPU2)			
ORIGEM DA INTERRUPÇÃO	ISR	OBJETIVO	
SRC_ID_GTMDPLL18	isr_rpm	tarefa AVR	TE0I

Fonte: Elaborado pelo Autor(2016)

6.1.4 Sistema Operacional de Tempo Real

Foi utilizado o RTOS GliwOS, fornecido pela Gliwa Embedded Systems. Para realizar a analise temporal de todo o sistema é necessário que o sistema operacional seja instrumentado, ou seja, devem ser inseridas rotinas e funções que permitam o registro de características temporais de cada tarefa, executável, ISR, além de acesso a variáveis, ativação de tarefas além de outras características. O GliwOS fornecido em conjunto com o Kit ATdemo já é instrumentado e integrado ao módulo alvo T1 (T1-TARGET-SW), além disso foi fornecido o seu código fonte, tornando

possível não só a personalização de seus recursos e configurações, como também a inclusão de módulos de software desenvolvidos especificamente para este trabalho, como por exemplo o módulo AVR, que implementa as tarefas adaptativas de taxa de ativação variável.

6.1.4.1Tarefas Periódicas

. Dentre as principais características de um RTOS está o gerenciamento da execução de tarefas baseado em prioridades e períodos de ativação fixos ou variáveis.

Tabela 14 - Tarefas com escalonamento fixo em execução em cada um dos três núcleos

TAREFAS COM ESCALONAMENTO FIXO					
NÚCLEO	**TAREFA**	**EXECUTÁVEL**	**DELTA**	**CET**	**FUNÇÃO**
Core 0	Core0_1ms_Task	T1_AppHandler( );	1ms		handler T1
		T1_RxWheelSpeed( );			Simulação T1
		Core0_1msRunnable0( );			Executável fictício
	Core0_25msTask	Core0_25msRunnable0( );	25ms		Executável fictício
		Core0_25msRunnable1( );			Executável fictício
	Core0_50msTask	Core0_50msRunnable0( );	50ms		exibir informações no display.
Core 1	Core1_2msTask	T1_AppHandler( );	2ms		handler T1
		Core1_2msRunnable2( );			Executável fictício
	Core1_5msTask	Core1_5msRunnable0( );	5ms		Executável fictício
		Core1_5msRunnable1( );			Executável fictício
		Core1_5msRunnable2( );			Executável fictício
		Core1_5msRunnable3( );			Executável fictício
	Core1_10msTask	Core1_10msRunnable0( );	10ms		Executável fictício
		Core1_10msRunnable1( );			Executável fictício
		Core1_10msRunnable2( );			Executável fictício
		Core1_10msRunnable3( );			Executável fictício
		Core1_10msRunnable4( );			Executável fictício
		Core1_10msRunnable5( );			Executável fictício
		Core1_10msRunnable6( );			Executável fictício
		Core1_10msRunnable7( );			Executável fictício
		Core1_10msRunnable8( );			Executável fictício
		Core1_10msRunnable9( );			Executável fictício
		Core1_10msRunnable10( );			Executável fictício
		Core1_10msRunnable11( );			Executável fictício
		Core1_10msRunnable12( );			Executável fictício
		Core1_10msRunnable13( );			Executável fictício
		Core1_10msRunnable14( );			Executável fictício
		Core1_10msRunnable15( );			Executável fictício
	Core1_25msTask	Core1_25msRunnable0( );	25ms		Executável fictício
		Core1_25msRunnable1( );			exemplo de acesso a variáveis inter núcleos
Core 2	Core2_2msTask	T1_AppHandler( );	2ms		handler T1
		Core2_2msRunnable0( );			Executável fictício
	Core2_10msTask	Core2_10msRunnable0( );	10ms		Executável fictício
	Core2_25msTask	Core2_25msRunnable0( );	25ms		exemplo de acesso a variáveis inter núcleos
		Core2_25msRunnable1( );			exemplo de acesso a variáveis inter núcleos
		Core2_25msRunnable2( );			Executável fictício
		Core2_25msRunnable3( );			Executável fictício
		Core2_25msRunnable4( );			Executável fictício
		Core2_25msRunnable5( );			Executável fictício
		Core2_25msRunnable6( );			Executável fictício
		Core2_25msRunnable7( );			Executável fictício
		Core2_25msRunnable8( );			Executável fictício
		Core2_25msRunnable9( );			Executável fictício
	Core2_50msTask	Core2_50msRunnable1( );	50ms		calcular e atribuir variável do modo AVR

Fonte: Elaborado pelo Autor(2016)

As tarefas periódicas são aquelas configuradas no RTOS para serem executadas a cada determinado período de tempo, no caso analisado o sistema operacional foi configurado para executar tarefas com períodos de 1ms, 2ms, 25ms, 50ms e 70ms. Estes períodos fixos são utilizados como forma de manter a organização e facilidade de manutenção e desenvolvimento do sistema, pois a grande maioria das tarefas pode ser encaixada em cada um destes conjuntos de tarefas, porém caso seja necessário o RTOS pode ser configurado para executar tarefas com qualquer período desejado.

6.1.4.2Tarefas Periódicas Com Escalonamento Variável

As tarefas periódicas com escalonamento variável são tarefas cujo período pode variar de acordo com determinadas condições do sistema, por exemplo, determinada tarefa deve ser executada a cada 50ms em condições normais, mas em uma situação de falha de algum sensor do sistema ela deve ser executada a cada 25ms.

6.1.4.3 A Tarefa AVR

Foi configurada uma tarefa AVR com três modos de operação distintos. O aplicativo T1 conta com temporizadores (*delays*) que podem ser programados para simular funções que ainda estão em fase de desenvolvimento mas já se tem uma estimativa do valor de u (utilização computacional) que será utilizado por esta função. Desta forma pode-se criar funções fictícias que simulam a função real que ainda não foi desenvolvida de forma que se possa realizar a analise temporal do sistema completo antes mesmo de todas as funções de software estarem disponíveis. Para simular as funções utilizadas no três modos de operação da tarefa AVR foram definidas funções simuladas com u=400ms, 200ms e 80ms para cada modo respectivamente.

Tabela 15 – Tarefa AVR programada

Modo	C^m	$[\omega^{m-}, \omega^{m+}]$ (rpm)	Funcionalidade
1	400ms	[500, 2000]	f1()
2	200ms	[1500, 4000]	f2()
3	80ms	[3500, 6000]	f3()

Fonte: Elaborado pelo Autor(2016)

6.1.5 Integração do sistema

Para que o sinal do CKP gerado pelo arduino pudesse ser utilizado pelo microcontrolador era necessário localizar uma porta na placa ShieldBuddy TC275 que não fosse utilizada nem por ela mesma e nem pelo Gliwa shield, além disso esta porta deveria ter acesso ao canal 0 do "Timer Input Module – TIM" do microcontrolador, uma vez que a GPIO do microcontrolador não permite acesso pleno de todas as portas à todos os seus periféricos internos e somente este canal pode ser utilizado para as funções especificas de manipulação do sinal CKP imprescindíveis para o gerenciamento do motor.

Sendo assim, foi localizada a porta 24 do ShieldBuddy TC275 que por sua vez está conectada à porta 15.6 do microcontrolador. Assim a GPIO foi configurada para que a porta 15.6 fosse de entrada e o sinal ali presente fosse direcionado para o canal 0 da TIM0, podendo assim ser utilizado pelo sistema.

6.1.6 Ferramenta T1 e o Quadro Básico de Escalonamento (BSF)

O aplicativo T1 da Gliwa Embedded Systems monitora em tempo real a execução de tarefas, troca de informações entre os núcleos do microcontrolador e

carga na CPU, fornecendo subsídio para uma completa e detalhada analise temporal de todo o sistema.

Figura 41 - Barramentos USB e CAN independentes conectam o hardware de destino com o PC

Fonte: Gliwa, Adaptado pelo Autor(2016)

Podem ser gerados gráficos do tempo de execução das tarefas, interrupções, executáveis, funções, fragmentos de código, fluxo de dados entre os diferentes núcleos, ativações inter-núcleo, entre outras possibilidades.

Para realizar as medições de carga computacional na CPU a ferramenta T1 utiliza o conceito de “Quadro Básico de Escalonamento” (BSF). As tarefas gerenciadas pelo RTOS geralmente possuem escalonamento fixo e após certo período de tempo estas tarefas se repetem de forma cíclica. O menor período de tempo a partir do qual as tarefas se repetem, sempre com a mesma quantidade de ativações entre períodos subsequentes, é chamada de “Quadro Básico de Escalonamento” (Basic Scheduling Frame – BSF). Este período é determinado pelo mínimo múltiplo comum do período de todas as tarefas agendadas.

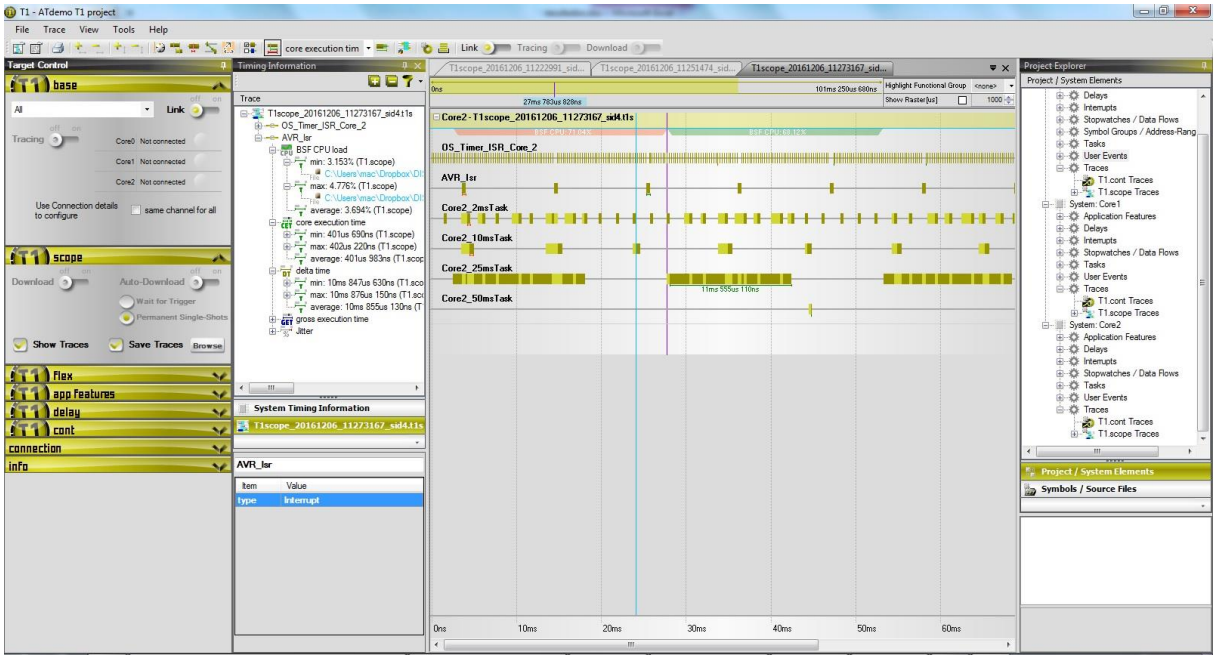
Desta forma, todas as medições realizadas pela ferramenta T1 são referentes a um quadro BSF.

6.2 COLETA DOS RESULTADOS

Através do valor da velocidade do motor (rpm) informado no *display* do kit ATdemo, ajusta-se o potenciômetro até se aproximar o máximo possível da velocidade desejada e faz-se o *download* de um rastreamento (*trace*).

A partir do rastreamento obtido pode-se observar as informações desejadas através dos diversos gráficos e relatórios disponíveis.

Figura 42 – Captura de tela da ferramenta T1 após o *download* de um rastreamento

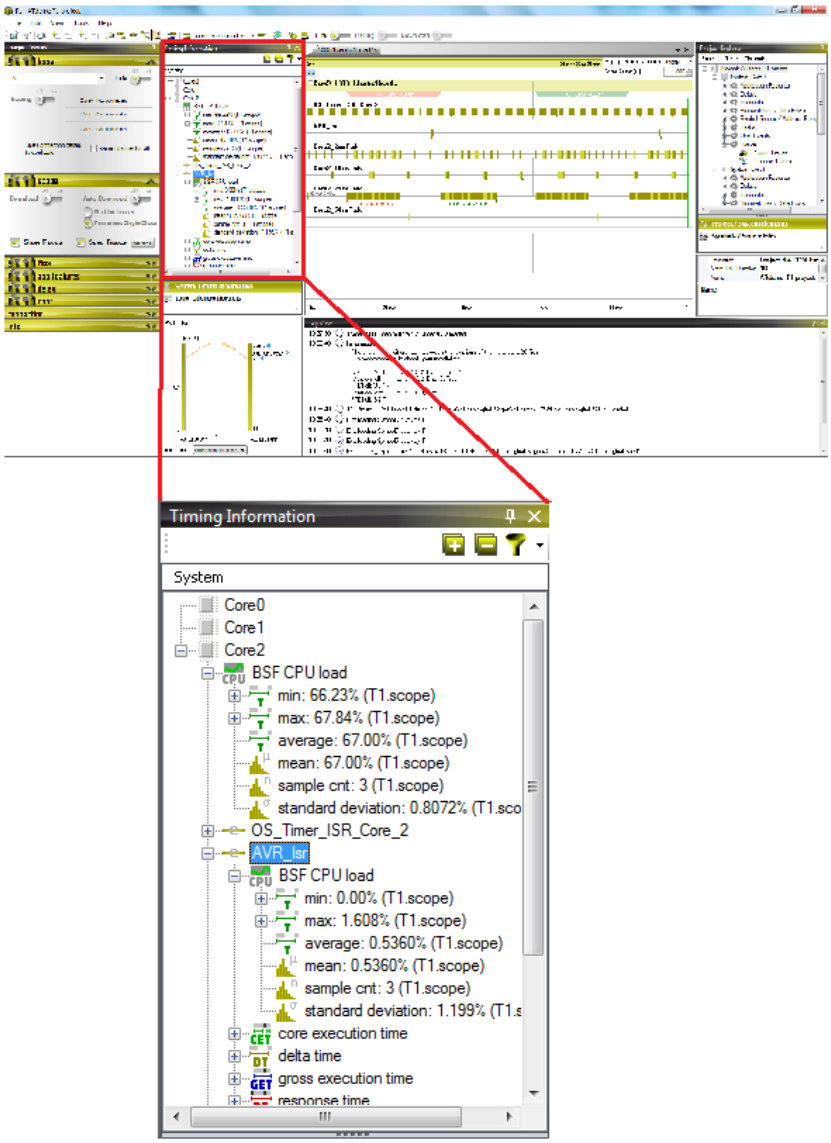


Fonte: Gliwa, Adaptado pelo Autor(2016)

As informações relevantes para a analise do impacto da tarefa AVR na carga do sistema são a utilização da CPU (u), a utilização da CPU pela tarefa AVR (u_i), o tempo computacional C_i utilizado pela tarefa AVR e o período $\boldsymbol{T_i}$ da tarefa AVR.

A ferramenta T1 oferece diversos caminhos para se obter as informações desejadas. Os valores de u, u_i , C_i e $\boldsymbol{T_i}$ podem ser obtidos diretamente da janela "*Timing Information*" (Figura 43), através dos gráficos de rastreamento (*traces*) e através de relatórios gerados em padrão html ou csv.

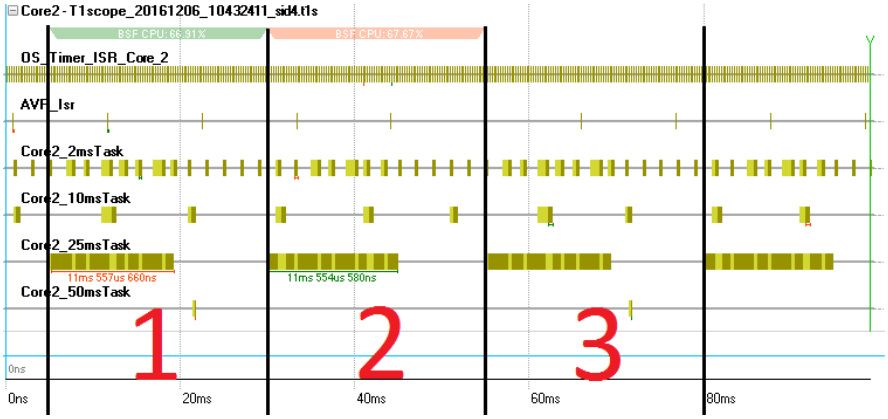
Figura 43 – Quadro “Timing Information” com informações temporais detalhadas



Fonte: Elaborado pelo Autor(2016)

O valor da utilização da CPU (u) obtido através do quadro “*Timing Information*” é denominado pela ferramenta T1 como “*BSF CPU load*”, o que significa que a carga da CPU foi medida com relação a um Quadro Básico de Escalonamento (BSF). Uma vez que todas as medições realizadas pela ferramenta T1 têm como base quadros BSF (Figura 44), e que em regime de operação a execução das tarefas nada mais é do que uma sequencia contínua de quadros BSF, cada medição realizada contém uma quantidade determinada de quadros BSF.

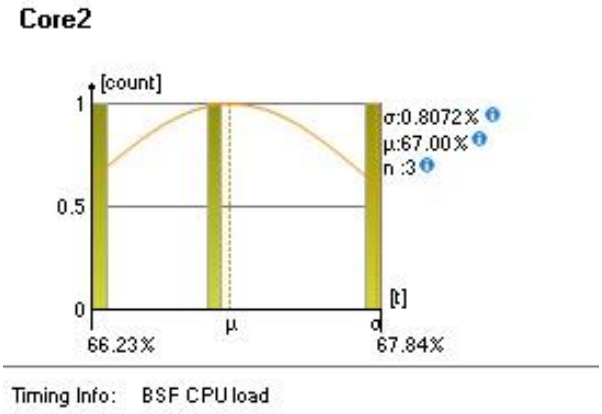
Figura 44 – Delimitação dos Quadros Basicos de Escalonamento (BSF)



Fonte: Elaborado pelo Autor(2016)

Assim sendo, a ferramenta T1 calcula tantos valores da utilização da CPU (u) quantos forem os quadros BSF medidos. De forma a melhor representar estes valores é feita uma analise estatística (Figura 45) onde se pode observar a média, a quantidade de amostras (quadros BSF), o desvio padrão, os valores mínimos, máximos e a quantidade de ocorrências de cada valor.

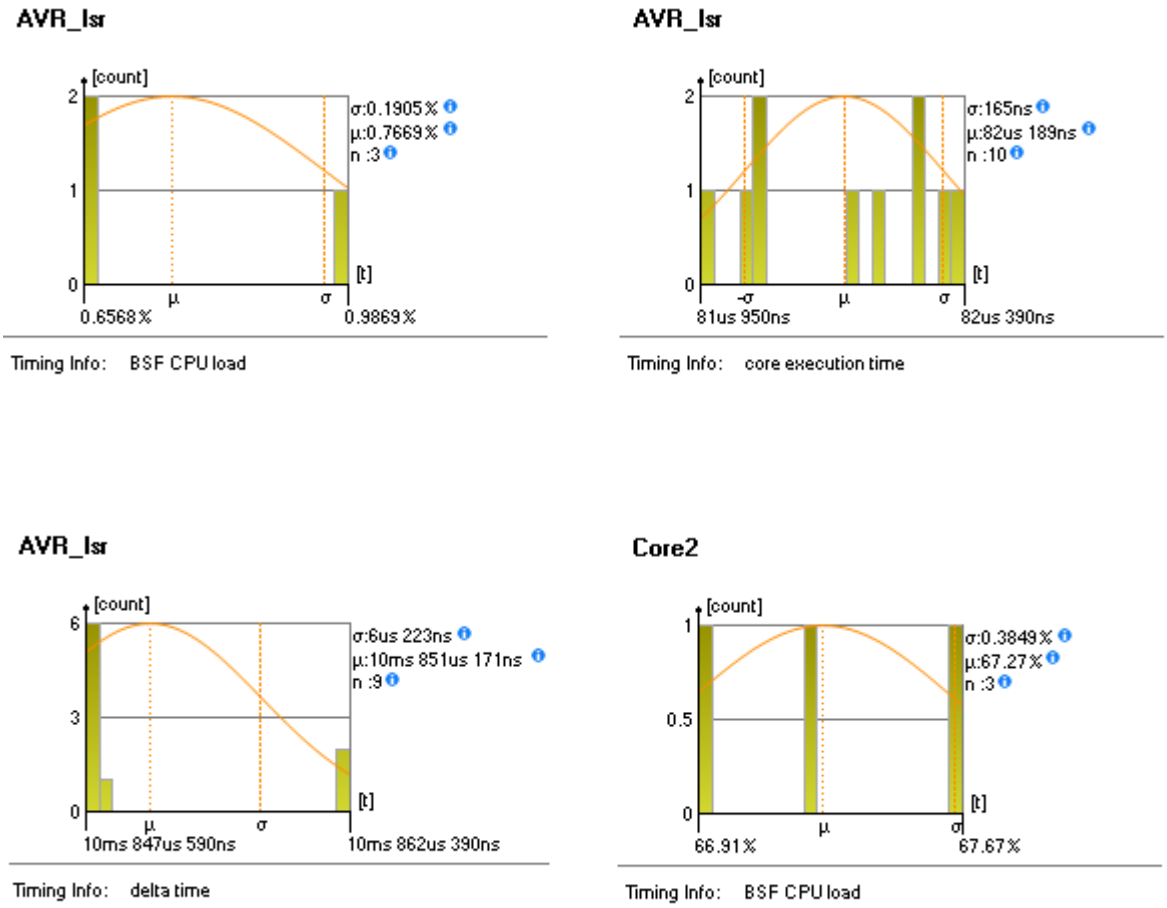
Figura 45 – Analise estatística dos valores de utilização da CPU2 do microcontrolador



Fonte: Gliwa T1, Adaptado pelo Autor(2016)

Os valores da utilização da CPU pela tarefa AVR (u_i), seu tempo computacional C_i e seu período $\boldsymbol{T_i}$ também podem ser obtidos através do quadro "*Timing Information*", dentro da arvore de valores referentes à tarefa AVR, ou através dos gráficos de analise estatística (Figura 46)

Figura 46 - Analise estatística dos valores medidos



Fonte: Gliwa T1, Adaptado pelo Autor(2016)

O período T_i é obtido através do valor de “delta time”, uma vez que a ferramenta T1 não pode identificar o período da tarefa AVR por ser uma tarefa com período variável, além de ser ativada sem a intervenção do RTOS, portanto não existindo informações de escalonamento disponíveis para que a ferramenta T1 calcule o seu período.

6.2.1 Confecção das Tabelas

Para a analise temporal realizada neste trabalho foi utilizado o valor médio das medições (Figura 46) na montagem das tabelas temporais.

Foram confeccionadas três tabelas. A primeira tabela, para o sistema com AVR habilitado e velocidade de rotação crescente (Tabela 16), foi confeccionada

iniciando em 500 rpm até atingir 6.000 rpm com incrementos de 500 rpm. Nas faixas de rotação limítrofes às alterações de modo de operação foi realizada uma medição com rotação 1 rpm inferior ao valor de comutação de modos e 1 rpm superior ao mesmo valor.

Tabela 16 – Tabela para velocidades crescentes com destaque para as faixas de rotação limítrofes entre os modos de operação AVR

VELOCIDADE (ω) CRESCENTE - AVR HABILITADO										
ω (rpm)	ω display (rpm)	u_AVR (%)			C (µs)	delta (ms)	u_CPU (%)			
		min	med	max			min	med	max	modo AVR
500										1
1000										1
1500										1
1999										1
2001										2
2500										2
3000										2
3500										2
3999										2
4001										3
4500										3
5000										3
5500										3
6000										3

Fonte: Elaborado pelo Autor(2016)

A segunda tabela, para o sistema com AVR habilitado e velocidade de rotação decrescente (Tabela 17), foi confeccionada iniciando em 6.000 rpm até atingir 500 rpm com decrementos de 500 rpm. Nas faixas de rotação limítrofes às alterações de modo de operação foi realizada uma medição com rotação 1 rpm inferior ao valor de comutação de modos e 1 rpm superior ao mesmo valor.

Tabela 17 – Tabela para velocidades decrescentes com destaque para as faixas de rotação limítrofes entre os modos de operação AVR

VELOCIDADE (ω) DECRESCENTE - AVR HABILITADO										
ω (rpm)	ω display (rpm)	u_AVR (%)			C (µs)	delta (ms)	u_CPU (%)			
		min	med	max			min	med	max	modo AVR
6000										3
5500										3
5000										3
4500										3
4000										3
3501										3
3499										2
3000										2
2500										2
2000										2
1501										2
1499										1
1000										1
500										1

Fonte: Elaborado pelo Autor(2016)

A terceira tabela, para o sistema com AVR desabilitado e velocidade de rotação crescente, foi confeccionada iniciando em 500 rpm até atingir 6000 rpm com incrementos de 500 rpm.

Tabela 18 – Tabela para velocidades crescentes com AVR desabilitado

VELOCIDADE (ω) CRESCENTE - AVR DESABILITADO										
ω (rpm)	ω display (rpm)	u_AVR (%)			C (µs)	delta (ms)	u_CPU (%)			
		min	med	max			min	med	max	modo AVR
500										-
1000										-
1500										-
2000										-
2500										-
3000										-
3500										-
4000										-
4500										-
5000										-
5500										-
6000										-

Fonte: Elaborado pelo Autor(2016)

Não foram feitas medições para velocidades decrescentes com AVR desabilitado, pois como neste modo de operação não há alteração no comportamento das tarefas, não haveria diferenças entre os valores medidos com velocidade de rotação crescente e os valores medidos com velocidade de rotação decrescente.

7 RESULTADOS

7.1 ANALISE TEMPORAL DA TAREFA ANGULAR COM AVR HABILITADO E VELOCIDADE DO MOTOR CRESCENTE (ACELERAÇÃO POSITIVA)

Tabela 19 – Valores medidos com AVR e aceleração positiva

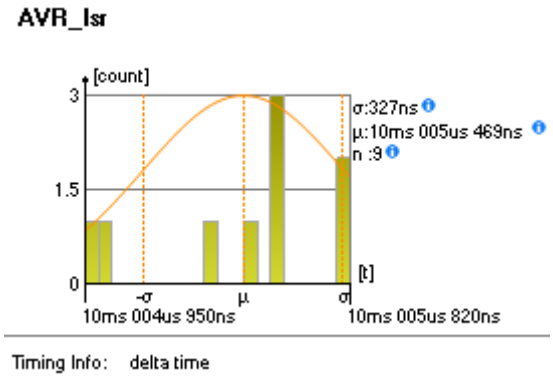
VELOCIDADE (ω) CRESCENTE - AVR HABILITADO										
ω (rpm)	ω display (rpm)	u_AVR (%)			C (µs)	delta (ms)	u_CPU (%)			modo AVR
		min	med	max			min	med	max	
500	497	0	0,5358	1,608	401,86	-	66,34	67,05	67,9	1
1000	995	0	0,536	1,608	401,945	60,278	66,23	67	67,84	1
1500	1503	0	1,073	1,61	402,42	39,791	66,3	67,59	65,86	1
1999	1951	0	1,072	1,609	402,197	30,737	66,96	67,81	68,54	1
2001	2006	0	0,5391	0,809	202,227	29,894	66,22	67	67,75	2
2500	2502	0,8076	0,8081	0,8088	202,042	23,988	67,11	67,54	67,75	2
3000	3016	0,8078	1,078	1,619	202,208	19,899	67,75	67,81	67,94	2
3500	3502	0,8083	1,078	1,617	202,168	17,123	67,13	67,61	68,56	2
3999	3954	0,8074	1,349	1,62	202,257	15,195	67,76	68,09	68,56	2
4001	4010	0,3286	0,5472	0,6572	82,075	14,949	66,58	67,32	67,69	3
4500	4528	0,6584	0,6587	0,6594	82,268	13,259	66,89	67,41	67,67	3
5000	4978	0,6575	0,7671	0,9859	82,216	12,058	66,9	67,53	68,01	3
5500	5530	0,6568	0,7669	0,9869	82,189	10,851	66,91	67,27	67,67	3
6000	5998	0,6569	0,7669	0,9853	82,125	10,005	67,22	67,52	67,67	3

Fonte: Elaborado pelo Autor(2016)

7.1.1 Validação do valor da velocidade do motor

Para validar a velocidade do motor (rpm) informada no *display* do kit ATdemo e ajustada através do potenciômetro foi utilizado o valor médio (Figura 47) de *delta* medido pela ferramenta T1.

Figura 47 – Período entre ativações consecutivas da tarefa AVR a 6000 rpm



Fonte: Gliwa T1, Adaptado pelo Autor(2016)

Através do valor do período a cada faixa de rotação foi calculada a velocidade real do motor e comparada ao valor exibido no display e utilizado para ajustar o potenciômetro.

Tabela 20 – Validação da velocidade do motor

ω display (rpm)	delta (ms)	ω calculado (rpm)	diferença (rpm)	diferença (%)
497	-	-	-	-
995	60,278	995,4	0,4	0,04
1503	39,791	1507,9	4,9	0,32
1951	30,737	1952,0	1,0	0,05
2006	29,894	2007,1	1,1	0,05
2502	23,988	2501,3	-0,7	-0,03
3016	19,899	3015,2	-0,8	-0,03
3502	17,123	3504,1	2,1	0,06
3954	15,195	3948,7	-5,3	-0,14
4010	14,949	4013,6	3,6	0,09
4528	13,259	4525,2	-2,8	-0,06
4978	12,058	4975,9	-2,1	-0,04
5530	10,851	5529,4	-0,6	-0,01
5998	10,005	5997,0	-1,0	-0,02

Fonte: Elaborado pelo Autor(2016)

A diferença percentual entre o valor exibido no display e o valor calculado em função do período medido ficou muito próxima de zero e demonstrou a confiabilidade dos valores exibidos no display e ajustados para a montagem das tabelas.

7.1.2 Validação do Custo computacional da tarefa AVR em função da velocidade de rotação do motor ω

O custo computacional medido pela ferramenta T1 (Tabela 21) foi totalmente compatível com os valores teóricos definidos para os três modos de operação AVR. Nota-se que em faixas de rotação limítrofes ocorre uma grande variação no custo computacional para uma pequena variação de rotação do motor, o que é determinado pela comutação entre modos de operação AVR distintos.

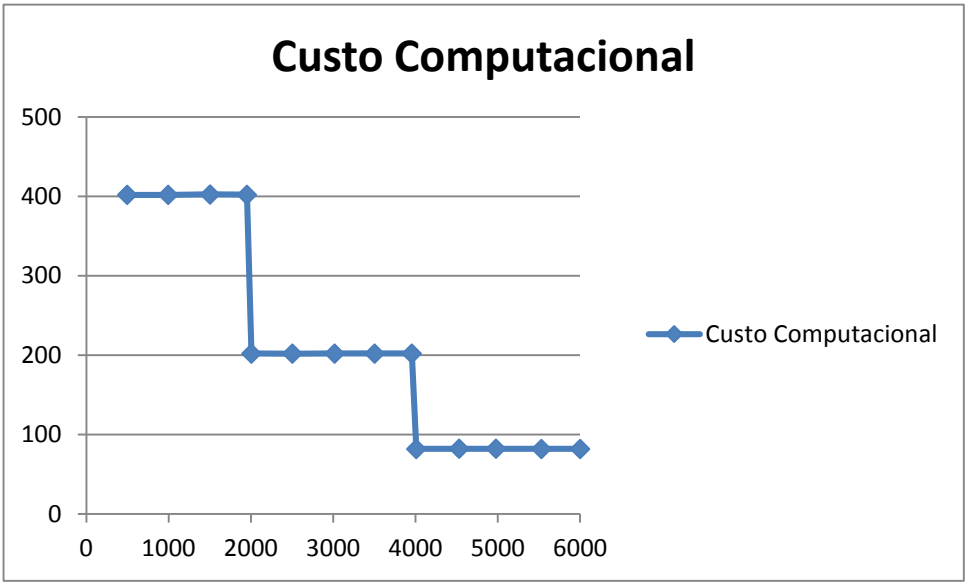
Tabela 21 - Custo Computacional (CET) da tarefa AVR medido pela ferramenta T1

VELOCIDADE (ω) CRESCENTE - AVR HABILITADO			
ω (rpm)	ω display (rpm)	C (µs)	modo AVR
500	497	401,86	1
1000	995	401,945	1
1500	1503	402,42	1
1999	1951	402,197	1
2001	2006	202,227	2
2500	2502	202,042	2
3000	3016	202,208	2
3500	3502	202,168	2
3999	3954	202,257	2
4001	4010	82,075	3
4500	4528	82,268	3
5000	4978	82,216	3
5500	5530	82,189	3
6000	5998	82,125	3

Fonte: Elaborado pelo Autor(2016)

Os valores do tempo de execução da tarefa AVR (CET) foram reduzidos nas rotações determinadas para tal, a medida que a rotação do motor aumenta (aceleração positiva)

Figura 48 – Custo computacional da tarefa AVR para aceleração positiva



Fonte: Elaborado pelo Autor(2016)

7.1.3 Validação da utilização da CPU u pela tarefa AVR em função da velocidade de rotação do motor ω

Para o valor da utilização u do processador pela tarefa AVR foram registrados os valores mínimo, médio e máximo de u dentre os diversos Quadros Básicos de Escalonamento (BSF) medidos.

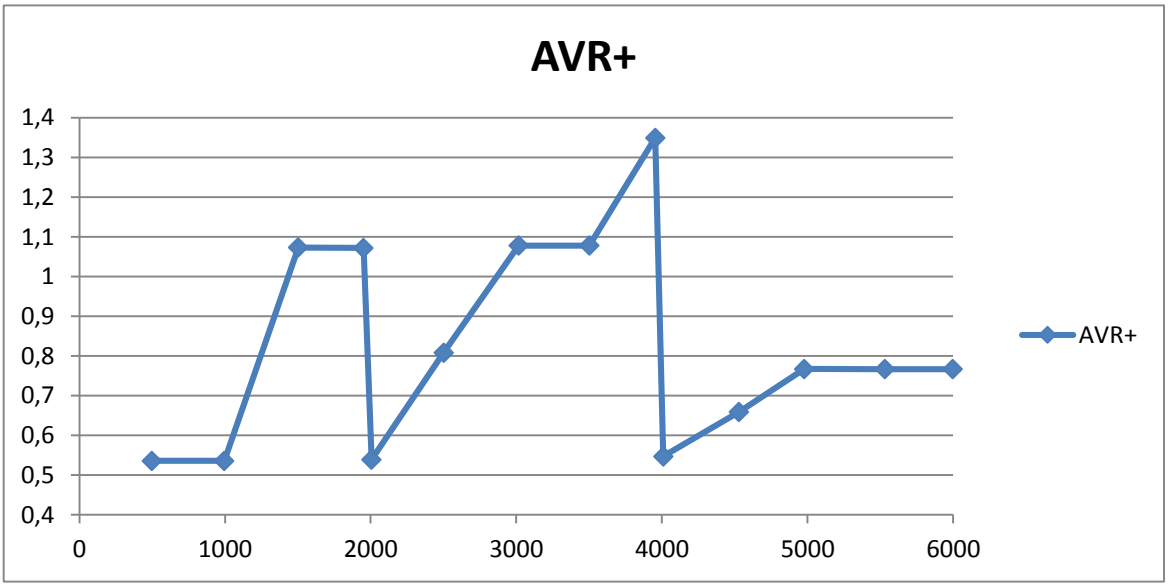
Tabela 22 – Valores de u da tarefa AVR

VELOCIDADE (ω) CRESCENTE - AVR HABILITADO					
ω (rpm)	ω display (rpm)	u_AVR (%)			modo AVR
		min	med	max	
500	497	0	0,5358	1,608	1
1000	995	0	0,536	1,608	1
1500	1503	0	1,073	1,61	1
1999	1951	0	1,072	1,609	1
2001	2006	0	0,5391	0,809	2
2500	2502	0,8076	0,8081	0,8088	2
3000	3016	0,8078	1,078	1,619	2
3500	3502	0,8083	1,078	1,617	2
3999	3954	0,8074	1,349	1,62	2
4001	4010	0,3286	0,5472	0,6572	3
4500	4528	0,6584	0,6587	0,6594	3
5000	4978	0,6575	0,7671	0,9859	3
5500	5530	0,6568	0,7669	0,9869	3
6000	5998	0,6569	0,7669	0,9853	3

Fonte: Elaborado pelo Autor(2016)

Nota-se certo padrão anômalo nos valores mínimos e máximos (Tabela 22), com sequencias de valores praticamente iguais para faixas de rotação distintas.

Figura 49 – Valores médios de u da tarefa AVR

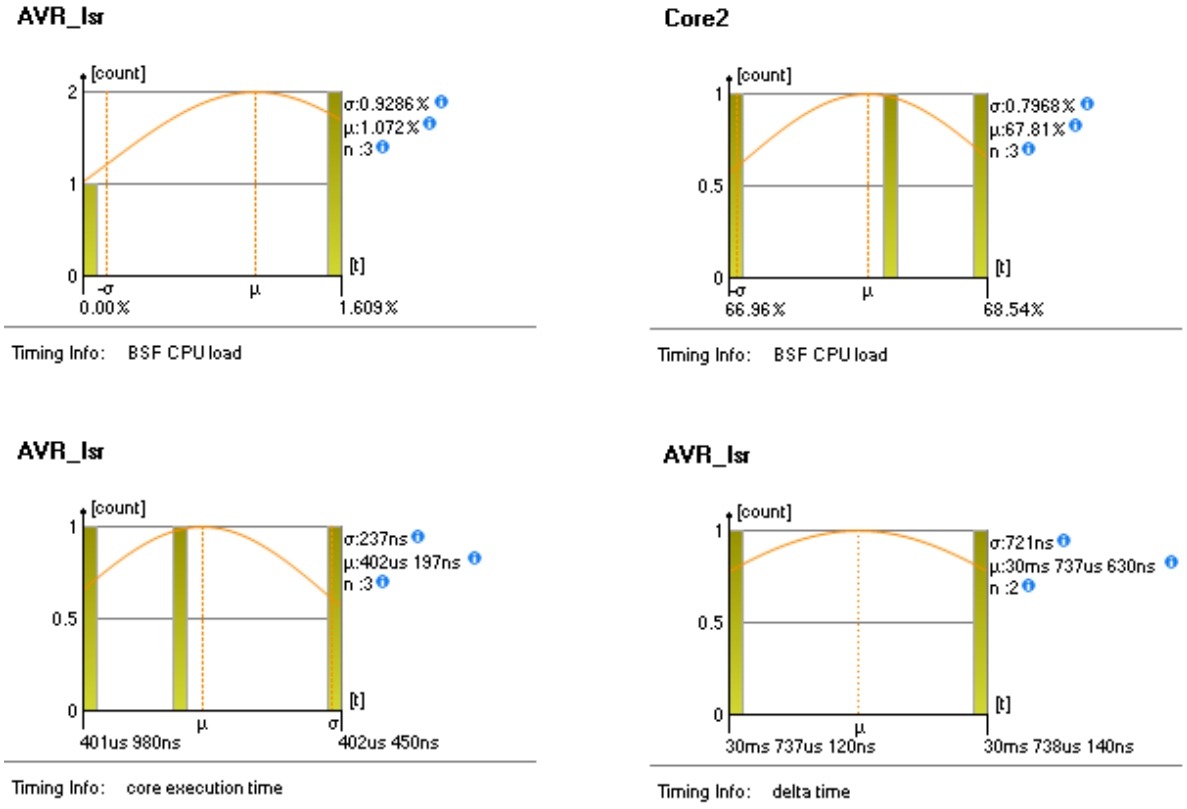


Fonte: Elaborado pelo Autor(2016)

Utilizaram-se os valores médios de u para o traçado dos gráficos que demonstrariam a funcionalidade da tarefa AVR de acordo com o comportamento demonstrado na definição das tarefas AVR (Figura 17), na qual a utilização da CPU é crescente até que haja comutação entre modos, onde ocorre uma redução abrupta no seu valor e inicia-se outra sequencia crescente de valores.

O gráfico obtido apresentou as quedas abruptas no valor de u quando da comutação entre modos e também apresentou comportamento crescente em faixas de rotação subsequentes dentro de um mesmo modo, porém ocorreram distorções não justificáveis, o que ensejou um olhar mais aprofundado nos valores medidos.

Figura 50 – Valores medidos para 2000rpm em modo AVR 1



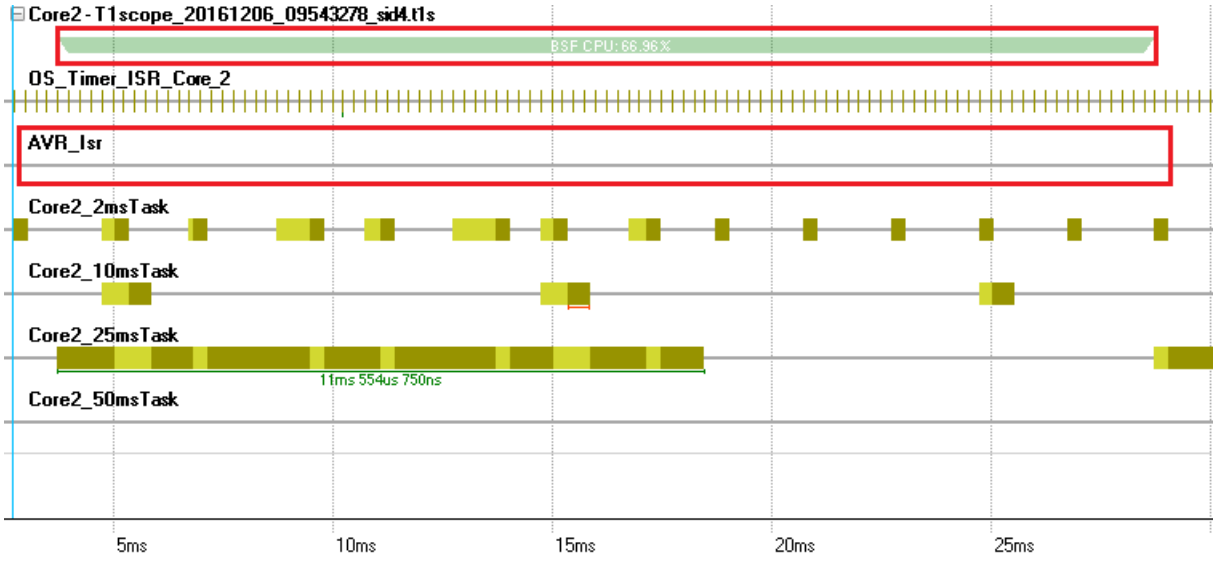
Fonte: Gliwa T1, Adaptado pelo Autor(2016)

Ao observar os valores mínimos e máximos de u, u_i , C_i e $\boldsymbol{T_i}$ nota-se que apenas os valores da utilização da tarefa AVR u_i possuem valor significativo de desvio padrão σ, o que reflete em valores mínimos e máximos bastantes distintos.

Ocorre que, para realizar as medições temporais a ferramenta T1 precisa determinar o Quadro Básico de Escalonamento (BSF), ou seja, a faixa de tempo onde serão realizados os cálculos necessários para determinar os valores de u, u_i , C_i e $\boldsymbol{T_i}$. Geralmente, todas as tarefas de um sistema embarcado são periódicas e sua execução é escalonada pelo sistema operacional de tempo real, porém este não é o caso da tarefa AVR. A tarefa AVR não tem um período fixo e, portanto, não é levada em consideração para o cálculo do quadro BSF.

O diagrama (Figura 51) do BSF que gerou a medição de 0% de utilização da CPU pela tarefa AVR u_i deixa claro o motivo deste valor. Dentro do quadro BSF utilizado para realizar a medição dos valores não ocorreu nenhuma ativação da tarefa AVR.

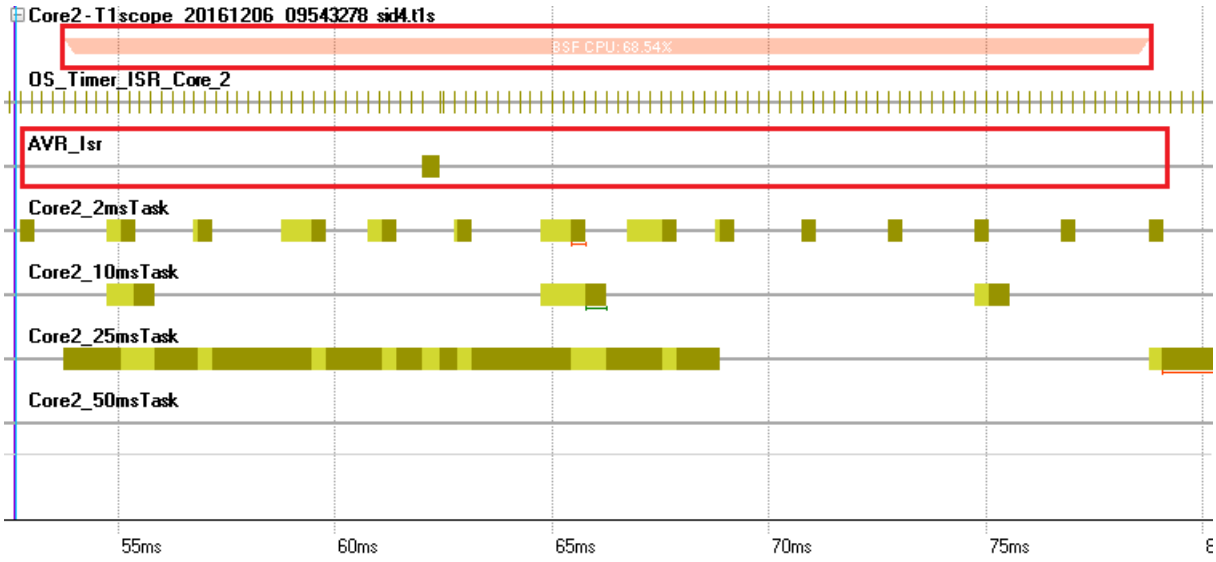
Figura 51 – "*Trace*" com destaque para a delimitação do quadro BSF e da tarefa AVR



Fonte: Gliwa T1, Adaptado pelo Autor(2016)

Já o diagrama (Figura 52) do BSF que gerou a medição de 1,609% de utilização da CPU pela tarefa AVR u_i mostra que ocorreu uma única ativação da mesma.

Figura 52 - "*Trace*" com destaque para a delimitação do quadro BSF e da tarefa AVR



Fonte: Gliwa T1, Adaptado pelo Autor(2016)

Sendo assim os valores da utilização u_i da tarefa AVR medidos pela ferramenta T1 não podem ser utilizados diretamente, pois em não fazendo parte do calculo do quadro BSF, os valores lidos perdem a validade, o que não ocorre com os valores do período "delta" e do Custo Computacional C, pois o conceito de quadro BSF só é necessário para os cálculos que envolvem carga na CPU.

7.1.3.1Fator de correção para os valores de utilização u_i

De forma a validar a implementação da tarefa AVR utilizando-se os dados medidos pela ferramenta T1 determinou-se um fator de correção para o valor de u_i . Ao observar-se o diagrama (Figura 52) do quadro BSF que gerou o valor de 1,609% de utilização da CPU pela tarefa AVR u_i nota-se apenas uma ocorrência da tarefa AVR, ou seja, o valor de utilização da CPU em cada ativação da tarefa AVR a 2000 rpm em modo AVR 1 é de 1,609%. Desta forma é possível estabelecer um fator de correção conforme segue:

Considerando:

T_{BSF} - Período do quadro BSF.

T_i- Período da tarefa AVR.

u_{AVR} - Utilização máxima da tarefa AVR medida pela ferramenta T1

n - quantidade de ativações dentro do quadro BSF que gerou o valor u_{AVR}

u_i - Valor corrigido da Utilização da tarefa AVR.

Tem-se que:

$$u_i = \frac{T_{BSF}}{T_i} . \frac{u_{AVR}}{n} \quad (7.1)$$

Aplicando-se o fator de correção tem-se os valores corrigidos da utilização do processador u_i pela tarefa AVR (Tabela 23). Destacado em vermelho, o valor do período para a velocidade de 500 rpm, devido ao fato de só haver uma ativação da tarefa AVR para esta faixa de rotação, portanto, não havendo valor medido de delta, foi utilizado o valor calculado do período de ativação para 500 rpm.

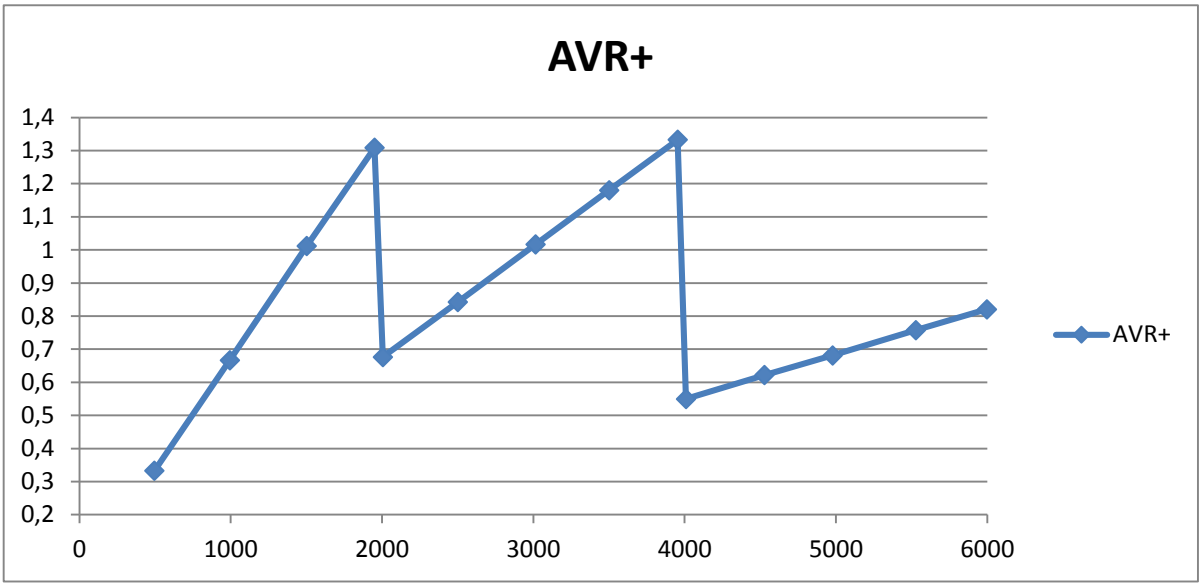
Tabela 23 – Valores corrigidos da utilização do processador pela tarefa AVR

VELOCIDADE (ω) CRESCENTE - AVR HABILITADO							
ω (rpm)	ω display (rpm)	u_AVR (%)	n	Tbsf	delta (ms)	T_BSF/T_AVR	u_AVR corrigido(%)
		max					
500	497	1,608	1	25	120,7	0,207125104	0,333057167
1000	995	1,608	1	25	60,278	0,414745015	0,666909984
1500	1503	1,61	1	25	39,791	0,628282778	1,011535272
1999	1951	1,609	1	25	30,737	0,813351986	1,308683346
2001	2006	0,809	1	25	29,894	0,836288218	0,676557169
2500	2502	0,8088	1	25	23,988	1,042187761	0,842921461
3000	3016	1,619	2	25	19,899	1,25634454	1,017010905
3500	3502	1,617	2	25	17,123	1,460024528	1,180429831
3999	3954	1,62	2	25	15,195	1,645278052	1,332675222
4001	4010	0,6572	2	25	14,949	1,672352666	0,549535086
4500	4528	0,6594	2	25	13,259	1,885511728	0,621653217
5000	4978	0,9859	3	25	12,058	2,073312324	0,68135954
5500	5530	0,9869	3	25	10,851	2,303935121	0,757917857
6000	5998	0,9853	3	25	10,005	2,498750625	0,820672997

Fonte: Elaborado pelo Autor(2016)

Traçando-se o gráfico dos valores medidos pela ferramenta T1 e corrigidos pelo fator de correção do Quadro Básico de Escalonamento tem-se o resultado esperado.

Figura 53 - Valores corrigidos de u da tarefa AVR



Fonte: Elaborado pelo Autor(2016)

7.2 ANALISE TEMPORAL DA TAREFA ANGULAR COM AVR HABILITADO E VELOCIDADE DO MOTOR DECRESCENTE (ACELERAÇÃO NEGATIVA)

Tabela 24 – Valores medidos com AVR e aceleração negativa

VELOCIDADE (ω) DECRESCENTE - AVR HABILITADO										
ω (rpm)	ω display (rpm)	u_AVR (%)			C (µs)	delta (ms)	u_CPU (%)			modo AVR
		min	med	max			min	med	max	
6000	5998	0,6569	0,7669	0,9853	82,125	10,005	67,22	67,52	67,67	3
5500	5530	0,6568	0,7669	0,9869	82,189	10,851	66,91	67,27	67,67	3
5000	4978	0,6575	0,7671	0,9859	82,216	12,058	66,9	67,53	68,01	3
4500	4528	0,6584	0,6587	0,6594	82,268	13,259	66,89	67,41	67,67	3
4000	4010	0,3286	0,5472	0,6572	82,075	14,949	66,58	67,32	67,69	3
3501	3558	0,3275	0,4384	0,6584	82,225	16,882	66,57	66,94	67,67	3
3499	3502	0,8083	1,078	1,617	202,168	17,123	67,13	67,61	68,56	2
3000	3016	0,8078	1,078	1,619	202,208	19,899	67,75	67,81	67,94	2
2500	2502	0,8076	0,8081	0,8088	202,042	23,988	67,11	67,54	67,75	2
2000	2006	0	0,5391	0,809	202,227	29,894	66,22	67	67,75	2
1501	1540	0	0,5393	0,8091	202,27	38,94	66,95	67,27	67,74	2
1499	1503	0	1,073	1,61	402,42	39,791	66,3	67,59	65,86	1
1000	995	0	0,536	1,608	401,945	60,278	66,23	67	67,84	1
500	497	0	0,5358	1,608	401,86	-	66,34	67,05	67,9	1

Fonte: Elaborado pelo Autor(2016)

7.2.1 Validação do Custo computacional da tarefa AVR em função da velocidade de rotação do motor ω

O custo computacional medido pela ferramenta T1 (Tabela 25) foi totalmente compatível com os valores teóricos definidos para os três modos de operação AVR. Nota-se que em faixas de rotação limítrofes ocorre uma grande variação no custo computacional para uma pequena variação de rotação do motor, o que é determinado pela comutação entre modos de operação AVR distintos.

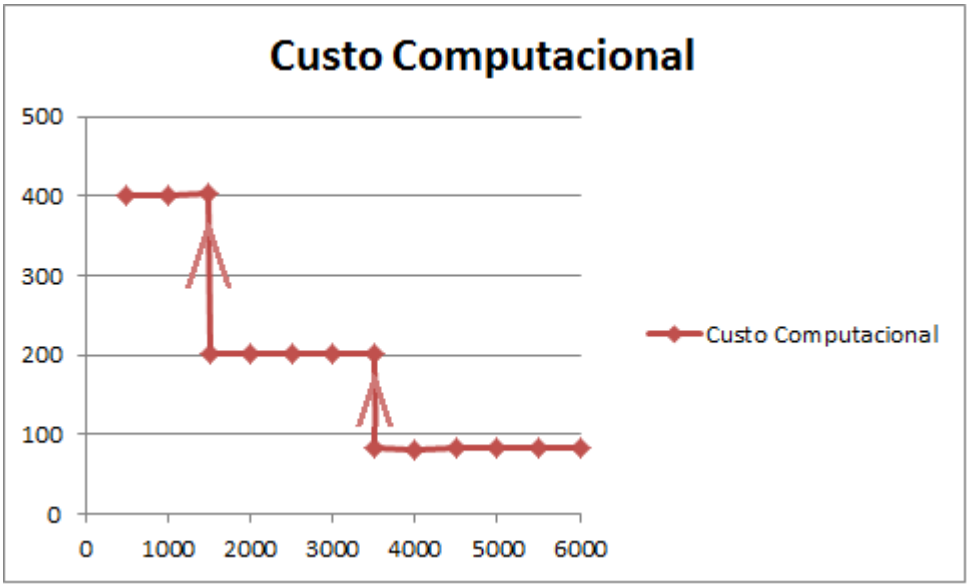
Tabela 25 - Custo Computacional (CET) da tarefa AVR medido pela ferramenta T1

VELOCIDADE (ω) DECRESCENTE - AVR HABILITADO			
ω (rpm)	ω display (rpm)	C (µs)	modo AVR
6000	5998	82,125	3
5500	5530	82,189	3
5000	4978	82,216	3
4500	4528	82,268	3
4000	4010	82,075	3
3501	3558	82,225	3
3499	3502	202,168	2
3000	3016	202,208	2
2500	2502	202,042	2
2000	2006	202,227	2
1501	1540	202,27	2
1499	1503	402,42	1
1000	995	401,945	1
500	497	401,86	1

Fonte: Elaborado pelo Autor(2016)

Os valores do tempo de execução da tarefa AVR (CET) foram aumentados nas rotações determinadas para tal, a medida que a rotação do motor diminui (aceleração negativa).

Figura 54 – Custo computacional da tarefa AVR para aceleração positiva



Fonte: Elaborado pelo Autor(2016)

7.2.2 Validação da utilização da CPU u pela tarefa AVR em função da velocidade de rotação do motor ω

Aplicando-se o fator de correção ((7.1) de forma a validar a implementação da tarefa AVR utilizando-se os dados medidos pela ferramenta T1 tem-se os valores corrigidos da utilização do processador u_i pela tarefa AVR (Tabela 26)

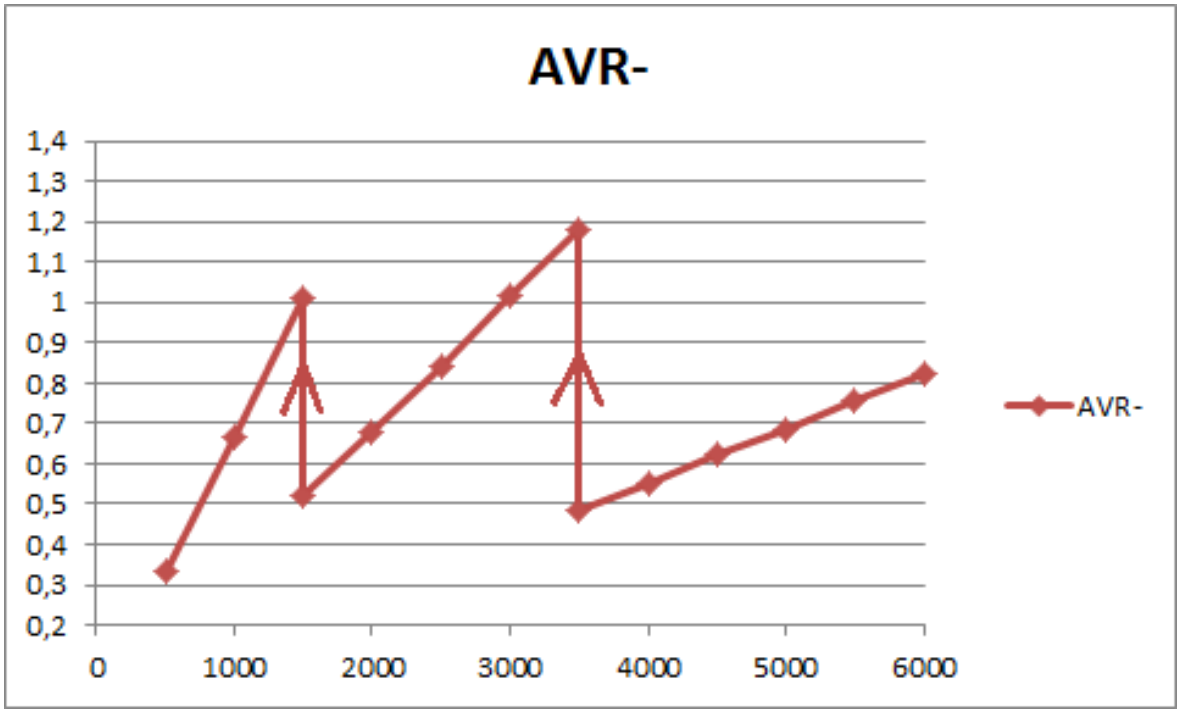
Tabela 26 – Valores corrigidos da utilização do processador pela tarefa AVR

VELOCIDADE (ω) DECRESCENTE - AVR HABILITADO							
ω (rpm)	ω display (rpm)	u_AVR (%)	n	Tbsf	delta (ms)	T_BSF/T_AVR	u_AVR corrigido (%)
		max					
6000	5998	0,9853	3	25	10,005	2,498750625	0,820672997
5500	5530	0,9869	3	25	10,851	2,303935121	0,757917857
5000	4978	0,9859	3	25	12,058	2,073312324	0,68135954
4500	4528	0,6594	2	25	13,259	1,885511728	0,621653217
4000	4010	0,6572	2	25	14,949	1,672352666	0,549535086
3501	3558	0,6584	2	25	16,882	1,480867196	0,487501481
3499	3502	1,617	2	25	17,123	1,460024528	1,180429831
3000	3016	1,619	2	25	19,899	1,25634454	1,017010905
2500	2502	0,8088	1	25	23,988	1,042187761	0,842921461
2000	2006	0,809	1	25	29,894	0,836288218	0,676557169
1501	1540	0,8091	1	25	38,94	0,642013354	0,519453005
1499	1503	1,61	1	25	39,791	0,628282778	1,011535272
1000	995	1,608	1	25	60,278	0,414745015	0,666909984
500	497	1,608	1	25	120,7	0,207125104	0,333057167

Fonte: Elaborado pelo Autor(2016)

Traçando-se o gráfico dos valores medidos pela ferramenta T1 e corrigidos pelo fator de correção do Quadro Básico de Escalonamento tem-se o resultado esperado.

Figura 55 - Valores corrigidos de u da tarefa AVR



Fonte: Elaborado pelo Autor(2016)

7.3 ANALISE TEMPORAL DA TAREFA AVR COMPLETA (ACELERAÇÃO POSITIVA E NEGATIVA)

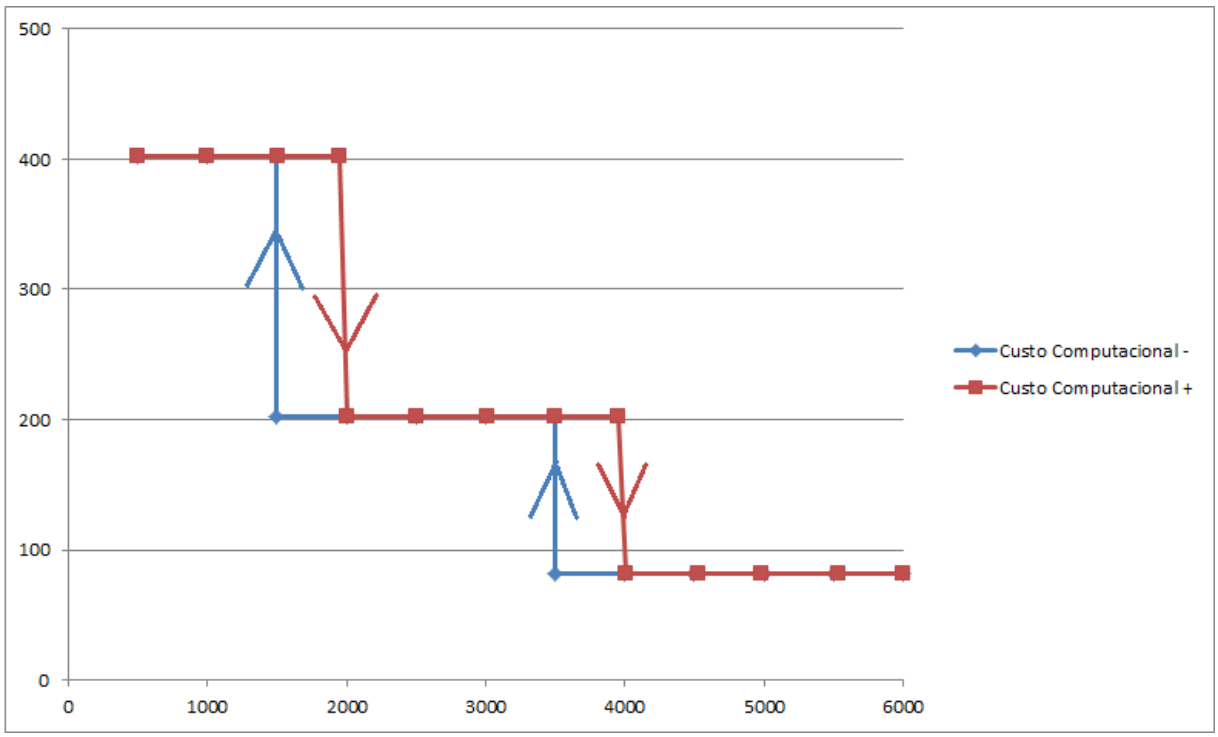
Ao se combinar os resultados obtidos para aceleração positiva e negativa observa-se o comportamento completo da tarefa AVR.

7.3.1 Validação do Custo computacional da tarefa AVR em função da velocidade de rotação do motor ω

Ao analisar-se o custo computacional da tarefa AVR em função da velocidade de rotação do motor nota-se a diferença de comportamento do sistema quando em aceleração positiva ou aceleração negativa. A faixa de histerese de 500rpm ocorre

nos pontos determinados de forma que não haja instabilidade nas faixas de rotação limítrofes.

Figura 56 – Custo computacional da tarefa AVR para acelerações positivas ou negativas

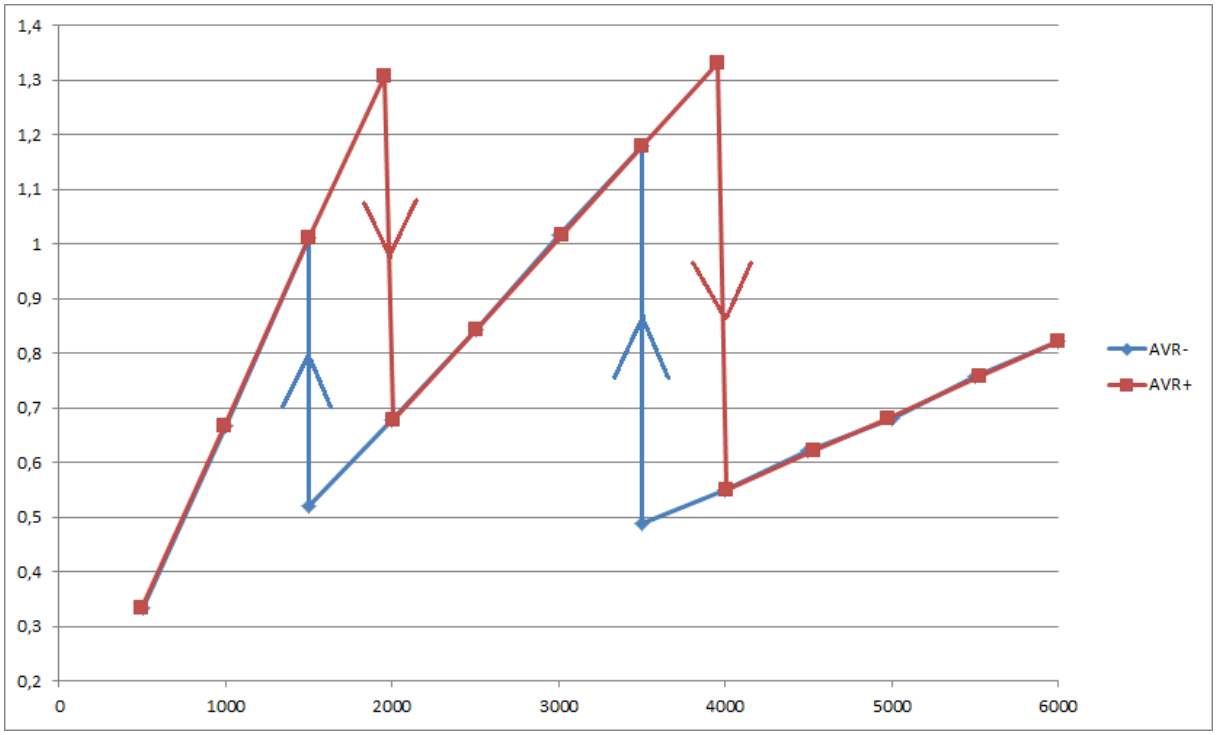


Fonte: Elaborado pelo Autor(2016)

7.3.2 Validação da utilização da CPU u pela tarefa AVR em função da velocidade de rotação do motor ω

Da mesma forma que o custo computacional, a histerese determinada em projeto também se manifesta no valor da utilização da CPU u pela tarefa AVR. Observando-se o comportamento do valor de u em aceleração positiva nota-se que ocorre uma utilização mais alta da CPU em relação à aceleração negativa nas faixas de histerese, em compensação a funcionalidade reduzida das tarefas se mantem por faixas de rotação maiores quando em desaceleração.

Figura 57 – Utilização do processador pela tarefa AVR em acelerações positivas ou negativas



Fonte: Elaborado pelo Autor(2016)

7.4 ANALISE TEMPORAL DA TAREFA ANGULAR COM AVR DESABILITADO

Tabela 27 - Valores medidos com AVR desabilitado

VELOCIDADE (ω) CRESCENTE - AVR DESABILITADO										
ω (rpm)	ω display (rpm)	u_AVR (%)			C (µs)	delta (ms)	u_CPU (%)			modo AVR
		min	med	max			min	med	max	
500	497	0	0,5358	1,608	401,86	-	66,34	67,05	67,9	-
1000	995	0	0,536	1,608	401,945	60,278	66,23	67	67,84	-
1500	1503	0	1,073	1,61	402,42	39,791	66,3	67,59	65,86	-
2000	2015	1,591	1,603	1,609	402,032	29,804	67,29	67,93	68,54	-
2500	2500	1,591	2,112	3,153	401,96	23,993	67,23	67,94	68,76	-
3000	3007	1,607	2,139	3,189	401,918	19,998	67,89	68,64	69,09	-
3500	3504	1,592	2,642	3,184	401,95	17,122	67,23	68,48	69,42	-
4000	4016	1,591	2,654	3,217	402,02	14,949	67,23	68,48	70,13	-
4500	4518	3,151	3,152	3,153	401,943	13,259	68,16	68,38	68,83	-
5000	5030	3,152	3,688	4,728	401,69	11,933	68,77	69,32	70,38	-
5500	5530	3,153	3,694	4,776	401,983	10,885	68,12	69,3	71,04	-
6000	5995	3,152	3,662	4,682	401,903	10,005	68,78	68,87	68,99	-

Fonte: Elaborado pelo Autor(2016)

7.4.1 Validação do Custo computacional da tarefa angular em função da velocidade de rotação do motor ω

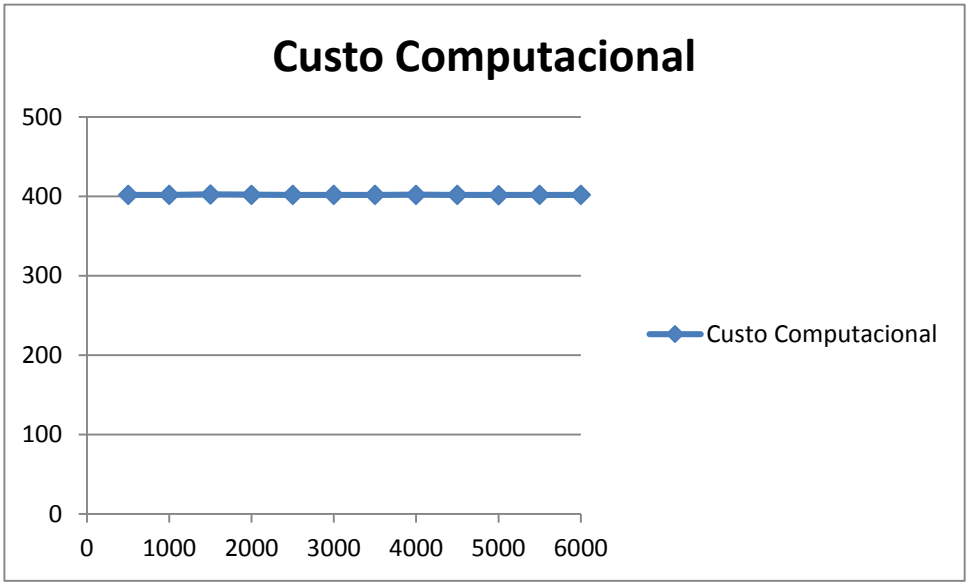
Nota-se que custo computacional medido pela ferramenta T1 (Tabela 28) ficou inalterado em todas as faixas de rotação, uma vez que com o recurso AVR desabilitado a função passa a ser uma função angular convencional que não tem sua funcionalidade alterada de acordo com a rotação do motor, portando, mantendo o mesmo custo computacional para qualquer velocidade de rotação do motor.

Tabela 28 - Custo Computacional (CET) da tarefa angular sem AVR medido pela ferramenta T1

VELOCIDADE (ω) CRESCENTE - AVR DESABILITADO			
ω (rpm)	ω display (rpm)	C (µs)	modo AVR
500	497	401,86	-
1000	995	401,945	-
1500	1503	402,42	-
2000	2015	402,032	-
2500	2500	401,96	-
3000	3007	401,918	-
3500	3504	401,95	-
4000	4016	402,02	-
4500	4518	401,943	-
5000	5030	401,69	-
5500	5530	401,983	-
6000	5995	401,903	-

Fonte: Elaborado pelo Autor(2016)

Figura 58 – Custo computacional da tarefa angular sem AVR



Fonte: Elaborado pelo Autor(2016)

7.4.2 Validação da utilização da CPU u pela tarefa angular em função da velocidade de rotação do motor ω

Aplicando-se o fator de correção (7.1) de forma a validar a implementação da tarefa angular utilizando-se os dados medidos pela ferramenta T1 tem-se os valores corrigidos da utilização do processador u_i pela tarefa angular (Tabela 29)

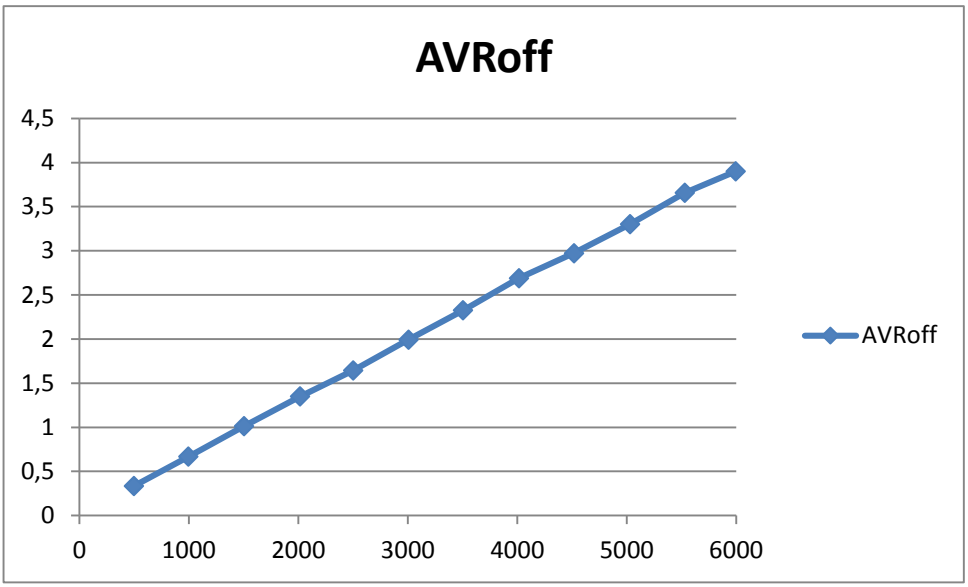
Tabela 29 – Valores corrigidos da utilização do processador pela tarefa angular sem AVR

VELOCIDADE (ω) CRESCENTE - AVR DESABILITADO							
ω (rpm)	ω display (rpm)	u_AVR (%)	n	Tbsf	delta (ms)	T_BSF/T_AVR	u_AVR corrigido (%)
		max					
500	497	1,608	1	25	120,7	0,207125104	0,333057167
1000	995	1,608	1	25	60,278	0,414745015	0,666909984
1500	1503	1,61	1	25	39,791	0,628282778	1,011535272
2000	2015	1,609	1	25	29,804	0,838813582	1,349651054
2500	2500	3,153	2	25	23,993	1,041970575	1,642666611
3000	3007	3,189	2	25	19,998	1,250125013	1,993324332
3500	3504	3,184	2	25	17,122	1,4601098	2,324494802
4000	4016	3,217	2	25	14,949	1,672352666	2,689979263
4500	4518	3,153	2	25	13,259	1,885511728	2,972509239
5000	5030	4,728	3	25	11,933	2,095030587	3,301768206
5500	5530	4,776	3	25	10,885	2,296738631	3,656407901
6000	5995	4,682	3	25	10,005	2,498750625	3,899716808

Fonte: Elaborado pelo Autor(2016)

Traçando-se o gráfico dos valores medidos pela ferramenta T1 e corrigidos pelo fator de correção do Quadro Básico de Escalonamento tem-se o resultado esperado.

Figura 59 - Valores corrigidos de u da tarefa angular



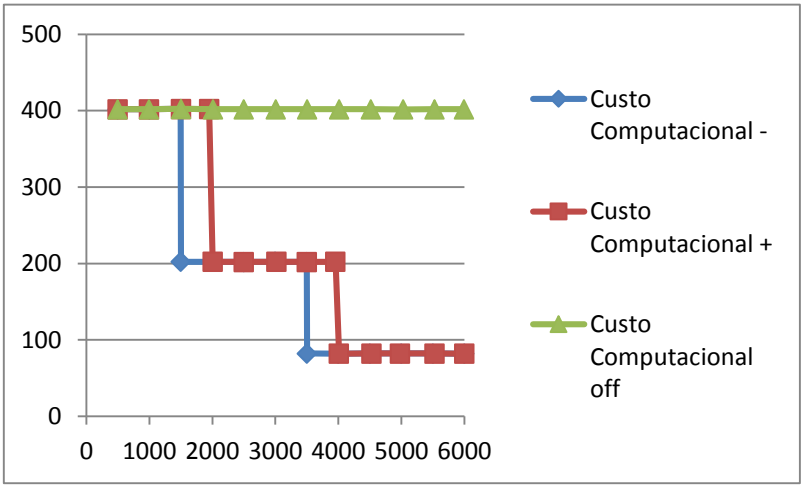
Fonte: Elaborado pelo Autor(2016)

7.5 COMPARATIVO ENTRE TAREFAS ANGULARES COM AVR E SEM AVR

7.5.1 Custo Computacional

O Custo Computacional de uma tarefa angular convencional sem mantem inalterado (Figura 60) qualquer que seja a velocidade de rotação do motor, enquanto a tarefa angular com AVR reduz sua funcionalidade de forma a reduzir o custo computacional a medida que a velocidade de rotação aumenta.

Figura 60 – Custo Computacional tarefa angular com AVR e sem AVR

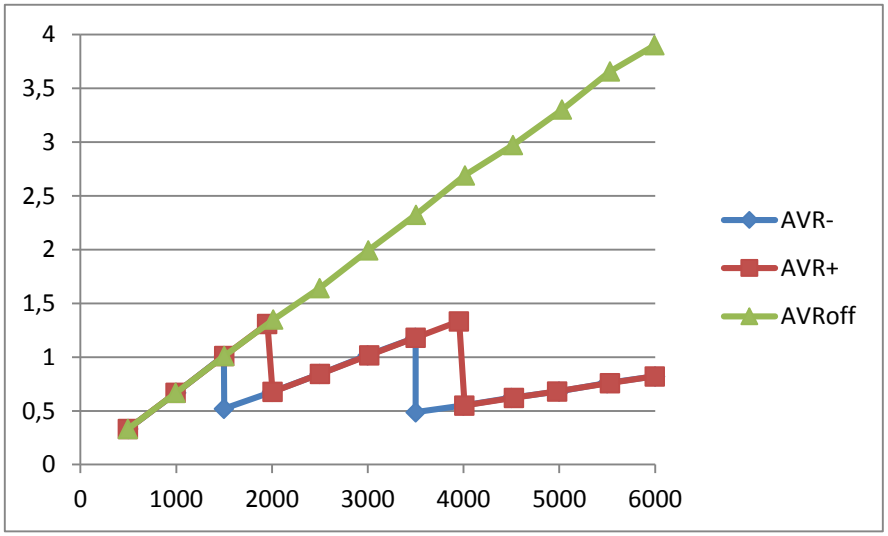


Fonte: Elaborado pelo Autor(2016)

7.5.2 Utilização da CPU

A utilização da CPU (Figura 61) por parte da tarefa AVR torna-se muito inferior à utilização pela tarefa angular convencional quando analisada em altas rotações, pois o alto custo computacional inerente à tarefa convencional causa um impacto significativo na sua utilização da CPU, impacto este diretamente proporcional à velocidade de rotação do motor.

Figura 61 – Utilização da CPU tarefa com AVR e sem AVR



Fonte: Elaborado pelo Autor(2016)

8 CONCLUSÕES

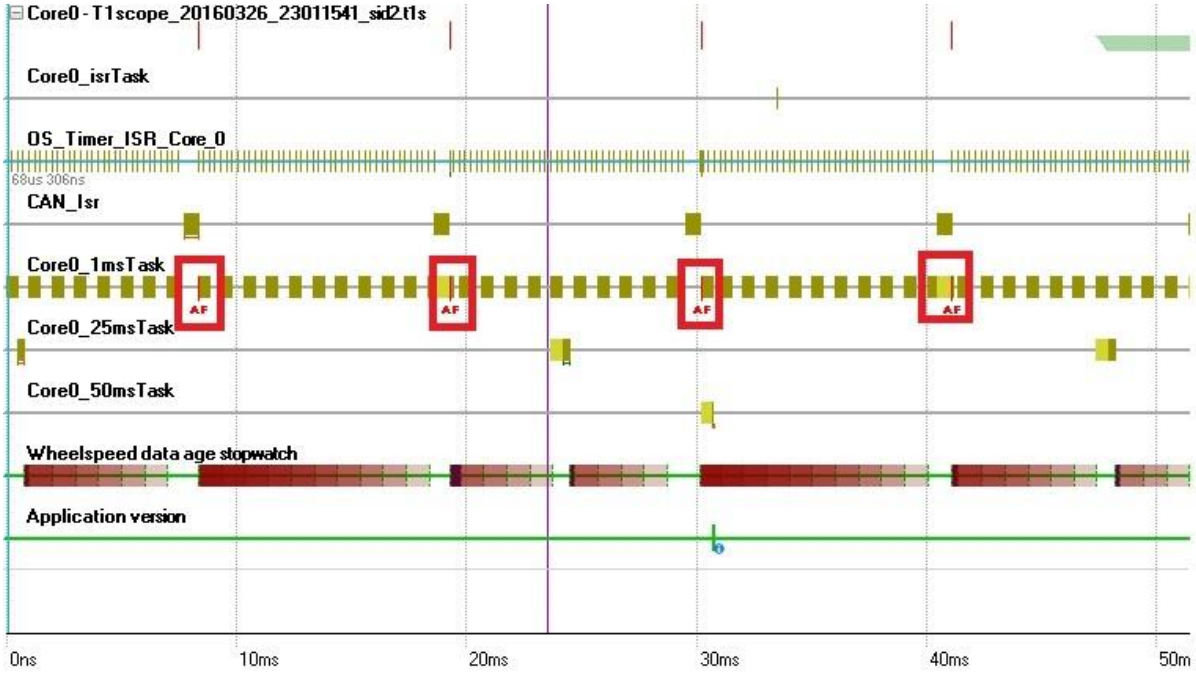
No decorrer do trabalho foi possível perceber a magnitude dos assuntos “Analise Temporal” e “Sistemas Embarcados”, e ainda assim o quanto ainda há a ser explorado. Em sistemas embarcados automotivos especialmente, notou-se certa dificuldade em se obter informações, uma vez que muitas destas informações refletem o estado da arte em tecnologia automotiva e normalmente são tecnologias proprietárias. O acesso a ferramentas como o Kit ATdemo e a ferramenta de analise temporal T1 *suite* foi essencial para o desenvolvimento do trabalho.

Através das ferramentas disponíveis foi possível analisar os resultados temporais da tarefa AVR em tempo real, e através dos relatórios e diagramas temporais emitidos foi possível comparar os resultados obtidos no sistema real com os resultados teóricos esperados e comprovar a funcionalidade da “plataforma de emulação de sistemas de gerenciamento de motor com AVR” desenvolvida para este trabalho.

Foi comprovada a eficácia da técnica AVR para redução da utilização do processador por tarefas com alta frequência de ativação.

Também foi possível utilizar a ferramenta de analise temporal para solucionar falhas na própria metodologia inicialmente proposta, quando a interrupção da tarefa AVR havia sido direcionada para a CPU0, onde existem diversas tarefas relacionadas ao gerenciamento do sistema operacional e da ferramenta T1, além de tarefas com período muito baixo, como é o caso da tarefa “Core0_1ms”, que deve ser executada a cada 1ms. Em certas situações, a velocidade de rotação do motor aliada ao alto tempo de execução da tarefa AVR (CET) causou diversas falhas de ativação (Figura 62) da tarefa “Core0_1ms”, além de eventuais falhas de comunicação entre os módulos T1-TARGET-SW no kit ATdemo e T1-HOST-SW no PC, o que ensejou a alteração da configuração do módulo “Interrupt Router” do microcontrolador de forma a direcionar a interrupção da tarefa AVR para a CPU2.

Figura 62 – Inscrição "AF" (*Activation Failed)* indicando a falha na ativação da tarefa "Core0_1ms"



Fonte: Gliwa T1, Adaptado pelo Autor(2016)

9 TRABALHOS FUTUROS

Dentre os possíveis trabalhos futuros destacam-se:

- - Analise temporal de diversas tarefas AVR com a mesma fonte de rotação, porém com períodos angulares variados.
- - Utilização da ferramenta T1 em conjunto com a "plataforma de emulação de sistemas de gerenciamento de motor com AVR" para realizar a analise temporal de funções periódicas convencionais
- - Analise comparativa entre diversas técnicas de escalonamento aplicadas ao mesmo sistema

REFERÊNCIAS

AUTOSAR. **Explanation of Interrupt Handling in AUTOSAR.** Disponível em: <https://www.autosar.org/fileadmin/files/releases/3-2/software-architecture/general/auxiliary/AUTOSAR_InterruptHandling_Explanation.pdf>. Acesso em 08 mar. 2016.

BIONDI, A.; BUTTAZZO, G. Engine Control: Task Modeling and Analysis. In: INTERNATIONAL CONFERENCE ON DESIGN, AUTOMATION & TEST IN EUROPE 2015. March 9-13, 2015. Grenoble, France.

BIONDI, A.; NATALE, M. D.; BUTTAZZO, G. Response-Time Analysis for Real-Time Tasks in Engine Control Applications. In: IEEE INTERNATIONAL CONFERENCE ON CYBER-PHYSICAL SYSTEMS ICCPS 2015. April 13-16, 2015. Seattle, USA.

DAVIS, R. I.; FELD, T.; POLLEX, V.; SLOMKA, F. Schedulability tests for tasks with variable rate-dependent behavior under fixed priority scheduling. In: 20th IEEE REAL-TIME AND EMBEDDED TECHNOLOGY AND APPLICATIONS SYMPOSIUM, Berlin, Germany, April 2014.

GLIWA GmbH. **OT1 - An open timing information exchange format.** Disponível em: <https://www.gliwa.com/downloads/OT1%20Introduction.pdf>. Acesso em 17 out. 2015.

GLIWA, P. **Timing analysis and timing verification today and in the future.** In: 18th EUROFORUM Deutschland SE, 2014. Munich, Germany.

GLIWA, P.; SARNOWSKI, H.; JERSAK,M.; RICHTER,K. Timing analysis for the early verification of software. **ATZelektronik**, Heidelberg (Germany), v.4, p. 28-31, Jan. 2009.

GUZZELLA, L.; ONDER, C.H. **Introduction to Modeling and Control of Internal Combustion Engine Systems.** Springer-Verlag, 2010.

INFINEON. **AURIX Family - TC27xT.** Disponível em: <http://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/aurix-tm-family/aurix-tm-family-%E2%80%93-tc27xt/channel.html?channel=db3a30433cfb5caa013d01df64d92edc>. Acesso em 22 out. 2015.

KIM, J.; LAKSHMANAN, K.; RAJKUMAR, R. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems In: THIRD IEEE/ACM INT. CONFERENCE ON CYBER-PHYSICAL SYSTEMS (ICCPS 2012), Beijing, China, April 2012, pp. 28–38.

PAZZAGLIA, P.; BIONDI, A.; BUTTAZZO, G.; NATALE, M. D. A Simulation Framework to Analyze the Scheduling of AVR tasks with respect to Engine Performance. In: 7th INT. WORKSHOP ON ANALYSIS TOOLS AND METHODOLOGIES FOR EMBEDDED AND REAL-TIME SYSTEMS (WATERS 2016) , IN CONJUNCTION WITH THE 28TH EUROMICRO CONFERENCE ON REAL-TIME SYSTEMS (ECRTS 2016), Toulouse, France, July 5, 2016. p 11-15

WILHELM, R. et al. The worst-case execution-time problem overview of methods and survey of tools. **ACM Transactions on Embedded Computing Systems (TECS)**, New York (NY-USA), v. 7, n. 3, p. 1–53, apr. 2008.

SANTOS, M. M. D. **Redes de Comunicação Automotiva:** Características, Tecnologias e Aplicações. 1. ed. São Paulo: Erica, 2010.

NAVET, N.; SIMONOT-LION, F. **Automotive Embedded Systems Handbook**

BOUYSSOUNOUSE, B.; SIFAKIS, J. **Embedded Systems Design:** The Artist Roadmap For Research And Development. Heidelberg: Springer Berlin Heidelberg 2005.

MARWEDEL, P. **Embedded System Design:** Embedded Systems Foundations Of Cyber-Physical Systems. 2. ed. Dortmund: Springer 2011.