Universidade Federal do Amazonas

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Programa de Pós-Graduação em Informática

TXM: Uma Metodologia de Desenvolvimento de HW/SW Ágil para Sistemas Embacardos

Lucas Carvalho Cordeiro

Manaus – Amazonas

Outubro de 2007

Lucas Carvalho Cordeiro

TXM: Uma Metodologia de Desenvolvimento de HW/SW Ágil para Sistemas Embacardos

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Departamento de Ciência da Computação da Universidade Federal do Amazonas, como requisito parcial para obtenção do Título de Mestre em Informática. Área de concentração: Engenharia da Computação.

Orientador: Prof. Dr. Raimundo da Silva Barreto
Co-orientador: Prof. Dr.-Ing. Vicente Ferreira de Lucena Junior

Lucas Carvalho Cordeiro

TXM: Uma Metodologia de Desenvolvimento de HW/SW Ágil para Sistemas Embacardos

Banca Examinadora

Prof. Dr. Raimundo da Silva Barreto – Presidente e Orientador
Departamento de Ciência da Computação – DCC/UFAM

Prof. Dr.-Ing. Vicente Ferreira de Lucena Junior – Co-orientador
Departamento de Eletrônica e Telecomunicações – DET/UFAM

Prof. Dr. Meuse Nogueira de Oliveira Junior
Centro Federal de Educação Tecnológica de Pernambuco – DAES/CEFET-PE

Prof. Dr. Edward David Moreno Ordonez
Universidade do Estado do Amazonas – UEA/BenQ

Manaus – Amazonas
Outubro de 2007

Este trabalho é dedicado a três pessoas que são importantes em minha vida:
Minha mãe Luciete Carvalho pelo seu símbolo de caráter, dignidade
e coragem para enfrentar as dificuldades da vida,
ao meu pai Armando Cordeiro pelos valorosos conselhos
nos momentos de decisão da minha vida
e a minha futura esposa Susy Nunes pela compreensão e apoio.

Agradecimentos

O material contido nesta dissertação de mestrado foi resultado de um trabalho de dois anos e as contribuições foram conduzidas em uma maneira valiosa para a comunidade científica. Desta forma, eu devo agradecer a várias pessoas pelas conquistas obtidas.

Gostaria primeiramente de agradecer a DEUS por ter me dado saúde e paz para a realização dos meus objetivos. Gostaria também de agradecer a minha família e noiva que me apoiaram durante os dois últimos anos para a conclusão deste trabalho. Sem o sincero apoio deles, seria muito difícil ter concluído esta dissertação.

Meus sinceros agradecimentos ao meu orientador e amigo Raimundo Barreto pelo tempo investido na discussão das idéias, pelo esforço, dedicação e paciência durante toda a execução do meu trabalho de mestrado.

Obrigado também a todos os membros do programa de pós-graduação em informática da UFAM pela oportunidade de realizar o mestrado e pelos recursos fornecidos durante o curso.

Sou grato ao colega Rafael Barcelos pelas valorosas idéias sobre os artigos científicos resultantes desta dissertação e pelas longas conversas sobre o mestrado e suas implicações. Sou também grato aos colegas Carlos Mar, Eduardo Bezerra, Fabiano Cruz e Daniel Patrick que cooperaram com a execução dos experimentos e pelas pizzas que comprávamos no final de um dia de trabalho pesado nos experimentos. Além disso, gostaria também de agradecer a Petrina Kimura pela ajuda na implementação dos algoritmos de particionamento.

"Doing the same thing, and Expecting a different outcome is the definition of insanity".

Albert Einstein (1879-1955)

Resumo

Atualmente, a sociedade tem se tornando cada vez mais dependente em sistemas embarcados. Tais sistemas podem ser representados desde simples aparelhos domésticos até aeronaves. Sistemas embarcados diferem bastante da maioria dos sistemas de aplicação *Desktop*, pois devem ser altamente otimizados para o ciclo de vida, devem atender restrições temporais e de consumo de energia, e devem ainda tratar de limitações de recursos tais como tamanho e peso. Porém, sistemas embarcados também compartilham algumas características com aplicacações *Desktop* tais como *complexidade* e *incerteza*. Várias metodologias podem ser aplicadas ao desenvolvimento de sistemas embarcados. No entanto, a diversidade extrema das aplicações do mundo embarcado faz com que dificulte a sua generalização.

Por conseguinte, esta dissertação de mestrado tem como objetivo adaptar uma metodologia de desenvolvimento (nomeada como TXM - The neXt Methodology) através do uso de métodos ágeis (XP e Scrum) com padrões organizacionais e adaptá-los para o desenvolvimento de sistemas embarcados de tempo real levando em consideração restrições de consumo de energia, tempo de execução, tamanho de programa e memória de dados. O conceito de projeto baseado em plataforma assim como a técnica de particionamento de hardware/software são usadas na metodologia proposta com o intuito de assegurar que as restrições do sistema sejam atendidas em uma maneira iterativa e incremental e reduzir o custo e tempo de projeto do produto. Estudos de caso envolvendo projetos do oxímetro de pulso, *soft-starter digital* e simulador do motor de indução são desenvolvidos com o propósito de discutir os pontos fortes e fracos da metodologia ágil proposta.

Palavras-chave: Metodologias Ágeis, Padrões Organizacionais, Desenvolvimento Ágil Embarcado, Projeto Baseado em Plataforma, Software de Tempo Real.

Abstract

Nowadays, the human life has become more and more dependent on embedded systems. Such systems are everywhere, from home appliances to spaceships. Embedded systems differ significantly from more traditional desktop applications. Usually, embedded systems must be highly optimized for life-cycle, meet stringent timing and energy constraints, and address resources limitations. However, embedded systems share common characteristics with desktop applications such as complexity and uncertainty. Several methodologies may be applied to embedded system development. However, the extreme diversity of applications makes the generalization difficult.

Therefore, this master thesis aims to adapt a development methodology (named as TXM - The neXt Methodology) by combining the agile methods (Scrum and XP) with organizational patterns and adapt them to build embedded real-time systems focusing on the energy consumption, execution time, program size, and data memory size constraints. The platform-based design concept as well as hardware/software partitioning technique are used in the proposed methodology with the purpose of helping the embedded system designer meet the system constraints in an iterative and incremental manner and help reduce substantially the design time and cost of the product. Three case study involving the pulse oximeter, digital soft-starter and induction motor simulator projects are developed in order to discuss strengths and weakness of the proposed agile methodology in the domain of embedded real-time systems.

Keywords: Agile methodologies, Organizational Patterns, Embedded Agile Development, Platform-Based Design, Real-time Software.

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1

Introdução

A quantidade de sistemas embarcados produzidos está aumentando cada vez mais. Atualmente, sistemas embarcados estão presentes desde sistemas críticos de segurança até software de entretenimento. Sistemas embarcados estão se tornando cada vez mais importantes na nossa sociedade e, ao mesmo tempo, estão aumentando em *complexidade* e *tamanho*. A medida que os micro-controladores se tornam mais baratos, menores e mais confiáveis faz com que seja possível mover mais funcionalidades de hardware para software. Análise de mercado mostra que mais de 80% das funcionalidades do produto é implementada em software [55]. Sendo assim, equipes de desenvolvimento estão usando software com o propósito de customizar produtos, aumentar flexibilidade, fornecer rápida resposta a mudanças e lançar produtos no mercado rapidamente.

No entanto, várias metodologias de desenvolvimento que são usadas para produzir software que executa nos computadores pessoais (PCs) não são apropriadas para desenvolver sistemas embarcados de tempo real. Este tipo de sistema contém características diferentes tais como software e hardware dedicado, e restrições que não são comuns para sistemas baseado em PC (p.e., consumo de energia, desempenho, tamanho da memória). Além disso, engenheiros de sistemas embarcados não possuem boas habilidades em engenharia de software. Eles possuem habilidades de desenvolvimento de hardware e freqüentemente usam linguagens de programação para resolver os problemas em mão de forma empírica [25]. Um outro ponto importante é que algumas classes de sistemas embarcados de tempo real podem pôr vidas ou funções de negócio críticas em risco (*criticidade da missão*). Sendo assim, estes sistemas devem ser tratados diferentemente do caso onde somente o custo da

falha é o investimento do projeto.

Baseado neste contexto, esta dissertação de mestrado propõe uma metodologia de desenvolvimento inovadora baseado nos princípios ágeis tais como planejamento adaptativo, flexibilidade, e a abordagem iterativa e incremental com o intuito de facilitar o desenvolvimento de sistemas embarcados de tempo real. Para alcançar isso, a metodologia proposta é composta por boas práticas de Engenharia de Software e Métodos Ágeis (eXtreme Programming e Scrum) que têm como objetivo minimizar os principais problemas presentes no contexto de desenvolvimento de sistemas embarcados (*gerenciamento de risco* e *volatilidade de requisitos*), e pôr outras práticas que são necessárias para alcançar sistemas embarcados de tempo real (i.e., *projeto baseado em plataforma* [55]). Nesta dissertação de mestrado, a metodologia proposta e seus componentes (papéis, processos e ferramentas) são descritos e experimentos são realizados com o propósito de validar a metodologia.

O restante deste capítulo descreve o contexto desta dissertação de mestrado e o problema que será resolvido com a metodologia proposta. São também apresentados os objetivos e contribuições da metodologia proposta. Finalmente, a estrutura da dissertação de mestrado é apresentada.

1.1 Contexto

Atualmente, métodos ágeis têm sido amplamente usados em pequenas e grandes organizações [34]. Estas organizações têm como objetivo melhorar o processo de desenvolvimento com o propósito de entregar software rapidamente e ao mesmo tempo com alta qualidade. Os métodos ágeis enfatizam simplicidade, software funcional no início das iterações, flexibilidade, e comunicação direta entre desenvolvedores e clientes [3]. Um dos principais benefícios dos métodos ágeis é se adaptar às mudanças e fornecer rápida resposta para as solicitações do cliente. No entanto, existe pouca evidência em qual ambiente e em quais condições os métodos ágeis funcionam.

A literatura cita seu uso em projetos de desenvolvimento de aplicações em PC Desktop que são geralmente implementadas em linguagens orientadas a objetos [46, 9]. Em contrapartida, desenvolvimento de software embarcado difere do desenvolvimento de aplicação

tradicional embora eles compartilhem questões em comum como *complexidade* e *incerteza*. Desenvolvimento de software embarcado pode tratar de requisitos temporais, consumo de energia, tamanho de código, confiabilidade entre outros [25]. O estudo de caso desenvolvido nesta dissertação de mestrado, o oxímetro de pulso, é um bom exemplo de tal tipo de sistema. A principal função deste equipamento é medir o nível de saturação de oxigênio (SpO_2) e freqüência cardíaca (HR) do paciente.

A vida do paciente pode ser comprometida caso a aplicação não cumpra com o *deadline*[1] para leitura de um dado do sensor (restrições críticas de tempo real). Os oxímetros de pulso são de importância crítica na medicina de emergência e são também bastante usados em pacientes com problemas respiratórios e cardíacos. Deste modo, o tempo no qual os resultados são produzidos pelo oxímetro de pulso é de extrema importância. Isto leva à definição de duas diferentes classes de sistemas embarcados: *sistemas de tempo real crítico e brando*.

Um sistema de tempo real crítico é solicitado para produzir os resultados da aplicação no momento correto [31]. Um sistema de tempo real brando é um sistema cujo operação é degradada se os resultados não são produzidos no momento correto [11]. Além do oxímetro de pulso, dois diferentes exemplos podem ser considerados para diferenciar entre sistema de tempo real crítico e brando: um controlador embarcado para operar uma aeronave é um exemplo de sistema de tempo real crítico, pois uma falha para verificar a altitude em tempos pré-definidos pode levar a falhas catastróficas enquanto que um telefone celular é um exemplo de sistema de tempo real brando devido ao fato de que o atraso de uma resposta para reagir a um estímulo do usuário pode ainda ter utilidade mesmo depois de perder o *deadline*.

Além disso, o projeto de sistema embarcado solicita o balanceamento constante entre requisitos de hardware e software. Em termos gerais, o projetista de sistemas embarcados de tempo real deve definir o desempenho e quantidade de micro-controladores, tamanho da memória, subsistema mecânico, e interface homem-máquina [30]. Do ponto de vista de hardware/software co-design, elementos de hardware e software são particionados e desenvolvidos simultaneamente no início do projeto. Portanto, o hardware pode evoluir e solicitar mudanças de interface para serem feitas no software levando assim a atrasos

[1] A palavra inglesa *deadline* será usada nesta dissertação no lugar da expressão "prazo de entrega".

do processo de desenvolvimento. Estas mudanças são difícies de evitar devido à incerteza nos requisitos do sistema.

Em contrapartida, a metodologia de projeto baseada em plataforma a qual tem sido discutida por várias décadas no mundo dos PCs, fornece uma outra abordagem para projetar sistemas embarcados de tempo real. O projeto baseado em plataforma pode ser definido como uma camada de abstração que esconde os detalhes da implementação de camadas inferiores [12]. Além disso, a plataforma possibilita o reuso do software e fornece flexibilidade para suportar diferentes produtos ou variantes de produto por meio da programação dos componentes (p.e., micro-controlador e FPGA).

Do ponto de vista da plataforma, software programável produz uma solução mais flexível pelo fato de que é mais fácil de ser modificado enquanto que o hardware programável executa mais rápido e consome menos energia do que o software correspondente [55]. Sendo assim, o balanceamento para a metodologia baseada em plataforma está entre *flexibilidade* e *desempenho*. A próxima seção descreve o problema que será resolvido com a metodologia de desenvolvimento proposta.

1.2 Descrição do Problema

Existem basicamente dois aspectos que são levados em consideração no desenvolvimento de sistemas embarcados. O primeiro aspecto é compartilhado com projetos de desenvolvimento de software de todos os tipos (seja software convencional ou embarcado). O projeto deve ser entregue no prazo especificado com nível de qualidade aceitável e dentro do custo planejado. O segundo aspecto é particular a desenvolvimento de sistemas embarcados. Sistemas embarcados possuem alguns desafios adicionais como, por exemplo: *(i)* desenvolvimento concorrente de hardware/software, *(ii)* o software é desenvolvido em uma plataforma e executa em outra completamente diferente, *(iii)* restrições no ambiente de execução como, por exemplo, tempo de execução, tamanho da memória de dados e programa, *(iv)* configurações de E/S customizadas, *(v)* a plataforma de desenvolvimento muitas vezes não está disponível no início do processo de desenvolvimento.

Com os desafios mencionados acima, surgem as seguintes perguntas: (1) Existe uma maneira de progredir no início do ciclo de desenvolvimento em projetos de sistemas em-

barcados usando as práticas de métodos ágeis? (2) É possível, usando métodos ágeis, gerar uma versão otimizada do sistema que reduza custos de produção?, (3) Em algumas situações a plataforma de desenvolvimento não existe ou o hardware ainda está evoluindo então é possível manter a equipe de desenvolvimento de software fora do caminho crítico do projeto? (4) Como realizar decisões no *design* do sistema que possam atender as restrições da aplicação? Estas perguntas serão respondidas detalhadamente e definem as principais contribuições desta dissertação de mestrado.

1.3 Objetivos

A metodologia de desenvolvimento ágil que foi adaptada nesta dissertação de mestrado é baseada em princípios ágeis tais como *planejamento adaptativo, flexibilidade e desenvolvimento iterativo e incremental* com o propósito de facilitar o processo de desenvolvimento de sistemas embarcados. Para alcançar este objetivo, esta metodologia é composta por boas práticas de Engenharia de Software e Métodos Ágeis (Scrum e XP) o qual tem como objetivo minimizar os principais problemas presentes no contexto de desenvolvimento de software embarcado *(volatilidade de requisitos e gerenciamento de risco).*

A solução de um sistema embarcado envolve componentes de hardware e software interconectados de tal maneira que implementam um conjunto de funcionalidades enquanto satisfazem um conjunto de restrições. Sendo assim, a metodologia também fornece práticas que são necessárias para atingir o projeto de sistemas embarcados de tempo real *(projeto baseado em plataforma [55]).* Com estes objetivos em mente, a metodologia proposta define papéis, responsabilidades, processos, práticas e ferramentas para auxiliar o projetista do sistema no ciclo de vida do projeto. Deste modo, os principais objetivos desta dissertação de mestrado podem ser descritos da seguinte maneira:

i) Propor uma metodologia de desenvolvimento através da integração de práticas, técnicas e ferramentas para balancear custo e tempo para mercado em vista de restrições de funcionalidades e desempenho.

ii) Desenvolver os equipamentos oxímetro de pulso, *soft-starter* digital e simulador do motor de indução monofásico com o intuito de validar a metodologia de desenvolvimento ágil proposta.

iii) Investigar as técnicas de particionamento de hardware/software assim como soluções ótimas e heurísticas usadas para resolver o problema do particionamento.

iv) Investigar metodologias de desenvolvimento de sistemas embarcados com ênfase especial na metodologia de projeto baseada em plataforma.

v) Investigar e analisar os padrões e práticas ágeis com o propósito de adaptá-los com a técnica de particionamento de hardware/software e o conceito de projeto baseado em plataforma.

Sendo assim, a metodologia proposta tem como objetivo ser aplicada em diferentes projetos de sistemas embarcados de tempo real, porém o uso dos processos, práticas, técnicas e ferramentas pode variar dependendo do tipo de projeto. Para esta dissertação de mestrado, nós aplicamos a metodologia proposta somente nas áreas de dispositivos médicos e de sistemas de controle embarcado.

1.4 Metodologia de Trabalho

Esta seção descreve as principais atividades que foram identificadas para se alcançar os objetivos desta dissertação de mestrado. Estas atividades fornecem os passos e direções necessárias para desenvolver a metodologia proposta e elas podem ser descritas da seguinte maneira:

/A1/ Investigar e avaliar princípios e práticas ágeis que são relevantes para o desenvolvimento de sistemas embarcados de tempo real.

/A2/ Investigar as características de sistemas embarcados de tempo real que influenciam seu ciclo de desenvolvimento.

/A3/ Estudar a técnica de particionamento de hardware/software assim como as soluções ótimas e heurísticas usadas para resolver o problema do particionamento.

/A4/ Estudar metodologias de desenvolvimento de sistemas embarcados com ênfase especial na metodologia de projeto baseada em plataforma.

/A5/ Analisar as práticas ágeis investigadas em /A1/ com o propósito de combiná-las com a técnica de particionamento de hardware/software e o conceito de projeto baseado em plataforma estudado nos itens /A3/ e /A4/ respectivamente.

/A6/ Criar e definir papéis e processos da metodologia proposta usando os resultados obtidos em /A5/.

/A7/ Analisar se as práticas propostas nesta nova metodologia cobrem todo o ciclo de vida de desenvolvimento do produto comparando-a com os padrões organizacionais [16] apresentados na seção 2.3.

/A8/ Propor práticas de desenvolvimento a partir dos padrões organizacionais [16] para cobrir as lacunas na metodologia proposta identificadas em /A7/.

/A9/ Implementar algoritmos de particionamento com o propósito de auxiliar o projetista de sistemas embarcados a decidir quais funções serão implementadas em hardware ou em software. Soluções ótimas e heurísticas devem ser desenvolvidos para este propósito.

/A10/ O oxímetro de pulso descrito na seção 6.1 deve ser desenvolvimento usando a metodologia proposta em /A6/.

Deste modo, esta dissertação de mestrado é composta de um conjunto de tarefas que requerem uma certa ordem de precedência para a execução das atividades. A tabela 1.1 mostra as atividades mais importantes que foram identificadas para o projeto assim como a duração e dependências com outras tarefas. Por exemplo, a partir da tabela 1.1, as atividades /A3/ e /A4/ dependem da atividade /A2/. Isto significa que /A2/ (estudar as características de sistemas embarcados) deve ser finalizada antes de iniciar /A3/ (Estudar as técnicas de particionamento de HW/SW).

Tabela 1.1: Duração e dependências das atividades

Atividade	Duração (dias)	Dependências
A1	25	-
A2	30	-
A3	20	A2
A4	20	A2
A5	50	A1, A3 e A4
A6	30	A5
A7	15	A6
A8	20	A7
A9	25	A3 e A4
A10	90	A6 e A9

Para melhor visualizar as dependências entre as atividades, a Figura 1.1 mostra a rede de atividades deste projeto de dissertação.

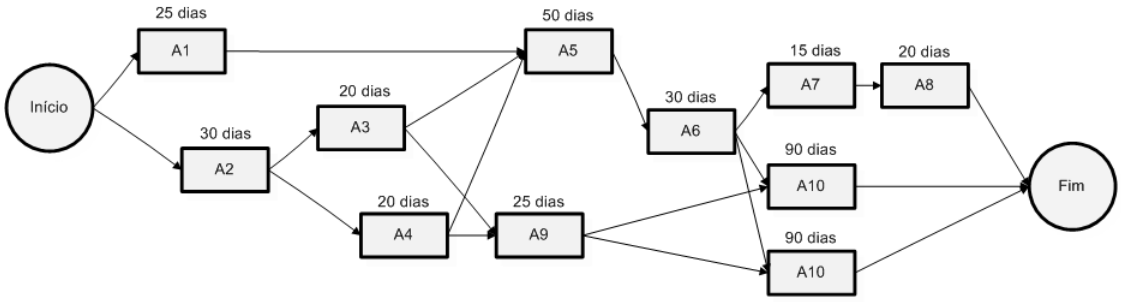

Figura 1.1: Rede de Atividades do Projeto de Dissertação.

Esta dissertação de mestrado consiste de quatro marcos. No primeiro marco, o estado da arte é realizado o qual tem como objetivo estudar todos os conceitos, teorias e tecnologias que fazem parte desta dissertação de mestrado. No segundo marco, a metodologia de desenvolvimento ágil para sistemas embarcados é proposta. No terceiro marco, os experimentos que tem como intuito implementar o oxímetro de pulso, *soft-starter* digital e o simulador do motor de indução monofásico usando a metodologia proposta no marco anterior. Finalmente, o documento de dissertação de mestrado é redigido e a apresentação final é preparada. Figura 1.2 mostra a linha de tempo do projeto planejado.

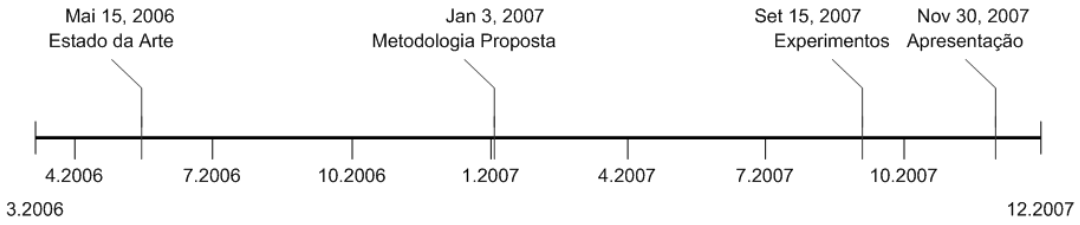

Figura 1.2: Linha de Tempo do Projeto.

Cada marco do projeto consiste de um subconjunto de atividades. Para este propósito, a Tabela 1.2 apresenta todas as atividades e a data final de cada marco do projeto planejado.

1.5 Organização da Dissertação

A organização deste trabalho conserva uma relação de interdependência entre os capítulos, de modo que conceitos e definições apresentadas nos capítulos iniciais são amplamente utilizados nos capítulos seguintes. O texto está organizado em oito capítulos, dos quais este

Tabela 1.2: Marco do Projeto

Marco	Atividade	Data
Estado da Arte	A1, A2, A3 e A4	15.05.2006
Metodologia Proposta	A5, A6, A7, A8 e A9	01.03.2007
Realização dos Experimentos	A10 e A11	15.09.2007
Apresentação Final	Dissertação e Apresentação	30.11.2007
Entrega da Dissertação	Revisão da dissertação	30.12.2007

é o primeiro. O Capítulo 2 fornece os principais conceitos e teorias que são necessários para entender este trabalho, o qual inclui: uma breve visão geral de métodos ágeis e padrões que serão integrados e adaptados na metodologia proposta, a definição e características típicas de sistemas embarcados de tempo real, e uma visão geral da metodologia de projeto baseado em plataforma.

O Capítulo 3 descreve os trabalhos relacionados com as metodologias de desenvolvimento de sistemas embarcados. O Capítulo 4 descreve a metodologia ágil proposta em termos de papéis, responsabilidades e processos para serem aplicados em projetos de sistemas embarcados de tempo real. O Capítulo 5 descreve as ferramentas usadas na execução do experimento. O Capítulo 6 descreve como os processos da metodologia proposta foram usados para desenvolver os projetos do oxímetro de pulso, *soft-starter* digital e o simulador do motor de indução. O Capítulo 7 mostra os resultados experimentais da metodologia proposta. Finalmente, o Capítulo 8 resume os resultados e fornece idéias para pesquisas futuras.

Capítulo 2

Conceitos Básicos

Este capítulo apresenta os principais conceitos e teorias que são necessárias à contextualização e compreensão da metodologia de desenvolvimento proposta. Sendo assim, os conceitos e teorias são apresentados de forma disjunta, e somente nos capítulos seguintes, são relacionados de maneira a compor um referencial lógico e estruturado. Este capítulo está dividido entre três diferentes seções da seguinte maneira: conceitos relacionados a sistemas embarcados, uma breve descrição da metodologia de projeto baseada em plataforma e uma visão geral dos métodos ágeis XP e Scrum.

A primeira seção descreve as características dos sistemas embarcados de tempo real. Nesta seção são também apresentados aspectos básicos de sistemas embarcados assim como a técnica de particionamento de hardware/software. A seção seguinte apresenta a metodologia de projeto baseada em plataforma a qual representa um importante conceito dentro da metodologia de desenvolvimento proposta. Finalmente, a terceira seção apresenta as principais práticas, papéis e responsabilidades dos métodos ágeis XP e Scrum assim como os padrões de desenvolvimento ágil.

2.1 Domínio de Sistemas Embarcados

Esta seção apresenta os sistemas embarcados de tempo real a partir de diferentes perspectivas com ênfase na diferença fundamental entre sistemas de tempo real crítico e brando, características que influenciam o ciclo de vida e os desafios adicionais no desenvolvimento de sistemas embarcados. Além disso, é também apresentado aspectos básicos e técnicas

para a tarefa de particionamento de hardware/software que tem como objetivo implementar um sistema embarcado de tempo real atendendo a um conjunto de restrições, tais como custo, requisitos temporais, tamanho e consumo de energia.

2.1.1 Definição e Exemplos

Os micro-controladores se tornando cada vez mais baratos, menores e mais confiáveis faz com que seja enconomicamente atrativo usá-los como sistema computacional em diversas aplicações. Estes sistemas computacionais são usados em uma ampla gama de sistemas que vão desde monitoramento de condições de máquinas até sistemas de controle de *airbag*. Um sistema embarcado de tempo real pode então ser visto (veja Figura 2.1) como consistindo do sistema mecânico, o computador embarcado e uma interface homem-máquina [31]. Um telefone celular é um exemplo de sistema embarcado. Ele contém um teclado e *display* para interagir com o usuário e algumas milhares de linhas de código integradas no micro-processador para executar tarefas específicas.

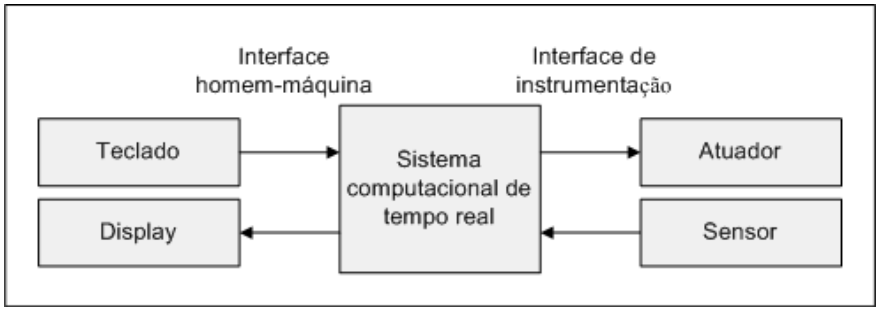

Figura 2.1: Sistema Embarcado de Tempo Real.

Existem essencialmente dois tipos de sistemas embarcados de tempo real: sistemas críticos e brandos. Um sistema de tempo real crítico é solicitado para produzir os resultados da aplicação no momento correto [31]. Estes sistemas exigem garantia quanto à previsibilidade e corretude de suas respostas e comportamento. São geralmente utilizados para a execução de tarefas críticas, onde qualquer falha ou atraso pode resultar em perdas de vidas humanas e grandes prejuízos materiais. Assim, são normalmente sistemas mais robustos que os convencionais e devem ser projetados para gerar o comportamento correto do sistema com o propósito de atender os requisitos da aplicação.

Um sistema brando de tempo real é um sistema cujo operação é degradada se os resultados não são produzidos de acordo com os requisitos temporais especificados [11]. Deste modo, estes sistemas não possuem requisitos tão restritos quanto os sistemas críticos e buscam um balanceamento entre tempo de computação e precisão de suas respostas. Nesse tipo de sistema, o não cumprimento de uma restrição resulta apenas em uma perda de desempenho em relação ao desejado.

A Tabela 2.1 mostra o impacto na engenharia de software para as quatro diferentes categorias de sistemas embarcados de tempo real [15]. Os símbolos + e ++++ indicam baixo e alto peso respectivamente. Sistemas embarcados de tempo real crítico e rápido, em termos computacionais, são menores (ou seja, fazem parte de um sistema maior). Álém disso, este tipo de sistema possui geralmente um tempo de execução curto (tipicamente dezenas de milisegundos ou mais rápido) e *deadline* crítico[1], porém possuem menor complexidade e tamanho do código [15]. Para atender os requisitos temporais da aplicação, o tamanho do software deve ser pequeno e conseqüentemente a complexidade do software é diminuída. Um bom exemplo de um sistema crítico e rápido é o controle de *airbag* equipado nos automóveis. Se o carro colide com algum obstáculo então o sistema de *airbag* deve ser ativado para proteger o motorista contra a colisão.

Tabela 2.1: Impactos na Engenharia de Software [15]

Atributo/Categoria	Tempo de execução	*Deadline*	Tamanho	Complexidade
Crítico/Rápido	++++	++++	+	+
Crítico/Lento	+	++++	+→+++	+→+++
Brando/Rápido	++++	++	+→+++	+→+++
Brando/Lento	++	++	+→+++	+→+++

Em contrapartida, sistemas embarcados de tempo real brando e lento tem um tempo de execução longo (tipicamente na ordem de segundos) e *deadline* não crítico, porém o tamanho e complexidade do software podem variar de baixo para alto. Uma máquina do tipo ATM (*Automated Teller Machine*) é um exemplo de sistema de tempo real embarcado brando e lento, pois este tipo de sistema deve atender a um tempo de resposta aceitável

[1]Neste contexto, tempo de execução define quanto tempo uma dada tarefa do sistema leva para executar em um dado processador e deadline define a criticidade da entrega dos resultados da tarefa em um ponto específico da linha do tempo.

para o cliente. De outro modo, o resultado pode ser considerado como não confiável e pode levar a insatisfação do cliente. O tempo de resposta para sistemas de tempo real rápido pode variar de micro até milisegundos enquanto que o tempo de resposta para um sistema de tempo real lento pode estar dentro de poucos segundos.

A próxima seção descreve as principais características de sistemas embarcados que podem influenciar no processo de desenvolvimento.

2.1.2 Características de Sistemas Embarcados

Desenvolvimento de sistemas embarcados tem uma quantidade de características que diferem substancialmente do desenvolvimento de aplicações convencionais. O relacionamento de hardware e software de um sistema embarcado impacta diretamente o processo de desenvolvimento do sistema através de vários aspectos no ciclo de vida [15]. Sistemas embarcados possuem também severas restrições temporais e limitações físicas que devem ser levados em consideração no processo de projeto do sistema. Em termos gerais, algumas diferenças com aplicação *desktop* que influenciam o processo de desenvolvimento são apresentadas a seguir [30, 31, 15]:

- A interface homem-máquina que consiste de dispositivos de entrada e saída não devem solicitar nenhum treinamento para operar o sistema. Em outras palavras, deve ser de fácil uso por parte do usuário.

- O custo de uma simples unidade de sistema embarcado deve ser a menor possível com o propósito de reduzir custos de produção. Deste modo, o projeto do sistema deve ser altamente otimizado para o custo do ciclo de vida e eficiência.

- O sistema embarcado tem funcionalidade fixa e estrutura rígida. O software é específico para a aplicação e reside em uma memória somente de leitura.

- A qualidade do software deve ser alta, pois não existe muita flexibilidade para mudar o software depois de ser liberado para o mercado.

- O tempo de vida de sistemas embarcados é freqüentemente longo. Desta forma, é necessário uma boa porta de diagnóstico para que seja possível realizar manutenção em campo.

Do ponto de vista do valor de negócio agregado, sistemas embarcados não são vendidos somente porque eles contam com um micro-processador potente. Sistemas embarcados são tipicamente vendidos por que eles fornecem as funcionalidades, qualidade e custo que o cliente procura. Além disso, o tempo para identificar a oportunidade de venda de produto e o tempo para lançá-lo no mercado (também conhecido como *time-to-market*) podem ser de extrema importância para as organizações.

A próxima seção descreve brevemente a técnica de particionamento de hardware/software que tem como objetivo auxiliar o projetista na implementação das funcionalidades enquanto satisfaz simultaneamente as restrições do sistema.

2.1.3 Particionamento de Hardware/Software

As especificações do sistema podem ser modeladas como um conjunto de funções, onde cada função possui uma ou mais restrições. Como os sistemas embarcados tipicamente consistem de componentes de hardware e software então as funções do sistema são implementadas como um conjunto de componentes interconectados tais como circuitos integrados de aplicação específica (*ASIC - Application Specific Integrated Circuit*), microcontroladores (μC) e processadores de aplicação específica (*SAP - Single Application Processor*) como mostrado na Figura 2.2) [8]. Funções implementadas em software obtém atributos tais como número de instruções necessárias para executar em um processador específico. De modo oposto, funções implementadas em hardware afetam o custo do sistema. Para obter uma implementação que satisfaça todas as restrições do projeto, o projetista deve basicamente resolver dois problemas [22]:

a) Selecionar um conjunto de componentes do sistema (alocação);

b) Particionar as funcionalidades do sistema entre estes componentes (partição).

A decisão se uma dada função em um sistema embarcado de tempo real deveria ser implementada em hardware ou em software deve ser realizada com respeito a satisfazer as restrições do projeto assim como balancear entre custo, desempenho e outros atributos. O particionamento de hardware/software é uma instância do problema da otimização de múltiplos objetivos também conhecida como MOP (*multiple-objective optimization*) [26].

Existem duas diferentes abordagens para particionar o sistema como descrito a seguir [22]:

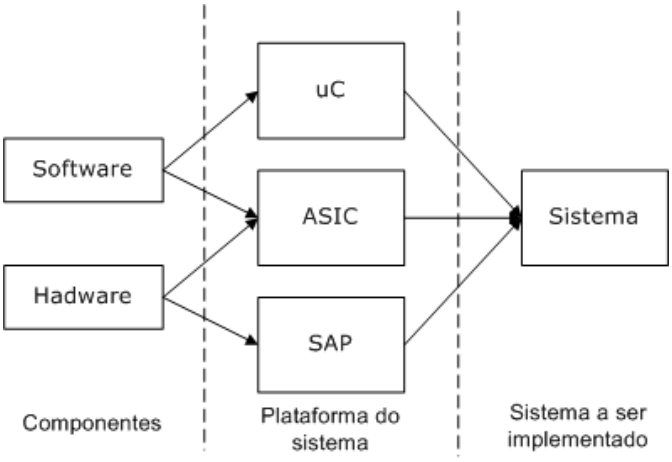

Figura 2.2: Elementos, componentes e sistemas.

- *Particionamento estrutural:* Nesta abordagem, o sistema é primeiro implementado com uma estrutura. A estrutura é uma interconexão de objetos de hardware ou mesmo uma unidade computacional complexa tal como multiplicador de ponto flutuante. Depois disso, a estrutura é particionada na qual separa os objetos em grupos, onde cada grupo representa um componente do sistema.

- *Particionamento funcional:* Nesta abordagem, as funções do sistema são primeiro descompostas em pedaços não divisíveis chamados de objetos funcionais. Depois disso, os objetos são particionados entre os componentes do sistema os quais podem ser implementados ou em hardware ou em software.

A Figura 2.3 mostra a configuração típica de particionamento de sistemas [22]. Primeiro de tudo, o sistema é convertido a um modelo funcional no qual algoritmos de particionamento podem ser usados (modelo do sistema). Depois disso, os algoritmos de particionamento requerem estimativas e uma função objetivo com o intuito de produzir os resultados esperados para posterior análise (saída). Métricas (p.e., tempo de execução, consumo de energia, tamanho do programa e da memória) que definem a qualidade do particionamento podem também ser usadas para melhorar os resultados. Como mostrado na Figura 2.3, as métricas são obtidas a partir da retro-alimentação do projeto que fornece os valores de métricas dos objetos funcionais implementados.

Existem duas maneiras para computar o valor da métrica. A primeira é desenvolver o sistema com o intuito de obter valores de métricas precisos. Em contrapartida, é im-

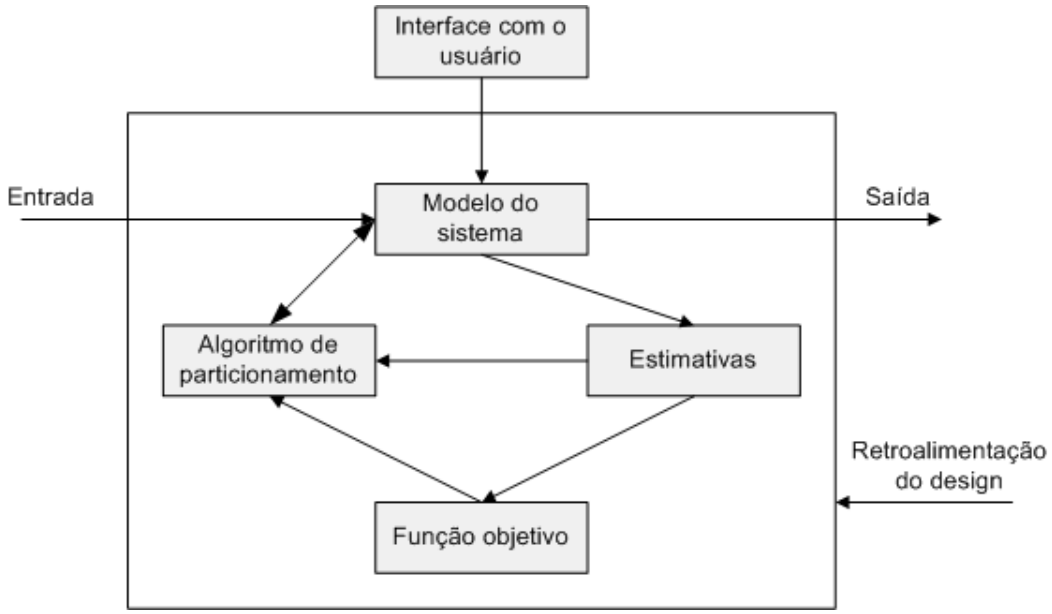

Figura 2.3: Configuração típica de particionamento de sistema.

praticável por que requer muito tempo para a implementação do sistema [22]. A segunda é implementar as principais funcionalidades do sistema. Porém, isto não fornece valores precisos. Sendo assim, *velocidade* e *precisão* são objetivos que competem para determinar os valores das métricas. Finalmente, a interface com o usuário da configuração típica mostrada na Figura 2.3 fornece meios pelos quais as várias partes do sistema podem ser acessadas pelo usuário.

Definição Formal

O principal propósito do particionamento de sistema é atribuir certos objetos a grupos (*clusters*), tal que uma dada função objetivo seja otimizada e as restrições do projeto sejam cumpridas [4]. O seguinte modelo matemático pode representar o problema do particionamento de sistema [22]:

Dado um conjunto de objetos n $O = \{o_1, o_2, ..., o_n\}$, um particionamento $P^k = \{p_1, p_2, ..., p_n\}$, consiste de k grupos $p_1, p_2, ..., p_n$ tal que $p_1 \cup p_2 \cup ... \cup p_n$, e $p_i \cap p_j = \phi$ para $i \leq j$, $j \leq k$, $i \neq j$.

O problema do particionamento é encontrar um particionamento P^k de um conjunto O de n objetos, tal que o custo determinado pela função objetivo $FuncObj(P^k)$ seja mínimo e um conjunto de restrições seja satisfeito. A função objetivo é uma combinação

de métricas que capturam a qualidade de um dado particionamento. O valor de retorno de tal função é chamado custo. A função objetivo usada nos algoritmos de particionamento é mostrada a seguir:

$$FuncObj = k_1 \cdot area + k_2 \cdot retardo + k_3 \cdot potencia \tag{2.1}$$

A função (2.1) não leva em consideração as restrições do sistema. Desde que a maioria das decisões de projeto são conduzidas por restrições então elas devem ser incorporadas nas funções tal que as partições que atendem as restrições são consideradas melhores do que aquelas que não atendem:

$$FuncObj = k_1 \cdot f\,(area, Restr_{area}) + k_2 \cdot f\,(retardo, Restr_{retardo}) + k_3 \cdot f\,(potencia, Restr_{potencia}) \tag{2.2}$$

A *FuncObj* retorna a quantidade pela qual a estimativa da métrica viola as restrições. Se não existir violação então a função retorna zero.

A próxima seção apresenta a definição básica de projeto baseado em plataforma, as características da arquitetura e API da plataforma assim como a plataforma do sistema.

2.2 Projeto Baseado em Plataforma

O conceito de plataforma tem sido discutido por várias décadas e se tornou importante no projeto de sistemas embarcados. Porém, várias definições fizeram com que sua interpretação ficasse confusa [55]. Como um termo genérico, plataformas têm significado de diferentes artefatos para diferentes pessoas. Plataforma também promove uma técnica de reuso comum a qual auxilia substancialmente o projetista do sistema a reduzir o tempo e custo do projeto.

2.2.1 Definição Básica

Uma plataforma é uma camada de abstração que esconde detalhes de várias implementações das camadas inferiores [12]. Por conseguinte, uma plataforma não é uma micro-arquitetura padronizada, mas é uma abstração caracterizada por um conjunto de

restrições na arquitetura [55]. Plataforma pode também ser vista como uma biblioteca de elementos caracterizados por modelos que representam as funcionalidades e oferecem uma estimativa das quantidades físicas que são importantes para o projetista. Neste sentido, a biblioteca contém interconexões e regras que definem a composição dos elementos [12].

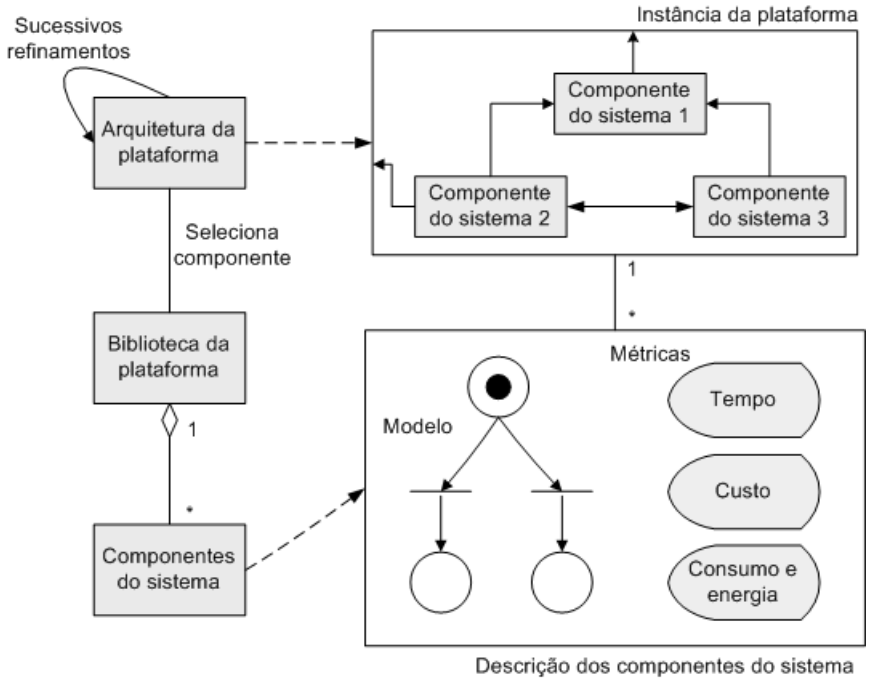

Figura 2.4: Definição de Projeto Baseado em Plataforma.

A composição de elementos e interconexões é chamada de instância da plataforma (*platform instance*) como mostrado na Figura 2.4. O projetista deriva uma instância da arquitetura da plataforma escolhendo um conjunto de componentes da biblioteca da plataforma ou ajustando os parâmetros dos componentes reconfiguráveis da biblioteca da plataforma. Deste modo, o projeto baseado em plataforma não é nem um processo *top-down* nem *bottom-up*, mas sim um processo *meet-in-the-middle*, onde sucessivos refinamentos da especificação atendem as abstrações das potenciais implementações que são capturadas nos modelos dos elementos da plataforma [55].

2.2.2 Características da API da plataforma

Uma interface de programação de aplicativos (*Application Programming Interface*) da plataforma como mostrado na Figura 2.5 é uma abstração de uma variedade de recursos computacionais (p.e., SOTR, Framework, Pilha de Protocolo) e periféricos disponíveis

(p.e., *drivers* de dispositivos). A API é uma representação da arquitetura da plataforma através de camadas de software [12]. A API fornece meios para maximizar o reuso do software e derivar diferentes produtos ou variantes de produtos. A API da plataforma incorpora as funcionalidades que são comuns à maioria dos produtos planejados.

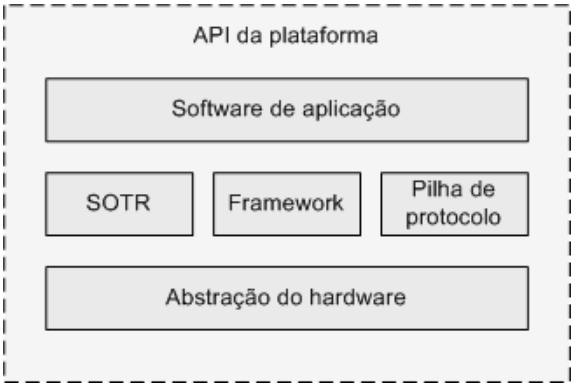

Figura 2.5: Layout da API da plataforma.

A API da plataforma cobre as partes essenciais da arquitetura da plataforma:

- Os módulos programáveis e o subsistema de memória através do SOTR (Sistema Operacional de Tempo Real);
- Pilhas de protocolo
- Framework (p.e., subsistemas de comunicação, biblioteca de interface com o usuário, componentes de middleware);
- Reuso de componentes de software reconfiguráveis.

O SOTR escalona as tarefas de software, gerencia recursos computacionais disponíveis e a comunicação entre eles e a memória do subsistema [55]. A API da plataforma deve ser o mais independente possível do hardware, configurável e extensível para suportar derivações de diferentes configurações para atender os requisitos de diferentes produtos.

2.2.3 Características da arquitetura da plataforma

Produtores de PC desenvolvem os produtos deles rapidamente e eficientemente usando uma plataforma padronizada que tem surgido gradualmente: a arquitetura do conjunto

de instruções do x86, um conjunto completo de barramentos, suporte legado para o controlador de interrupção e a especificação de um conjunto de dispositivo de E/S [55]. Fabricantes de semicondutores trabalham com um outro tipo de plataforma: um circuito integrado flexível que os projetistas possam customizá-lo para uma aplicação específica através da programação de um ou mais componentes de chip. A programação pode significar a customização das portas (*gate arrays*), modificação elétrica (personalização do FPGA - *field programmable gate array*) ou o software que executa em um micro-processador ou um processador de sinal digital (DSP - *Digital Signal Processor*).

Para software embarcado, a plataforma é uma micro-arquitetura fixa que minimiza o custo da realização de máscara do CI, porém é flexível o bastante para funcionar para um conjunto de aplicações tal que o volume de produção permanece alto através do tempo de vida do chip [54]. Deste modo, plataforma de software embarcado deve ser desenvolvida a partir de uma família de chips similares que diferem em um ou mais componentes, mas são baseados no mesmo micro-processador. Desta maneira, ICs usados para sistemas embarcados devem ser desenvolvidos como uma instância de uma arquitetura de plataforma particular. Isto significa que em vez de montá-los a partir de uma coleção de blocos desenvolvidos independentemente de funcionalidades, projetistas os derivam a partir de uma família específica de micro-processadores - possivelmente orientado em direção à classe particular de problemas [54]. O projetista do sistema pode então modificar o CIs estendendo-o ou reduzindo-o.

No geral, plataformas são caracterizadas por componentes programáveis. Sendo assim, cada instância de plataforma derivada a partir da arquitetura da plataforma mantém flexibilidade suficiente para suportar uma gama de aplicações que garante o volume de produção necessário para uma manufatura economicamente viável. A combinação de programabilidade, processadores configuráveis pelo projetista (p.e., processador Tensilica Xtensa) e lógica reconfigurável em tempo de execução (p.e., FPGA - *field-programmable gate arrays*) produz as plataformas "altamente programáveis" [55].

2.2.4 Instâncias da Plataforma

Um projetista deriva uma instância da arquitetura da plataforma escolhendo um conjunto de componentes a partir da biblioteca da platforma ou ajustando os parâmetros dos

componentes reconfiguráveis da biblioteca [55]. Componentes programáveis garantem uma flexibilidade na instância da plataforma que nada mais é do que a habilidade de suportar diferentes aplicações. Programabilidade de software produz uma solução mais flexível, pois é mais fácil de modificar enquanto que hardware configurável executa muito mais rápido e consome muito menos energia do que o software correspondente. Deste modo, o balanceamento da metodologia de projeto baseada em plataforma está entre flexibilidade e desempenho.

2.2.5 Camadas da Plataforma de Sistemas

Uma maneira de pensar sobre plataforma de sistema é considerando uma única plataforma obtida através da junção da camada de alto nível (API da plataforma) e a camada de mais baixo nível (a coleção de componentes que compreendem a arquitetura da plataforma) [54]. O projetista do sistema mapea uma aplicação na representação abstrata, escolhendo a partir de uma família de arquiteturas aquela que otimiza custo, consome menos energia, é mais eficiente e possui flexibilidade. Para que isto ocorra. as ferramentas devem estar ciente de ambas as características da arquitetura e a API. Deste modo, a camada da plataforma do sistema combina duas plataformas e as ferramentas que mapeam uma abstração na outra.

No espaço de projeto, existe um balanceamento óbvio entre nível de abstração da API e o número e diversidade de instâncias da plataforma. Uma API mais abstrata fornece um rico conjunto de instâncias da plataforma, mas também faz com que seja mais difícil escolher uma instância de plataforma ótima e mapeá-la automaticamente [55]. No geral, a API da plataforma é uma camada de abstração pré-definida em cima de um dispositivo complexo ou sistema que pode ser usado para o projeto em um nível mais alto. Um conjunto de chamadas do sistema operacional é também uma plataforma no sentido de que isto fornece uma camada de abstração sob a máquina.

Para escolher a correta arquitetura da plataforma, um modelo de execução da arquitetura da plataforma deve ser exportado para a API da plataforma com o propósito de estimar seu desempenho [55]. Este modelo pode incluir o tamanho, consumo de energia e o tempo (veja Figura 2.6). Contudo, restrições a partir de um nível mais alto da abstração pode também ser passado para níveis mais baixos com o intuito de continuar o processo

de refinamento e satisfazer as restrições de projeto original. Ao longo das estimativas e restrições, funções de custo podem também serem usadas para comparar soluções viáveis.

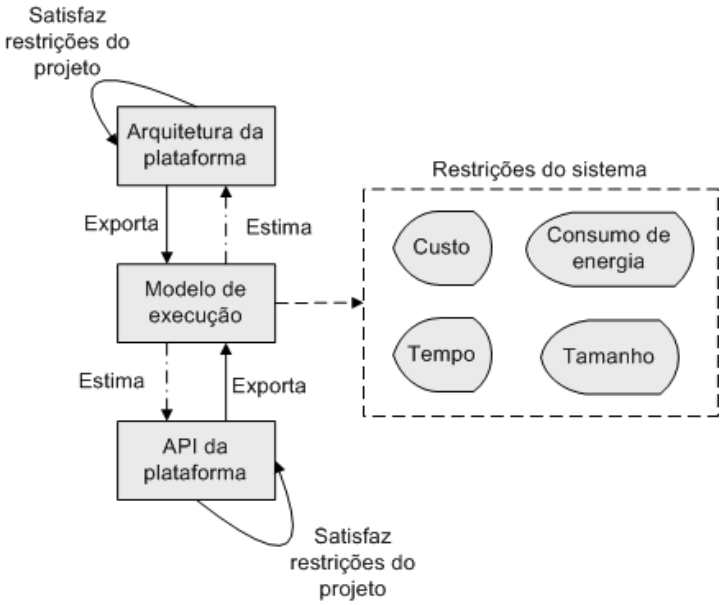

Figura 2.6: Processo para escolha da arquitetura da plataforma.

Em poucas palavras, a camada da plataforma do sistema é um modelo compreensivo que inclui a visão de plataformas a partir de ambas as perspectivas da aplicação e da arquitetura de implementação. A próxima seção descreve as principais práticas dos métodos ágeis XP e Scrum assim como os padrões de desenvolvimento de software ágil que foram integrados na metodologia proposta.

2.3 Métodos e Padrões Ágeis

O surgimento de vários métodos de desenvolvimento de software ágil tem chamado a atenção dos pesquisadores e praticantes de engenharia de software. Estes métodos definem um conjunto de boas práticas para serem usadas em projetos de software. No entanto, é importante salientar que desenvolvimento de software ágil é uma atividade centrada em pessoas que depende fortemente da motivação e habilidades para se gerar um verdadeiro trabalho em equipe. Nesta seção, uma breve descrição dos papéis, responsabilidades e práticas dos métodos XP e Scrum assim como os padrões de desenvolvimento de software ágil são apresentados.

2.3.1 Extreme Programming

O método ágil mais reconhecido é o eXtreme Programming (XP) que enfatiza colaboração entre clientes e desenvolvedores, entrega de software no início das iterações e práticas de desenvolvimento que exigem um certo grau de maturidade do desenvolvedor. XP é bastante orientada a comunicação. Clientes, desenvolvedores e gerentes formam uma equipe trabalhando juntos em uma sala de projeto em comum com o propósito de entregar software rapidamente com alto valor de negócio agregado. XP foi proposto por [9] após vários anos de experiência em desenvolvimento de software.

Papéis e Responsabilidades

Papéis dentro de um time XP não são fixos e rígidos. O objetivo principal é ter todos contribuindo para o sucesso do projeto. Por conseguinte, um programador pode exercer o papel de um arquiteto, um usuário pode se tornar um gerente de produto e assim por diante. Neste cenário, 4 (quatro) principais papéis podem ser identificados e a responsabilidade de cada papel pode ser descrita como segue [9]:

Testadores (testers): Testadores são responsáveis por auxiliar o cliente na escolha e escrita dos testes automatizados em nível de sistema antes de iniciar a fase de implementação e instruir os programadores em técnicas de teste. O papel do testador se desloca para o início do desenvolvimento com o propósito de ajudar a definir e especificar o que fará parte das funcionalidades do sistema.

Programadores (programmers): Programadores são responsáveis por escrever/estimar requisitos (também chamado de *stories cards* na literatura) e tarefas, escrever testes unitários e implementar o código das funcionalidades. Os programadores são também responsáveis por automatizar o processo de desenvolvimento (p.e., geração de versões intermediárias e *smoke tests*[2]) e melhorar gradualmente o projeto do sistema.

Cliente (customer): Clientes são responsáveis por escrever os requisitos e testes de aceitação. O cliente é também responsável por selecionar os requisitos para uma liberação do software e realizar decisões do domínio do projeto durante o desenvolvimento.

[2]O termo *smoke tests* vem da área de hardware e deriva da prática: Após modificar um componente de hardware, senão existir nenhuma fumaça depois de ligar o equipamento então o componente passou no teste. Na área de software, o termo descreve o processo de validar as mudanças no código antes de disponibilizá-lo na árvore de desenvolvimento.

Orientador (coach): O orientador é responsável pela otimização e conscientização do processo. Ele também repassa para a equipe suas experiências em outros projetos e fornece instruções especiais para o quê deveria ser feito para se alcançar os objetivos em comum.

Práticas

XP é composto de doze práticas e algumas das principais incluem: *iterações curtas, integração contínua, teste antes do desenvolvimento, refatoração, integração freqüente de código e projeto incremental* [9].

Integração contínua (Continuous integration): O código é compilado e testado em um processo automatizado toda vez que o mesmo é integrado na árvore de desenvolvimento do projeto (que geralmente está sob controle de versão).

Refatoração (Refactoring): Refatoração é o processo de mudar o sistema de software de tal maneira que o comportamento externo do código não seja alterado e ao mesmo tempo melhore a estrutura interna.

Teste antes do desenvolvimento (Test-Driven Development): Os testes unitários são escritos pelo desenvolvedor antes de escrever o código. Estes testes unitários são testes automatizados que testam partes das funcionalidades do código (p.e., classes, métodos, funções) .

Padronização do Código (Coding standards): Todos os envolvidos no projeto precisam seguir o mesmo estilo de codificação. Isto significa que um formato consistente para o código fonte deve ser seguido e mantido pelos membros da equipe.

Pequenas Versões (Small releases): Entregar software funcional em uma maneira iterativa e incremental para obter uma resposta do cliente com relação às funcionalidades implementadas.

Testes de aceitação: Todas as funcionalidades devem ter testes de aceitação (funcional) automatizado. Os testes de aceitação são escritos em colaboração com o cliente.

Programação em par (Pair programming): Escrever partes do código do sistema com duas pessoas em uma mesma máquina. Programação em par é um diálogo entre duas pessoas simultaneamente programando (analisando, projetando e testando).

Propriedade coletiva (Collective ownership): Significa que todos as pessoas envolvidas na implementação são responsáveis pelo código. Em outras palavras, todos têm permissão

de alterar qualquer parte do código.

Metáfora do sistema (System metaphor): Metáfora do sistema é um conceito de nomeação para classes, métodos e funções que tem como objetivo facilitar o entendimento das funcionalidades do sistema somente a partir de nomes.

Semana de 40 horas (40-hour week): Freqüentes horas extras é considerado como sinônimo de sérios problemas. A idéia é que os programadores não trabalhem mais do que 40 horas na semana, e se existir horas extras em uma dada semana então na seguinte as mesmas não deveriam ser incluídas.

A Figura 2.7 mostra a interação entre as principais práticas XP. XP promove uma abordagem evolucionária para projetar o sistema através das práticas de integração contínua, teste antes do desenvolvimento e refactoring [19]. Deste modo, o principal benefício da abordagem evolucionária é que o sistema evolui em uma maneira iterativa e incremental. Com isso, *riscos* e *incertezas* tendem a ser reduzidos no início do processo de desenvolvimento *(gerenciamento de risco).*

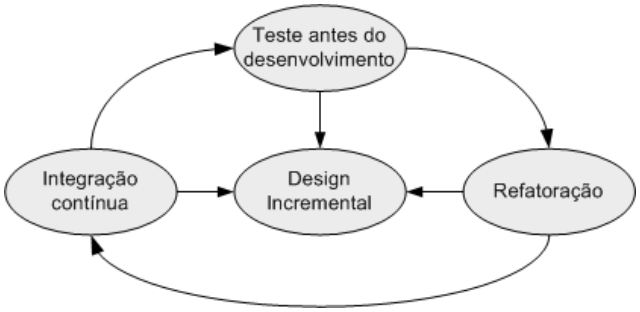

Figura 2.7: Interação entre as práticas da XP.

A próxima seção descreve a metodologia de desenvolvimento ágil Scrum que enfatiza práticas de gerenciamento de projeto.

2.3.2 Scrum

Scrum é uma abordagem simples e clara para gerenciar o processo de desenvolvimento de software. Scrum é baseado na suposição de que variáveis de ambiente (p.e., requisitos) e técnicas (p.e. tecnologias) provavelmente mudem durante o processo de desenvolvimento do produto [46]. A produtividade e qualidade nesta metodologia dependem fortemente da motivação e habilidades das pessoas envolvidas no projeto.

Papéis e Responsabilidades

O processo Scrum consiste basicamente de 3 (três) papéis e a responsabilidade de cada papel é descrita a seguir:

Mestre do Scrum (Scrum Master): Mestre do Scrum é responsável por assegurar que valores do Scrum, práticas e regras sejam seguidos pela equipe. Ele é também responsável for mediar entre o gerenciamento e a equipe do Scrum assim como monitorar progresso e remover impedimentos.

Proprietário do produto (Product Owner): Proprietário do produto é a pessoa que é oficialmente responsável pelo projeto. Esta pessoa cria e prioriza o *backlog*[3] de produto (veja Figura 2.8) e garante que o objetivo do projeto esteja claro para todos os envolvidos. Ele é também responsável por escolher os objetivos para o *sprint* e revisar o sistema com os clientes do projeto no final da iteração.

Equipe do Scrum (Scrum Team): A equipe do Scrum é responsável por trabalhar no *backlog* de *sprint* (veja Figura 2.8). A quantidade de trabalho que será tratada no *sprint* é decidida pela equipe. Eles devem avaliar o que pode ser realizado no *sprint* durante a reunião de planejamento. Sendo assim, a equipe tem autoridade de realizar decisões e solicitar que impedimentos sejam removidos.

Práticas

Scrum é composto por 14 práticas que ajudam a estabelecer um ambiente no qual produtos possam ser desenvolvidos incrementalmente. Estas práticas evoluíram após a aplicação em vários projetos de desenvolvimento [46]. As principais práticas do Scrum são descritas a seguir [33]:

Sprint: A iteração é organizada em um período de 30 dias. A iteração é também chamada de *Sprint*.

Planejamento do Sprint (Sprint planning): Duas reuniões são conduzidas no planejamento do *Sprint*. A primeira reunião, o *backlog* de produto é refinado e priorizado pelos clientes e objetivos para o próximo *Sprint* são escolhidos. Na segunda reunião, a equipe do Scrum avalia como alcançar os objetivos e cria o *backlog* de *Sprint*.

[3]A palavra inglesa *backlog* será usada nesta dissertação no lugar da palavra "pendências"

Revisão do Sprint (Sprint Review): A equipe do Scrum apresenta os resultados obtidos no fim de cada iteração mostrando o software funcional para o proprietário do produto, clientes e outras pessoas interessadas no sistema.

Scrum Diário (Daily Scrum): Reuniões diárias são conduzidas no mesmo lugar e no mesmo horário com perguntas especiais a serem respondidas pela equipe do Scrum.

Geração de versões diárias (Daily builds): Deve haver no mínimo uma integração diária e testes de regressão para todo o código do projeto minimizando assim problemas de integração.

O *backlog* de *sprint* e de produto mencionados acima (veja Figura 2.8) representam dois diferentes artefatos e contém uma lista de funcionalidades, casos de uso, melhorias e defeitos do sistema. O *backlog* de produto pode ser visto como uma lista, em evolução, de prioridade de itens a serem desenvolvidos no sistema enquanto que o *backlog* de *sprint* consiste de um subconjunto do backlog de produto que contém tarefas detalhadas para serem realizados na iteração atual.

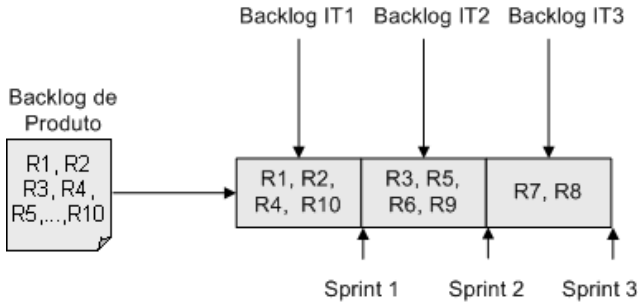

Figura 2.8: Backlog de Sprint e Produto.

Exemplos de itens de backlog de produto incluem:

- Um multiplexador deve ser desenvolvido de modo que permita que dois ou mais sinais sejam transmitidos por um mesmo canal do conversor digital-analógico.
- O driver do teclado deve ser desenvolvido de tal modo que possibilite ao usuário ajustar os parâmetros do dispositivo.
- O sistema deve possibilitar que falhas do sistema sejam armazenadas na memória RAM do micro-controlador.
- O sistema deve mostrar no display a velocidade em RPM no eixo do motor monofásico.

A próxima seção apresenta um breve resumo dos padrões de desenvolvimento de software ágil que influenciaram a metodologia de desenvolvimento proposta.

2.3.3 Padrões para Desenvolvimento de Software Ágil

Algumas práticas usadas em Scrum e XP vieram dos padrões organizacionais de desenvolvimento de software ágil descrito por [16]. Eles estudaram o processo de desenvolvimento de software em 100 diferentes organizações e estruturaram as práticas comuns em quatro diferentes linguagens de padrão[4]. Estes padrões organizacionais podem ser combinados com os métodos ágeis XP e Scrum com o propósito de estruturar o processo de desenvolvimento de software das organizações. Estes padrões estão divididos em quatro linguagens de padrão da seguinte maneira: A linguagem de padrões de gerenciamento de projeto *(Project Management Patterns)* fornece um conjunto de práticas que auxiliam a organização a gerenciar o desenvolvimento do produto, esclarecer os requisitos do produto, coordenar as atividades do projeto, gerenciar a geração de versões intermediárias do produto, e manter a equipe focada nos objetivos do projeto.

A linguagem de padrões para crescimento gradual *Piecemeal Growth Patterns* fornece um conjunto de padrões que auxiliam a organização a definir a quantidade de membros da equipe por projeto, assegurar e manter a satisfação do cliente, comunicar os requisitos do sistema, e assegurar uma visão comum para todos os envolvidos na equipe de desenvolvimento do produto. A linguagem de padrões estilo organizacional *(Organizational Style Patterns)* fornece um conjunto de padrões que auxiliam a organização a eliminar excesso de comunicação e latência nos projetos, assegurar que a estrutura da organização esteja compatível com a arquitetura do produto, organizar o trabalho para desenvolver produtos geograficamente distribuídos, assegurar que as necessidades do mercado sejam atendidas e agrupar atividades que estejam relacionadas.

A linguagem de padrões de codificação e pessoas *(people and code patterns language)* fornece um conjunto de padrões que auxiliam a organização a definir e manter o estilo de arquitetura do produto, assegurar que o arquiteto esteja materialmente envolvido

[4]A linguagem de padrões define soluções para um conjunto de problemas em um dado contexto da aplicação. Cada padrão é descrito da seguinte maneira: o contexto onde o padrão é aplicado, a solução do problema, as forças que limitam a aplicação do padrão, os padrões relacionados, usos conhecidos e o contexto resultante.

na implementação e atribuir funcionalidades a equipes de desenvolvimento em projetos não-triviais. A linguagem de padrões para gerenciamento de configuração *(Software Configuration Management Patterns Language)* não fazem parte dos padrões organizacionais propostos por [16]. Esta linguagem fornece um conjunto de padrões que auxiliam a equipe de desenvolvimento a definir mecanismos para gerenciamento das diferentes versões do produto de trabalho, desenvolver código em paralelo com outros desenvolvedores e integrar com o estado atual da linha de desenvolvimento [10].

2.4 Resumo

Este capítulo apresentou sistemas embarcados de tempo real a partir de diferentes perspectivas com ênfase na diferença fundamental entre sistemas de tempo real crítico e brando, ambientes e características de tais sistemas que influenciam o ciclo de desenvolvimento. Foram também apresentados os aspectos básicos e técnicas para a tarefa de particionamento de hardware/software que tem como objetivo implementar sistemas embarcados de tempo real satisfazendo um conjunto de restrições do *design*, tais como custo monetário, requisitos temporais, tamanho e consumo de energia.

Na seção seguinte, a metodologia de projeto baseada em plataforma que representa um conceito importante para projeto de sistemas eletrônicos foi apresentada. O conceito de plataforma tem como objetivo promover uma técnica de reuso e pode ser vista como uma biblioteca de elementos caracterizados por modelos que representam suas funcionalidades. Além disso, oferece uma estimativa das quantidades físicas que são de extrema importância para o projetista. Deste modo, o projetista deriva a instância da plataforma escolhendo um conjunto de componentes de uma dada biblioteca. Isto resulta na plataforma do sistema que consiste de uma única plataforma que é obtida pela junção da API e arquitetura da plataforma.

Além disso, este capítulo introduziu os métodos ágeis XP, Scrum e os padrões organizacionais e de gerenciamento de configuração de software nomeados nesta dissertação como padrões ágeis. Foi mostrado que XP é o método ágil mais reconhecido que enfatiza colaboração, entrega de software no início das iterações e práticas de desenvolvimento que exigem certo nível de maturidade do desenvolvedor. XP foi criado por [9] e é baseado

principalmente na teoria das restrições proposto por [24]. Scrum é uma abordagem simples e direta para gerenciar o processo de desenvolvimento de software.

Scrum foi criado por [46] e é principalmente baseado no modelo de controle de processo empírico que usa mecanismo de retro-alimentação para monitorar e adaptar as atividades. Os padrões ágeis propostos por [16, 10] fornecem práticas para estruturar o processo de desenvolvimento de software de uma organização. Quando XP, Scrum e padrões ágeis são combinados, eles cobrem várias áreas do ciclo de vida do desenvolvimento de sistemas.

Capítulo 3

Trabalhos Relacionados

Equipes de desenvolvimento de software embarcado geralmente não utilizam metodologias de desenvolvimento ou qualquer outro conceito mais complexo de engenharia de software [25]. Existem diferentes razões que explicam este fato, mas a principal delas é a falta de maturidade dos desenvolvedores com relação às práticas de engenharia de software. No entanto, foram identificadas algumas metodologias que permitiram avaliar o estado da arte neste contexto e que também serviu como base para a definição da metodologia proposta.

A seguir são citados alguns trabalhos significativos e que estão de algum modo relacionado com o tema proposto nesse trabalho: (i) Métodos ágeis aplicados a desenvolvimento de sistemas embarcados, (ii) metodologia de projeto baseada em plataforma, (iii) projeto concorrente de hardware/software, (iv) UML para especificação e projeto de sistemas embarcados.

3.1 Métodos Ágeis para Sistemas Embarcados

Existem poucos trabalhos publicados na área de métodos ágeis para sistemas embarcados até o momento da escrita desta dissertação de mestrado. Um dos poucos artigos nesta área descreve a experiência em aplicar práticas ágeis no desenvolvimento de *firmware* para a família de processadores da Intel® Itanium® [25]. Neste artigo, *Greene* identifica as práticas ágeis que a equipe dele aplicou com sucesso, porém não leva em consideração o desenvolvimento relacionado ao hardware, uma das principais partes deste tipo de desen-

volvimento. Greene somente mencionou que uma outra equipe da Intel estava aplicando os conceitos ágeis e eles obtiveram bons resultados. Mesmo assim, os comentários obtidos a partir desta aplicação dos conceitos ágeis foram bastante proveitosos durante a definição da abordagem proposta, pois este artigo comenta os benefícios de usar conceitos ágeis no contexto fora do desenvolvimento de software orientado a objetos.

Manhart and Schneider [35] relataram uma experiência de sucesso na indústria quando parcialmente adotaram métodos ágeis na produção de software para sistemas embarcados. Na verdade, eles realizam pequenas modificações em um processo de desenvolvimento de software estabelecido para o ramo automotivo adotando alguns princípios e práticas ágeis com o propósito de adaptar o processo deles às novas necessidades de flexibilidade e produção de software em alta velocidade. Como foi apontado no artigo [35], várias outras áreas podem se beneficiar dos experimentos deles, no entanto os autores não apresentaram qualquer resultado de medida que poderia provar as expectativas.

O artigo [45] enfatiza técnicas ágeis que os autores utilizaram em um projeto de sistema embarcado de tempo real. Schooenderwoert comenta as dificuldades que a equipe e gerenciamento enfrentaram na transição para o uso da metodologia de desenvolvimento XP (leia seção 2.3.1). Um ponto forte do artigo é a descrição detalhada das técnicas de teste que foram usadas. De acordo com Schooenderwoert, o uso das técnicas descritas no artigo manteve uma baixa taxa de defeitos no software. Em três anos de projeto, houve somente cinqüenta defeitos e a lista de defeitos abertos em cada iteração do projeto nunca passava de dois itens. Segundo Schooenderwoert, a equipe passou grande parte do tempo adicionando valor ao produto em vez de lutar para consertar defeitos. No entanto, os autores não apresentam nenhuma métrica (p.e., desempenho, consumo de energia, tamanho de código) que possa garantir que as técnicas propostas realmente conduzem a um sistema eficiente e com baixo custo.

Ronkainen e Abrahamsson avaliam a possibilidade de usar técnicas de desenvolvimento ágil no ambiente de software embarcado [44]. Com isso, eles definem requisitos para novos métodos ágeis com o intuito de facilitar o desenvolvimento do software embarcado. Estes requisitos incluem *(i)* maior ênfase na arquitetura de hardware/software, *(ii)* refatoração deveria ser integrada com um sistema de gerenciamento de configuração, *(iii)* técnicas para mensurar a maturidade do código em diferentes fases do desenvolvimento,

e *(iv)* técnicas para o desenvolvimento de casos de teste que levam em consideração não somente a corretude lógica, mas também temporal da aplicação. Embora este artigo seja totalmente conceitual, os requisitos para novos métodos ágeis serviram como base para o desenvolvimento da metodologia proposta.

3.2 Metodologia de Projeto Baseada em Plataforma

O aumento da complexidade e redução no tempo de desenvolvimento fez com que projetistas de sistemas embarcados escolhessem por implementações flexíveis e fabricantes de semicondutores fornecessem chips que possam funcionar para uma ampla gama de produtos. Este alinhamento de projestistas e fabricantes resultou no surgimento da metodologia de projeto baseada em plataforma na qual o reuso e programabilidade são fatores chaves [55, 54]. Vicentelli propõe uma rigorosa metodologia para desenvolvimento de software embarcado e projeto baseado em plataforma. De acordo com [55], uma metodologia de projeto de sistema deve *(i)* tratar de problemas de integração de fornecedores que nada mais é do que a conexão entre criadores de propriedade intelectual (IP - *Intellectual Property*), fabricantes de semicondutores e desenvolvedores de software, *(ii)* deve considerar métricas que governam o *projeto* de sistemas embarcados incluindo custo, peso, tamanho, consumo de energia e requisitos de desempenho, *(iii)* deve trabalhar em todos os níveis de abstração, desde a concepção até o software e implementação do silício e empacotamento com avaliação de balanceamento entre eficiência e corretude e *(iv)* favorecer o reuso através da identificação de requisitos para operação *plug-and-play* [55].

De acordo com Vicentelli, integrar o conceito de projeto baseado em plataforma com um fluxo de desenvolvimento de software embarcado que vai desde a especificação até implementação, solicitará ainda pesquisa, ferramenta, metodologia de desenvolvimento e aplicações experimentais. Além disso, é necessária intensa cooperação entre fabricantes de semicondutores e centros de desenvolvimento. Centros de desenvolvimento devem melhorar a produtividade e qualidade de software enquanto enfrentam complexidade em ambos os espectros: requisitos de produto do cliente e o hardware que eles usam para melhorar desempenho e atender restrições do sistema [55]. O trabalho desenvolvido pelo centro de pesquisa em silício *Gigascal*e e Universidade da Califórnia em Berkeley tratam de

vários detalhes para alinhar os fabricantes de semicondutores e centros de desenvolvimento de produto. Embora os artigos [55, 54] sejam totalmente conceituais, eles serviram como base para o desenvolvimento da metodologia proposta nesta dissertação.

3.3 Projeto Concorrente de Hardware/Software

A metodologia proposta por Gajski [21, 20] tem como objetivo desenvolver sistemas embarcados através da descrição formal das funcionalidades do sistema em uma linguagem executável em vez de uma linguagem natural. A especificação executável é refinada através das tarefas de *projeto* do sistema que são: alocação, particionamento e refinamento. Estimadores são também usados com o intuito de explorar as alternativas de *projeto*. Como os componentes do sistema são definidos formalmente então componentes são implementados somente pela compilação da descrição funcional do componente em código de máquina. Esta metodologia foi aplicada a vários projetos de sistemas embarcados de tempo real e também influenciou a metodologia proposta nesta dissertação. Porém, esta metodologia assume que todos os requisitos são obtidos antes de aplicar os algoritmos de particionamento.

Uma das tarefas cruciais em projeto concorrente de hardware/software é o particionamento que consiste essencialmente em decidir quais componentes do sistema deveriam ser implementados em hardware ou em software (leia seção 2.1.3). No passado, particionamento de hardware/software era realizado manualmente. Porém, a medida que a complexidade do sistema aumenta, este método manual torna-se inviável e esforços de pesquisa tem sido direcionados a automação do particionamento. No trabalho [4], os autores apresentam uma análise da complexidade do algoritmo de particionamento e fornecem duas abordagens de solucionar o problema do particionamento: programação linear inteira (ILP) e algoritmo genético. Os autores mostraram através de experimentos que a solução baseada em ILP funciona eficientemente para grafos com algumas centenas de vértices e produz uma solução ótima, enquanto que o algoritmo genético fornece uma solução aproximada e funciona eficientemente com grafos com alguns milhares de vértices.

O surgimento de plataformas de um único chip incorporando um micro-processador e FPGA tem recentemente realizado o particionamento de hardware/software muito mais

atrativo. Tais tipos de plataformas produzem uma comunicação mais eficiente entre o micro-processador e FPGA do que o projeto de dois chips, resultando assim em melhoria no desempenho e redução no consumo de energia. Stitt propõe uma abordagem de particionamento de hardware/software que consiste essencialmente em monitorar o programa binário executando em um micro-processador, detectar regiões críticas do código, decompilar estas regiões, sintetizá-las para o hardware, colocar e rotear na lógica configurável do chip e atualizar o código binário para comunicar com a lógica [51]. No entanto, umas das desvantagens desta abordagem é que o o projetista não tem nenhuma interação com os resultados do particionamento (é realizado de maneira transparente).

3.4 UML para Sistemas Embarcados

Chen [13] apresenta uma técnica de especificação para o projeto de sistemas embarcados que trata de aspectos relacionados à estrutura e comportamento do sistema, verificação da curretude funcional e checagem do atendimento a restrições. De acordo com Chen, análise de custo e desempenho do sistema depende fortemente da arquitetura da plataforma selecionada e, por conseguinte solicita ferramentas e modelos para uma definição formal da implementação dos recursos da plataforma e a qualidade dos serviços oferecidos. Com este objetivo em mente, Chen apresenta um novo perfil da UML, *UML Platform*, que introduz novos blocos (p.e. novos estereótipos) para representar os serviços e recursos da plataforma. Porém, para que possa existir ferramentas e métodos para auxiliar o projeto de sistemas embarcados usando a *UML Platform*, é vital que a indústria adote um perfil de plataforma padrão na UML. De outro modo, torna-se difícil desenvolver ferramentas que automatizem o fluxo da especificação até a implementação.

Sistemas embarcados modernos possuem certas características que demandam novas abordagens para especificação, *projeto* e implementação. Sendo assim, Martin [36] avalia os requisitos para projetos em nível de sistema de sistemas embarcados e fornece uma visão geral das extensões necessárias para a UML. Em particular Martin discute como o conceito de projeto baseado em plataforma se relaciona com a abordagem de desenvolvimento usando UML. No entanto, como o estado atual da linguagem ainda não está completo o suficiente para construir ferramentas, metodologias e fluxos, Martin aponta que o conceito

de plataforma e UML, complementado por trabalhos mais aprofundados em metodologias, podem fornecer conceitos adicionais para um fluxo compreensivo de *projeto* de software de sistemas embarcados baseado em UML.

Um novo perfil da UML 2.0 desenvolvido pela *Tampere University of Technology*, chamado perfil TUT, é apresentado por Kukkala [32]. O perfil TUT define um conjunto de estereótipos para estender as meta classes da UML assim como práticas de *projeto* para descrever aplicações, plataformas e o mapeamento delas. Este perfil é especialmente voltado a implementação de sistemas usando somente a descrição da UML 2.0. Para este propósito, o perfil TUT é usado com um conjunto de ferramentas e uma plataforma de hardware que é composta do processador NIOS da Altera. Embora este artigo tenha mostrado um estudo de caso com um terminal WLAN customizado, o perfil TUT deve ainda melhorar a parametrização dos elementos da plataforma e a especialização dos estereótipos.

3.5 Resumo

A Tabela 3.1 apresenta o framework de avaliação dos métodos sob investigação que incluem Projeto Concorrente de HW/SW, Projeto Baseado em Plataforma, eXtreme Programming, Scrum, Padrões Organizacionais e o Método Desejado que é apresentado em detalhes na Seção 4. Cada método foi avaliado usando os seguintes critérios: ciclo de vida, gerenciamento de projeto, flexível, requisitos não-funcionais, desenvolvimento de hardware, dependabilidade, guia concreto e resultados empíricos. A avaliação foi baseada no nível de evidências encontrado nos métodos analisados. Este nível de avaliação pode ser classificado como: ++→alto, +→baixo e 0→nenhum.

O framework de avaliação foi adaptado do artigo [1]. Os critérios flexibilidade, requisitos não-funcionais, desenvolvimento de hardware e dependabilidade foram adicionados com o propósito de analisar as principais características de metodologias para sistemas embarcados. O critério ciclo de vida define uma seqüência de atividades que uma organização deve utilizar para conceber, projetar, implementar, integrar, testar e validar um produto. O critério gerenciamento de projeto define atividades para possibilitar uma execução apropriada das tarefas do desenvolvimento do produto. O critério flexibilidade

Tabela 3.1: Framework de Avaliação

Método/Critério	Co-design	PBD	XP	Scrum	Padrões	Desejado
Ciclo de vida	+	+	+	+	++	++
Geren. de projeto	0	0	+	++	++	++
Flexível	+	++	++	++	++	++
Req. não-funcionais	++	++	+	+	+	++
Desenv. hardware	++	++	0	0	0	++
Dependabilidade	++	++	+	+	+	+
Guia Concreto	++	+	++	++	++	++
Result. Empíricos	++	+	+	0	0	+

avalia métodos para se adaptar às mudanças durante o ciclo de desenvolvimento.

O critério requisitos não-funcionais avalia práticas para tratar das métricas de tempo de execução, uso de memória e consumo de energia. O critério desenvolvimento de hardware avalia práticas para auxiliar o projetista no desenvolvimento/integração dos elementos físicos do sistema. O critério dependabilidade avalia técnicas para garantir a confiabilidade, disponibilidade e manutenabilidade. O critério guia concreto avalia se o método possui realmente práticas que definem como aplicar a metodologia em vez de contar com princípios totalmente abstratos. Finalmente, o critério resultados empíricos avalia a evidências empíricas do uso do método no desenvolvimento de produtos.

Conforme pode ser observado na tabela 3.1, o método desejado possui um baixo nível de evidências nos critérios dependabilidade e resultados empíricos. O método desejado garante a corretude lógica e temporal do sistema através do uso de práticas de teste unitário e funcional. Embora a metodologia forneça um guia concreto de como exercitar os caminhos de código das funções do sistema, o mesmo dependente fortemente do desenvolvedor seguir realmente as práticas.

Um outro ponto é que nós validamos o método desejado somente nos domínios de aplicação de equipamentos médicos e sistemas de controle discreto. Além disso, todos os projetos que nós desenvolvemos podem ser considerados como pequenos. Sendo assim, deve-se ainda validar a metodologia proposta para outros domínios de aplicação, como por exemplo, telecomunicações, sistemas eletro-eletrônicos, instrumentação científica entre outros.

A diferença da metodologia desejada comparada com outras metodologias pode ser

descrita da seguinte maneira: *(i)* a metodologia proposta tem como objetivo balancear flexibilidade e desempenho adotando plataformas altamente programáveis, *(ii)* técnicas de estimativa e particionamento de hardware/software são usadas com o propósito de explorar as alternativas de *projeto* e atender as restrições do sistema, *(iii)* através do uso da abordagem iterativa e incremental, o desenvolvimento do produto pode ser quebrado em uma seqüência de iterações e implementado um uma maneira incremental, *(iv)* a medida que as funcionalidades do sistema crescem iteração a iteração então a metodologia proposta oferece claramente um processo iterativo onde o projetista pode validar o particionamento da especificação de um sistema produzido pelos algoritmos, *(v)* e por último, mas não menos importante, a metodologia proposta adota um planejamento adaptativo que possibilita mudanças nos requisitos mesmo tarde no processo de desenvolvimento.

O próximo capítulo apresenta a metodologia de desenvolvimento de HW/SW ágil proposta.

Capítulo 4

TXM: Uma Metodologia de Desenvolvimento de HW/SW

Este capítulo descreve uma metodologia de desenvolvimento ágil que foi desenvolvida durante esta pesquisa de mestrado. Esta metodologia define papéis, responsabilidades e processos para serem aplicados em projetos de sistemas embarcados de tempo real.

Com este objetivo em mente, este capítulo está dividido em quatro seções da seguinte maneira: visão geral dos processos da plataforma, papéis e responsabilidades, descrição dos processos e ciclo de vida. A primeira seção fornece os principais elementos da metodologia proposta. A segunda seção define quatro diferentes papéis envolvidos no processo e descreve a responsabilidade de cada papel. A terceira seção apresenta uma visão geral dos processos e os descreve em termos de atividades, papéis responsáveis, artefatos produzidos e ferramentas usadas. Finalmente, a Seção 4 enfatiza o ciclo de vida do desenvolvimento da metodologia proposta.

4.1 Visão Geral dos Processos da Metodologia

A metodologia de desenvolvimento proposta chamada de TXM tem como objetivo definir papéis e responsabilidades e fornecer processos, prática, ciclo de vida e ferramentas para serem aplicados em projeto de sistemas embarcados de tempo real. A Figura 4.1 mostra uma visão geral dos processos da metodologia que contém essencialmente três diferentes grupos de processos que deveriam ser usados durante o ciclo de desenvolvimento, são eles:

plataforma do sistema, desenvolvimento e gerenciamento do produto.

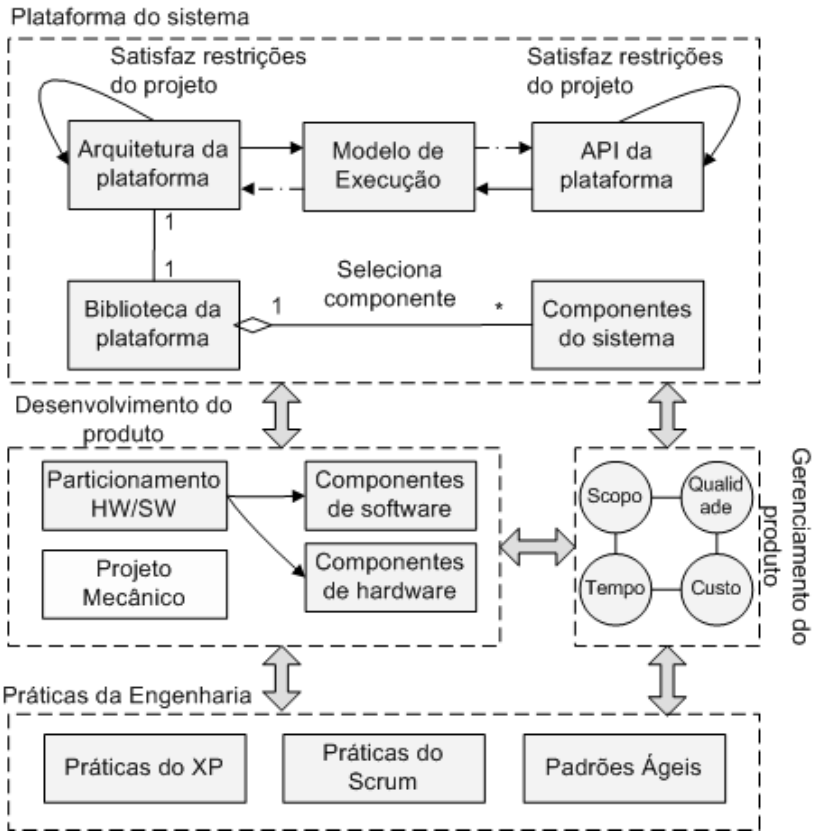

Figura 4.1: Visão Geral dos Processos da Metodologia Proposta.

O grupo de processos da plataforma do sistema tem como objetivo instanciar a plataforma para um dado produto. Isto significa que o projetista do sistema deve escolher a partir da biblioteca da plataforma os componentes do sistema que farão parte da arquitetura e API da plataforma. Depois disso, o projetista do sistema tem ainda a possibilidade de customizar a arquitetura e API da plataforma com o propósito de atender as restrições da aplicação. O processo de customização é realizado pela programação dos micro-processadores e FPGAs integrados na plataforma. O processo de customização é realizado através de sucessivos refinamentos em uma maneira iterativa e incremental dentro da metodologia proposta.

O grupo de processos de desenvolvimento do produto oferece práticas para desenvolver os componentes da aplicação e integrá-los na plataforma. As funcionalidades que constituem o produto são particionadas em elementos de hardware ou de software da plataforma. Os algoritmos de particionamento usados para realizar esta tarefa levam em

consideração consumo de energia, tempo de execução e tamanho da memória dos componentes da aplicação e drivers. O projeto mecânico faz também parte deste grupo de processos, mas está fora do escopo desta dissertação de mestrado. A técnica de particionamento é também aplicada em uma maneira iterativa e incremental.

Os parâmetros de custo, qualidade, tempo e escopo são monitorados e controlados pelo grupo de gerenciamento de produto. Estes parâmetros também influenciam a plataforma do sistema e os grupos de processos do desenvolvimento do produto. Quando o projeto inicia com um plano de projeto inviável que necessita de ações corretivas para serem realizadas então este grupo de processos tem como objetivo pôr o projeto de volta nos trilhos e assegurar que os parâmetros do projeto sejam atendidos. O grupo de processos de gerenciamento de produto consiste das práticas promovidas pelo método Ágil Scrum assim como os padrões ágeis descritos no Capítulo 2. As próximas seções estão relacionadas com a descrição dos grupos de processos, papéis, responsabilidades e o ciclo de vida dos processos da metodologia proposta.

4.2 Grupos de Processos

Esta seção está relacionada com a descrição dos grupos de processos que compõem a metodologia proposta. Sendo assim, esta seção está dividida em três diferentes grupos da seguinte maneira: platforma do sistema, grupos de gerenciamento e desenvolvimento de produto. Estes grupos de processos são descritos nas subseções seguintes.

4.2.1 Grupo de Processos da Plataforma do Sistema

O grupo de processos da plataforma do sistema é composto dos seguintes processos: *requisitos do produto, plataforma do sistema, linha de produto e otimização do sistema.* A Figura 4.2 mostra os processos que estão relacionados ao grupo de processos da plataforma do sistema. O processo de *requisitos do produto* tem como objetivo obter os requisitos do sistema (funcional e não funcional) que são relevantes para determinar a plataforma do sistema no qual o produto será desenvolvido. O processo de *instanciar a plataforma* auxilia a equipe de desenvolvimento a definir a plataforma do sistema através do uso de um conjunto de ferramentas de *projeto* e *benchmarks.*

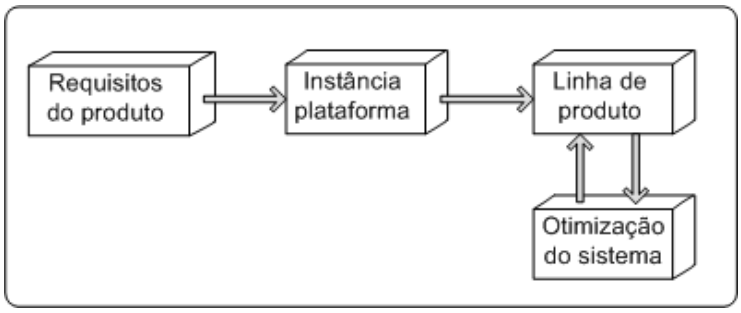

Figura 4.2: Grupo de Processos da Plataforma do Sistema.

Depois de definir a plataforma do sistema, o processo de *linha de produto* auxilia a equipe de desenvolvimento estruturar o repositório no qual os componentes da plataforma do sistema estarão disponíveis para o desenvolvimento do produto. Este processo também possibilita que a equipe de desenvolvimento implemente e integre as funcionalidades do sistema na linha de desenvolvimento do produto, o processo *otimização do sistema* fornece atividades para assegurar que as variáveis do sistema tais como tempo de execução, consumo de energia, tamanho da memória de dados e programa satisfaçam as restrições da aplicação.

4.2.2 Grupo de Processos de Desenvolvimento de Produto

O grupo de processos de desenvolvimento de produto é composto dos seguintes processos: *implementação da funcionalidade, integração de tarefa, refatoração do sistema e otimização do sistema.* A Figura 4.3 mostra os processos que estão relacionados ao grupo de processos do desenvolvimento do produto. O processo de *implementação da funcionalidade* assegura que os casos de teste são criados para todas as funcionalidades do produto. Este processo tem como objetivo melhorar a qualidade do produto e reduzir a complexidade das funções. O processo de *integração de tarefas* fornece meios para integrar novas funcionalidades implementadas na linha de desenvolvimento do produto sem ter que forçar outros membros da equipe de trabalhar ao redor disto.

O processo refatoração de sistema apoia a equipe de desenvolvimento na identificação de oportunidades para melhorar o código e mudá-lo sem alterar seu comportamento externo. Depois de refatorar o código, o processo de *otimização do sistema* permite que a equipe de desenvolvimento otimize pequenas partes do código através do uso de fer-

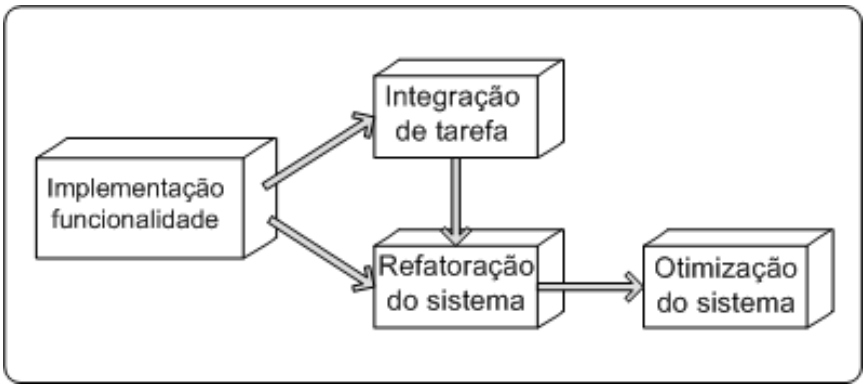

Figura 4.3: Grupo de Processos de Desenvolvimento do Produto.

ramentas de *profiler* que monitoram o programa e informam onde, por exemplo, está se consumindo energia, tempo e espaço de memória [39]. Este processo garante que métricas de software atendam as restrições do sistema.

4.2.3 Grupo de Processos de Gerenciamento de Produto

O grupo de processo de gerenciamento de produto é composto dos seguintes processos: *requisitos do produto, gerenciamento do projeto, rastreamento do produto, requisitos do sprint, linha de produto e prioridade de implementação.* A Figura 4.4 mostra os processos que estão relacionados ao grupo de gerenciamento de produto. O processo *requisitos do produto* (que também pertence ao grupo de processos de plataforma do sistema) tem como objetivo obter os requisitos do sistema (funcional e não funcional) que devem fazer parte do produto. O processo *gerenciamento de projeto* permite que a equipe de desenvolvimento implemente os requisitos do sistema através do gerenciamento do *backlog* de *sprint* e produto, coordenação de atividades, geração de versões intermediárias e rastreamento dos bugs do produto.

O processo de *rastreamento de bug* permite que o líder do produto gerencie o ciclo de vida dos artefatos do projeto (defeitos, tarefas e melhorias) e forneça as informações necessárias sobre a qualidade do produto através das notas de liberação (*release notes*) para o usuário final. O processo *requisitos do sprint* permite que a equipe de desenvolvimento analise, avalie e estime as funcionalidades do sistema antes de iniciar um novo *sprint* do projeto. Esta informação está inclusa no backlog de *sprint* que auxiliará a equipe de desenvolvimento a particionar as funcionalidades do sistema em hardware ou

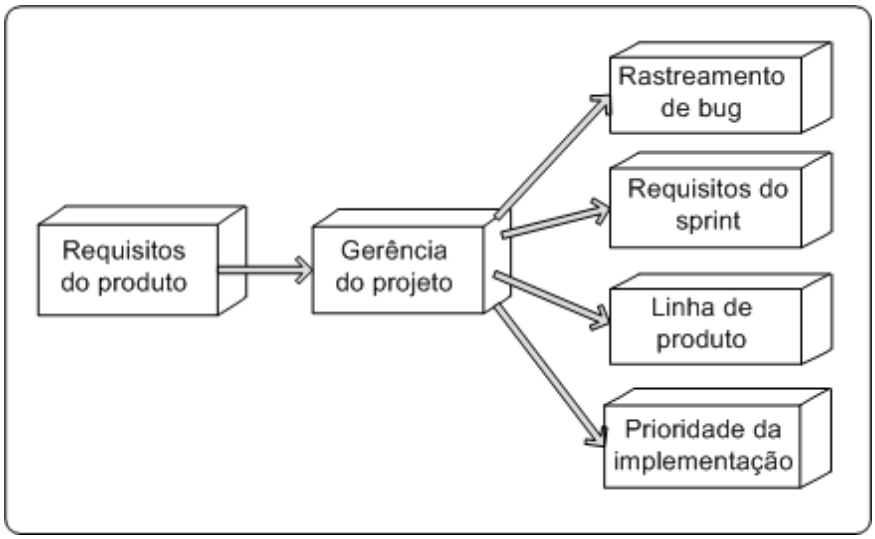

Figura 4.4: Grupo de Processos de Gerenciamento de Produto.

software antes de iniciar o sprint.

O processo de *linha de produto* garante que as funcionalidades do sistema implementadas durante o *sprint* serão integradas na linha de desenvolvimento do produto. Este processo também auxilia a equipe de desenvolvimento a liberar novas versões do produto no mercado. O processo de *prioridade de implementação* apoia o líder do produto a gerenciar qualquer tipo de interrupção que pode impactar os objetivos do projeto. Este processo garante que as tarefas do projeto serão 100 porcento terminadas depois de iniciadas.

4.3 Papéis e Responsabilidades

Os processos que serão descritos na próxima seção envolvem essencialmente quatro diferentes papéis e a responsabilidade de cada papel é descrita abaixo da seguinte maneira:

Dono da Plataforma (Platform Owner): O dono da plataforma é a pessoa que é oficialmente responsável pelos produtos que derivam de uma dada plataforma. Esta pessoa é responsável por definir qualidade, tempo de desenvolvimento e custos de um produto. Ele deve também criar e priorizar o *backlog* de produto e assegurar que isto seja visível para todos os envolvidos no projeto. O dono da plataforma é também responsável por escolher os objetivos do *sprint* e revisar o produto com os clientes no fim da iteração.

Líder do Produto (Product Leader): O líder do produto é responsável pela implementação, integração e teste do produto assegurando que qualidade, tempo de desen-

volvimento e custo definidos pelo dono da plataforma sejam atendidos. Ele é também responsável por mediar entre o gerenciamento e equipe de desenvolvimento assim como monitorar o progresso e remover impedimentos. Se o produto é composto por vários componentes e envolve várias equipes de desenvolvimento então o líder do produto deve trabalhar muito próximo ao líder de funcionalidades (*feature leader*)

Líder de Funcionalidades (Feature Leader): Líder de funcionalidades é responsável por gerenciar, controlar e coordenar projetos de subsistema, projetos de integração, parceiros externos que contribuem para um conjunto definido de funcionalidades. O líder de funcionalidade também rastrea o progresso e status do desenvolvimento das funcionalidades (entregas, status da integração, status do teste, defeitos e solicitações de mudanças) e informa o status para o líder do produto.

Equipe de desenvolvimento: A equipe de desenvolvimento que pode consistir de programadores, arquitetos e testadores são responsáveis por trabalhar no desenvolvimento do produto. A quantidade de trabalho que será tratada nas iterações é de inteira responsabilidade da equipe. Eles devem avaliar o que deve ser realizado durante o desenvolvimento do produto. Por conseguinte, a equipe de desenvolvimento tem autoridade de realizar qualquer decisão junto ao líder de produto e solicitar que impedimentos sejam removidos.

Se o produto a ser desenvolvido é pequeno, isto é, se é composto de poucos componentes e não solicita outras equipes de desenvolvimento para implementar as funcionalidades do produto então um líder de produto e a equipe de desenvolvimento é suficiente para o desenvolvimento do produto. A Figura 4.5 mostra os papéis envolvidos nos processos e seus respectivos níveis hierárquicos.

Contudo, se o produto é composto por vários componentes e solicita outras equipes de desenvolvimento para implementar as funcionalidades do produto então um outro papel (líder de funcionalidades) deve ser envolvido nos processos. Neste contexto, um líder de produto requer líderes de funcionalidades para gerenciar, controlar e coordenar projetos de componentes. Cada líder de funcionalidade é responsável por uma equipe de desenvolvimento e pode depender dos serviços fornecidos pelos componentes desenvolvidos por outras equipes de desenvolvimento. Deste modo, para projetos grandes ou médios, um líder de produto e vários líderes de funcionalidades e equipes de desenvolvimento podem ser envolvidos nos processos.

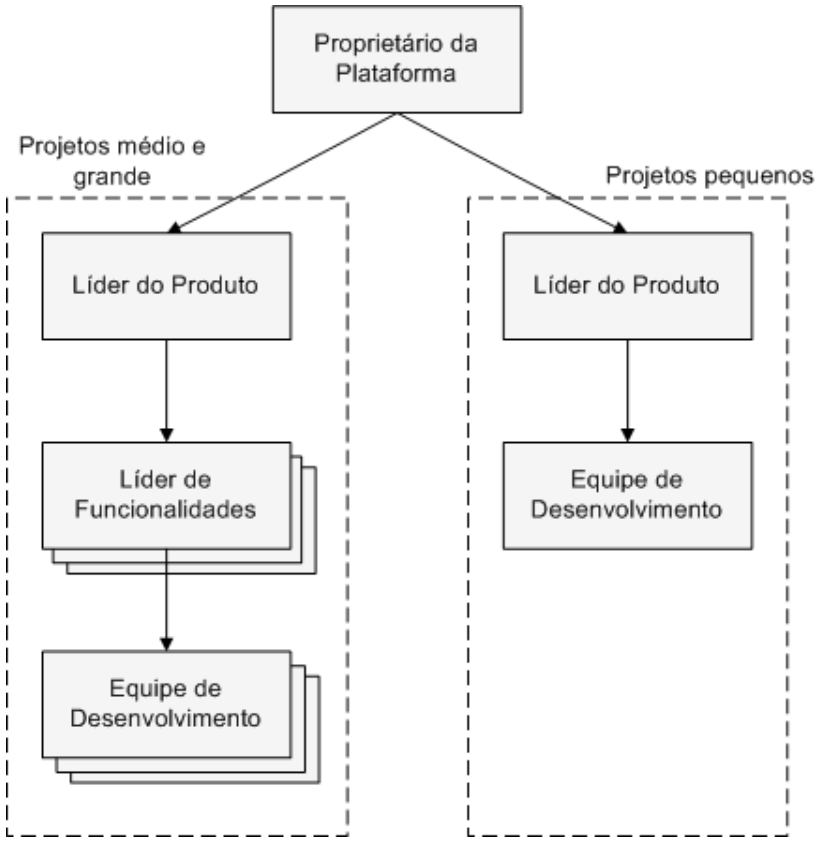

Figura 4.5: Papéis Envolvidos no Processo.

4.4 Ciclo de Vida dos Processos

A meotodologia proposta consiste de cinco fases (como mostrado na figura 4.6): *Exploração, Planejamento, Desenvolvimento, Liberação e Manutenção.* Na *fase de exploração* (ou exploração da especificação), os clientes fornecem requisitos para a primeira versão do produto. Estes requisitos são incluídos no backlog de produto pelo dono da plataforma. Depois disso, o líder de produto e proprietário da plataforma estimam o esforço de implementação dos requisitos, com itens que não ultrapassem 3 pessoas-dias de esforço[1]. Nesta fase, a equipe de desenvolvimento identifica as restrições da plataforma e aplicação e estima as métricas do sistema baseado nos itens de backlog de produto. Com esta informação em mãos, a equipe de desenvolvimento é capaz de definir a plataforma do sistema que será usada para desenvolver o produto nas próximas fases.

Na *fase de planejamento*, o dono da plataforma e clientes identificam mais requisitos

[1]Este valor é definido com o propósito de facilitar o processo de gerenciamento das tarefas. Isto é, caso ocorra algum problema na execução da atividade então o líder de produto deve ser capaz de tomar às devidas providências em um curto intervalo de tempo.

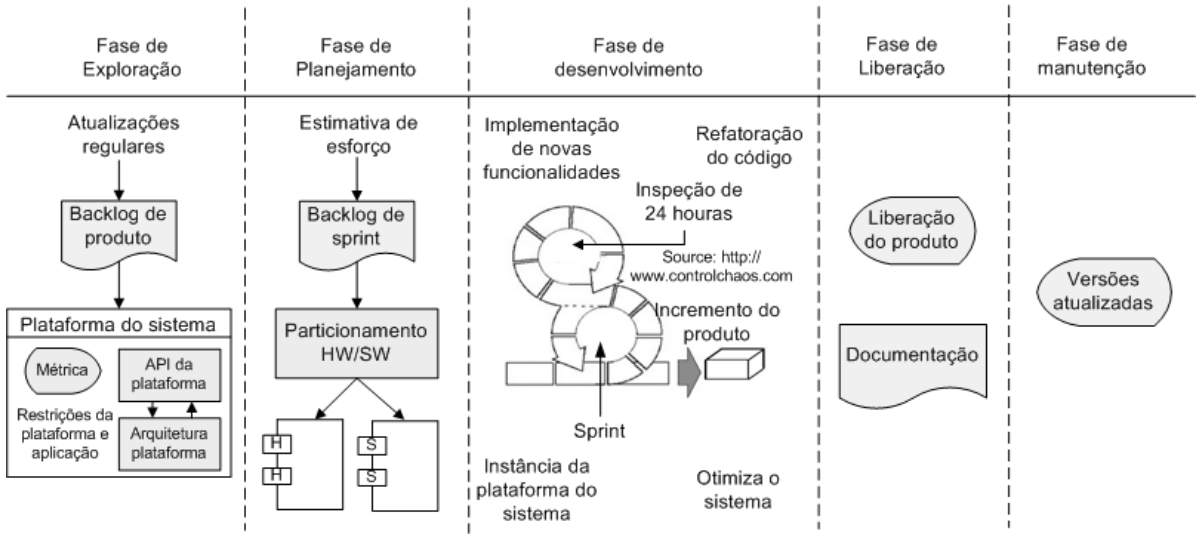

Figura 4.6: Ciclo de Vida dos Processos da Metodologia Proposta.

e priorizam o backlog de produto. Depois disso, a equipe de desenvolvimento passa aproximdamente um dia para estimar os itens de backlog de produto e os decompõem em tarefas. As tarefas que constituem o backlog de *sprint* devem levar de 1 a 16 horas para serem concluídas[2]. Protótipos e projeto exploratório podem também ser desenvolvidos nesta fase com o intuito de ajudar a esclarecer os requisitos do sistema.

Na *fase de desenvolvimento*, os membros da equipe implementam novas funcionalidades e melhoram o sistema baseado nos itens do backlog de *sprint*. As reuniões diárias são conduzidas no mesmo horário e local com o propósito de monitorar e adaptar as atividades para produzir os resultados desejados. No fim de cada iteração, testes funcionais e unitários são realizados em sistema de integração contínua de código. Otimização do sistema também ocorre durante esta fase. O último *sprint* fornece o produto para ser utilizado no ambiente operacional.

Na *fase de liberação*, o produto é instalado e colocado em uso prático. Esta fase geralmente envolve a identificação de erros e melhorias nos serviços do sistema. Deste modo, o dono da plataforma e clientes decidem se estas mudanças serão incluídas na versão atual ou subseqüente do produto. Esta fase tem como objetivo entregar a versão final do produto e a documentação necessária para o cliente. A *fase de manutenção* pode também requerer mais *sprints* com o intuito de implementar novas funcionalidades, melhorar e consertar defeitos apontados na fase de liberação.

[2]Este valor representa aproximadamente dois dias de trabalho de um membro da equipe.

4.5 Descrição dos Processos

A metodologia proposta TXM é composta por vários processos que tem como objetivo desenvolver projetos de sistemas embarcados de tempo real. A figura 4.7 mostra a interação entre processos.

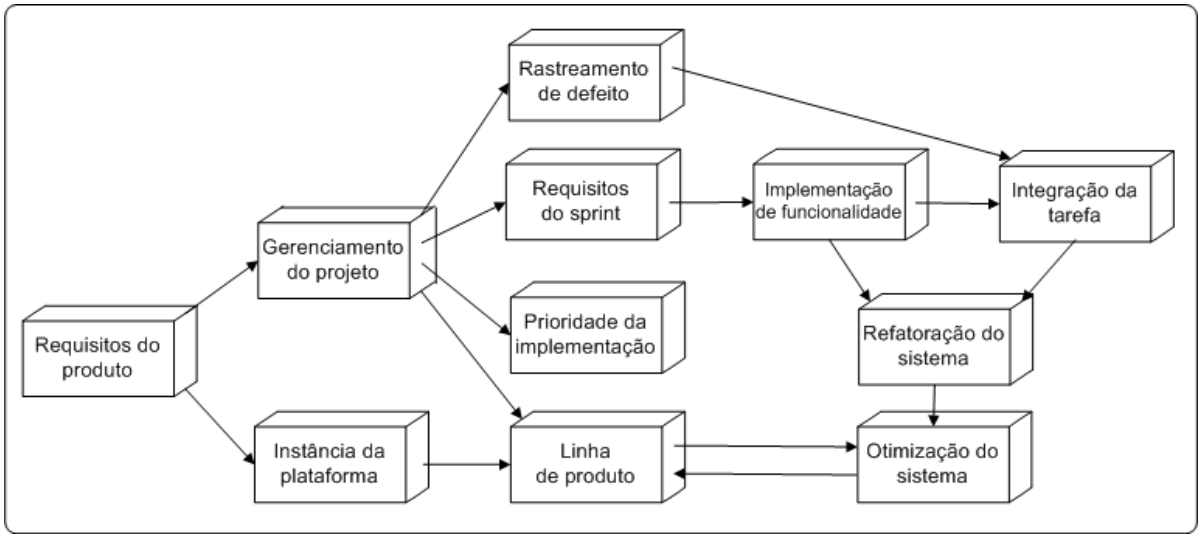

Figura 4.7: Interação entre Processos.

A metodologia proposta é composta de dez processos. Cada processo pode ser brevemente descrito da seguinte maneira:

Requisitos do produto: Este processo fornece atividades para criar, gerenciar e controlar todos os requisitos do sistema (funcional e não funcional).

Gerenciamento do Projeto: Este processo fornece atividades para gerenciar o backlog de *sprint* e produto, coordenar atividades, gerar versões intermediárias do produto, melhorar o processo de desenvolvimento e manter a equipe focada nos objetivos do projeto.

Instância da Plataforma: Este processo fornece atividades para estimar as métricas do sistema, determinar o particionamento hardware/software dos objetos funcionais e escolher a plataforma baseada nas restrições da aplicação e plataforma.

Rastreamento de Bug: Este processo fornece atividades para criar e gerenciar o ciclo de vida dos itens de projeto (defeito, tarefa e melhorias).

Requisitos do Sprint: Este processo fornece atividades para analisar, avaliar e estimar o esforço de implementação das funcionalidades do sistema antes de iniciar um

novo *sprint* do projeto.

Prioridade da Implementação: Este processo fornece atividades para gerenciar qualquer tipo de interrupção que pode impactar os objetivos do projeto.

Linha de Produto: O processo para gerenciar a linha de produto fornece atividades para estruturar o repositório, criar a linha de desenvolvimento do produto, permitir que a equipe de desenvolvimento integre novas tarefas no sistema e libere novas versões do produto no mercado.

Implementação de Funcionalidade: Este processo fornece atividades para cirar casos de teste[3] para cada código antes de implementá-lo. Isto ajuda a melhorar a qualidade do produto e reduzir a criação de funções ou métodos complexos.

Integração de Tarefas: Este processo fornece atividades para criar, implementar e integrar novas tarefas no sistema.

Refatoração do Sistema: Este processo fornece atividades para melhorar o código movendo funções entre objetos, eliminando duplicações e mantendo o número de funções e métodos o mais baixo possível.

Otimização do Sistema: Este processo fornece atividades para satisfazer as restrições do sistema e assegurar que a otimização de uma dada métrica não violará a restrição de uma outra.

É importante lembrar que as práticas ágeis dos métodos XP e Scrum assim como os padrões organizacionais foram integradas e adaptadas nos processos descritos acima. Sendo assim, a Seção 6 tem como objetivo descrever a aplicação das práticas e padrões ágeis no desenvolvimento dos estudos de caso.

As próximas subseções descrevem cada processo em termos de atividades, papéis responsáveis, práticas e padrões ágeis integrados, artefatos produzidos e ferramentas usadas no processo. Para eliminar ambigüidades na descrição dos processos da metodologia TXM, nós utilizamos a linguagem de modelagem de processos descrita por [56]. A Seção E do apêndice desta dissertação fornece uma visão geral dos principais elementos desta linguagem de modelagem.

[3]Caso de teste significa uma descrição das entradas, instruções de execução e resultados esperados que são criados com o intuito de exercitar os caminhos do código de um programa ou mostrar que um dado requisito esteja implementado.

4.5.1 Processo para Gerenciar os Requisitos do Produto

Objetivo: O processo para gerenciar o backlog de produto fornece atividades para criar, gerenciar e controlar todos os requisitos do sistema (funcional, não funcional e de ambiente).

Padrões e Práticas Ágeis: As práticas planejamento do sprint, revisão do sprint, metáfora do sistema foram integradas e adaptadas neste processo.

Fase: Todas as fases do projeto (exploração, planejamento, desenvolvimento, liberação e manutenção).

Entrada: Identificar as necessidades do mercado para uma linha de produto específica.

Descrição: O processo inicia através da identificação das necessidades do mercado para uma linha de produto específica. O proprietário da plataforma, cliente e pessoas técnicas participam em uma reunião (*brainstorming*) com o propósito de capturar em alto nível os requisitos do produto. Depois disso, eles criam um backlog de produto inicial que guiará o líder do produto e membros da equipe para capturar mais requisitos e criar um protótipo do produto. As primeiras iterações ajudarão a responder perguntas tais como se a tecnologia necessária para construir o produto existe, a dificuldade para construir o produto, e se a empresa tem experiência suficiente usando a tecnologia.

No final de cada iteração, o proprietário da plataforma e líder do produto verificam se o desenvolvimento do produto é viável ou não para a empresa. Sendo assim, se o projeto não for viável para a empresa então o mesmo pode ser cancelado exatamente depois do fim da iteração. Se o projeto for viável para a empresa então mais requisitos são identificados com o propósito de serem implementados na próxima iteração e o backlog de produto é atualizado. Se não existem mais requisitos para serem implementados então o produto é colocado em uso prático.

Papel Responsável: Proprietário da Plataforma

Papéis Envolvidos: Proprietário da Plataforma, Clientes, Líder do Produto e pessoas técnicas.

Ferramentas: Planilha do Excel

Saída: Este processo auxilia o proprietário da plataforma a criar e gerenciar a lista de todos os requisitos do sistema através da planilha de backlog de produto.

A Figura 4.8 mostra as atividades assim como os artefatos que são produzidos neste processo.

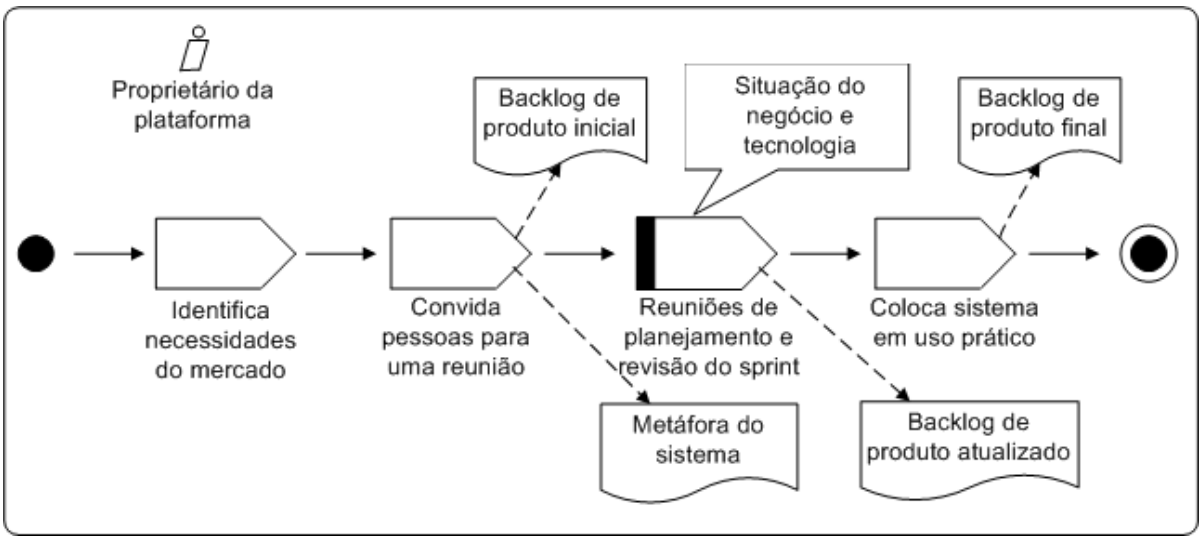

Figura 4.8: Processo para Gerenciar o Backlog de Produto.

4.5.2 Processo para Gerenciar o Projeto

Objetivo: O processo para gerenciar o desenvolvimento do projeto fornece atividades para gerenciar o backlog de produto e *sprint*, coordenar as atividades, gerenciar as versões intermediárias do produto, rastrear os defeitos do produto, melhorar o processo de desenvolvimento e manter a equipe focada nos objetivos do projeto.

Padrões e Práticas Ágeis: Este processo utiliza a maioria dos padrões e práticas ágeis que incluem: *planejamento do sprint, reuniões diárias, revisão do sprint, semana de 40 horas e geração de versões diárias.*

Fase: Todas as fases do projeto (exploração, planejamento, desenvolvimento, liberação e manutenção).

Entrada: Backlog de Produto

Descrição: O processo inicia através do refinamento e priorização do backlog de produto que contém os requisitos do sistema. No planejamento do sprint, o proprietário da plataforma e clientes escolhem os objetivos para a próxima iteração baseado nos itens de backlog de produto de alto valor de negócio e risco. Depois disso, o proprietário da

plataforma, o líder do produto e a equipe de desenvolvimento se reúnem para considerar como alcançar os objetivos do *sprint* e criar o backlog de *sprint*.

O *backlog* de *sprint* deve conter somente tarefas no intervalo de 4 a 16 horas[4]. Durante o desenvolvimento do produto, o backlog de *sprint* é atualizado em uma base diária a medida que as atividades são realizadas. Além disso, se novas tarefas são descobertas durante o desenvolvimento do sistema então os membros da equipe devem incluí-las no backlog de *sprint* e atualizá-las em uma base diária. O líder do produto conduz reuniões diariamente no mesmo lugar e horário com os membros da equipe com o propósito de monitorar e controlar a complexidade das tarefas. Estas reuniões diárias fornecem um status do projeto para o líder do produto e cria o hábito de compartilhar conhecimento.

Se o *sprint* não está concluído então os membros da equipe trabalham em um passo sustentável (isto significa que horas extras não são permitidas) nas tarefas do backlog de *sprint*. Durante esta fase, versões intermediárias do produto são produzidas em uma base diária que ajuda clientes a esclarecerem os requisitos e avaliar os riscos mais cedo no processo de desenvolvimento do sistema. Além disso, os membros da equipe podem integrar e testar as mudanças em uma maneira incremental dentro do projeto. Ou seja, eles não precisam esperar até o fim do *sprint* para integrar as mudanças. Sendo assim, os problemas de integração podem ser substancialmente reduzidos através das integrações incrementais e freqüentes.

No fim do *sprint*, o líder de produto e equipe de desenvolvimento devem mostrar os resultados do trabalho para o proprietário da plataforma e clientes. Esta reunião tem como objetivo apresentar o incremento do produto, situação do negócio e tecnologia. Estes artefatos ajudam o proprietário da plataforma e clientes a decidirem os objetivos para o próximo *sprint*. Depois da revisão do *sprint*, existe uma reunião de retrospectiva (*retrospective meeting*) que tem o propósito de coletar as melhoras práticas usadas durante o *sprint* e identificar o que poderia ser melhorado para o próximo sprint. Nesta reunião, o líder do produto escreve o que poderia ser melhorado em um formulário resumido e os membros da equipe priorizam e fornecem potenciais melhorias. A participação do proprietário da plataforma é opcional nesta reunião.

[4]Este intervalo corresponde a dois dias de trabalho e diminui o risco do projeto. Em outras palavras, este intervalo possibilita que o líder de produto tome as devidas ações em um curto intervalo de tempo caso a tarefa não seja realizada com sucesso.

Papel Responsável: Líder do Produto

Papéis Envolvidos: Proprietário da Plataforma, Líder do Produto, Clientes e Equipe de Desenvolvimento.

Ferramentas: Planilha Excel, ferramentas de integração e controle de versão.

Saída: Este processo auxilia o líder do produto a gerenciar as atividades de desenvolvimento do projeto. No fim de cada iteração do projeto, existe um incremento de novas funcionalidades e melhorias do produto.

A Figura 4.9 mostra as atividades assim como os artefatos que são produzidos neste processo.

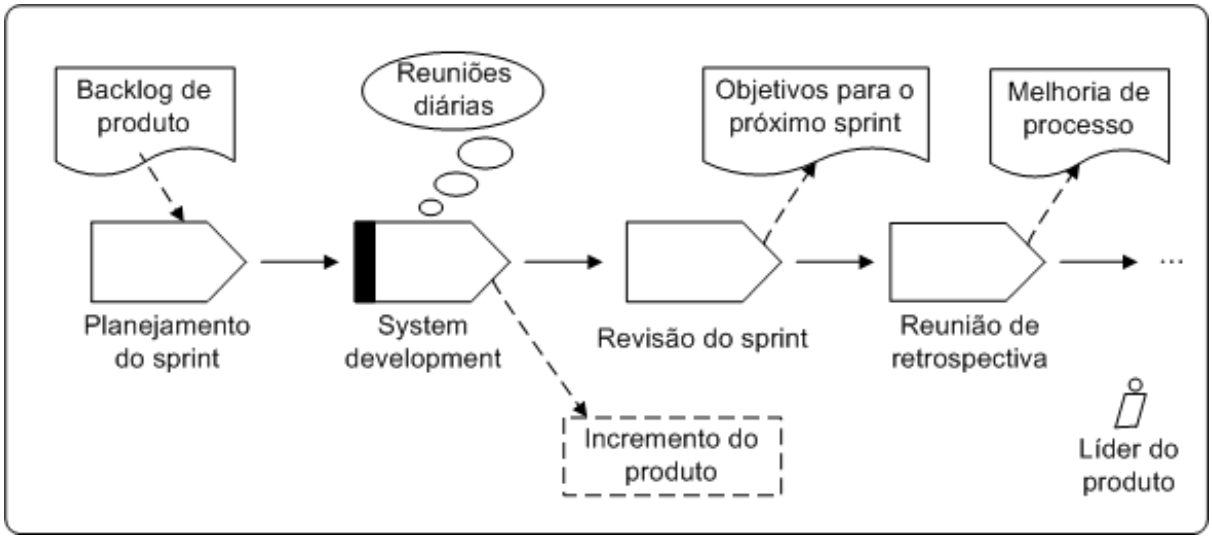

Figura 4.9: Processo para Gerenciar o Projeto.

4.5.3 Processo para Instanciar a Plataforma

Objetivo: O processo para instanciar a plataforma fornece atividades para estimar as métricas do sistema, determinar o particionamento hardware/software dos objetos funcionais e finalmente escolher a plataforma alvo (na qual o produto será desenvolvido) baseado nas restrições da aplicação e plataforma.

Padrões e Práticas Ágeis: Este é um novo processo proposto na metodologia de desenvolvimento TXM.

Fase: Fase de Exploração

Entrada: Backlog de Produto

Descrição: Este processo inicia através da estimativa das métricas do sistema tais como tempo de execução, consumo de energia, tamanho da memória de programa de dados. Estas métricas são estimadas baseadas nos requisitos funcionais e não funcionais do backlog de produto. No início do projeto, este backlog de produto não representa uma lista completa de requisitos do sistema, mas contém os requisitos mais importantes que podem ajudar a equipe de desenvolvimento a estimar as métricas do sistema.

As funcionalidades do sistema podem ser especificadas em uma linguagem de modelagem unificada (UML) através da criação de diagramas de classe, colaboração e seqüência. As ferramentas CASE (*Computer-Aided Software Engineering*) como *Together e Rational Rose* podem ser usadas para entrar com o modelo do sistema. Depois de especificar o modelo do sistema em UML usando estas ferramentas, o código poderia ser gerado automaticamente na linguagem selecionada pelo projetista do sistema (p.e., SystemC, Java, and C/C++). Nguyen et. al. [38] fornece uma ferramenta que possibilita o projetista do sistema especificar o modelo do sistema em UML e automaticamente convertê-lo em código SystemC. Depois de gerar o código na linguagem especificada, ferramentas de estimativas de hardware/software poderiam ser usadas para estimar as métricas do sistema. A ferramenta de estimativa desenvolvida por Oliveira é capaz de estimar o tempo de execução e consumo de energia baseado em código assembly do micro-controlador AT89S8252 [39].

Deste modo, depois de estimar as métricas do sistema, esta informação pode ser fornecida para a ferramenta de particionamento hardware/software desenvolvida em parceria com uma aluna de graduação [29]. Esta ferramenta está atualmente sendo integrada como um plug-in do Eclipse. A ferramenta permite que o usuário entre *(i)* com o modelo do sistema, *(ii)* os parâmetros da função objetivo tais como importância da métrica e restrições, *(iii)* selecionar os componentes da plataforma do sistema, e *(iv)* encontrar a melhor partição do sistema.

Por conseguinte, esta ferramenta procura pelo melhor particionamento que atenda as restrições do projeto. Desde que a maioria das decisões de projeto são conduzidas por restrições então as restrições da aplicação deveriam ser incorporadas na função objetivo tal que as partições que atendem as restrições são consideradas melhores do que aquelas que não atendem. Depois de instanciar a plataforma baseada nas restrições da aplicação e plataforma então os membros da equipe podem iniciar o desenvolvimento do produto.

Papel Responsável: Líder do Produto

Papéis Envolvidos: Líder do Produto e Equipe de Desenvolvimento

Ferramentas: Planilha do Excel e ferramentas de particionamento de sistema.

Saída: Este processo ajuda a equipe de desenvolvimento a definir a plataforma do sistema (que será usada para desenvolver o produto) baseado nas restrições da aplicação e da plataforma. Um conjunto de ferramentas de projeto ou *benchmarks* ajudam a equipe de desenvolvimento a realizar esta tarefa.

A Figura 4.10 mostra as atividades deste processo.

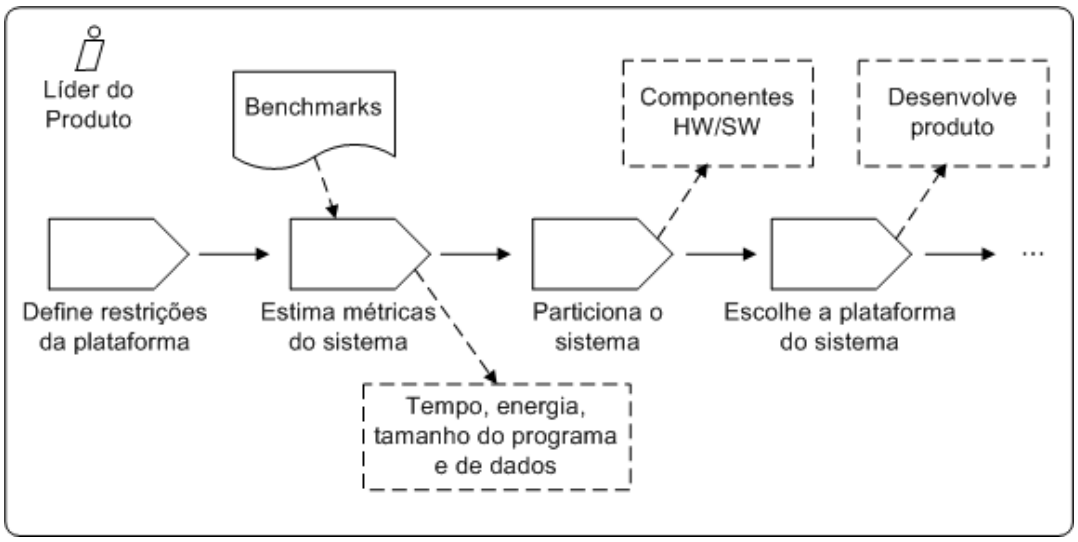

Figura 4.10: Processo para Instanciar a Plataforma.

4.5.4 Processo para Rastreamento de Bugs no Produto

Objetivo: Este processo para rastrear os defeitos do produto fornece atividades para gerenciar o ciclo de vida dos itens do projeto (defeito, tarefa e melhoria).

Padrões e Práticas Ágeis: Este é um novo processo proposto na metodologia de desenvolvimento TXM.

Fase: Fase de planejamento, desenvolvimento, liberação e manutenção.

Entrada: A equipe encontra um novo item no produto.

Descrição: Este processo inicia através da identificação de um novo item no produto que pode incluir um defeito, tarefa ou melhoria. Defeitos no sistema podem ser encontrados usando o sistema de *log* mostrado na figura. Este sistema de *log* é uma extensão

da idéia proposta por [45] que tem como objetivo eliminar o excesso de memória causado pela chamada *printf*. Sendo assim, este sistema de *log* usa um buffer circular na memória para armazenar mensagens de texto de comprimento fixo. Para que a equipe de desenvolvimento realize o diagnóstico do sistema, eles devem escrever mensagens de log do início e fim de uma rotina de serviço de interrupção (ISR - *Interrupt Service Routine*) e então subtrair o valor do temporizador para checar o tempo de execução do código.

```
Timer Componente Função:NomedoArquivo(linha)
12320   c_LCD  ->  LCD_Driver_InitModule: lcd_class_driver.c(85)
11789   c_LCD  ->  LCD_WriteData: lcd_class_driver.c(90)
13452   c_LCD  ->  LCD_InterfaceDesc: lcd_class_interface.c(102)
11216   c_LCD  ->  LCD_Interface_Create: lcd_class_interface.c(18)
12834   c_LCD  ->  LCD_initialize: lcd_class_interface.c(80)
```

Figura 4.11: Exemplo de Saída do Log.

Depois de identificar o defeito através ou do log do sistema ou do *processo de implementação de novas funcionalidades*, o item pode ser aberto em uma ferramenta de rastreamento de defeito por qualquer membro da equipe. Porém, a pessoa que abre o item deve fornecer as informações necessárias para reproduzir o defeito tal como: *resumo, plataforma, produto, versão do software, grupo funcional, componente e descrição do item.* Depois disso, o item é automaticamente atribuído para a pessoa responsável pelo grupo funcional na qual o defeito foi aberto. Se a pessoa responsável por consertar o item não for capaz de reproduzir o defeito então ele pode conversar diretamente com a pessoa que abriu o defeito com o propósito de obter mais informações de como reproduzi-lo.

Depois de consertar o defeito, a pessoa responsável deve mudar o estado para "fixed" com o intuito de ser verificado por um outro membro da equipe em uma nova versão do produto. Se o item for realmente consertado então os membros da equipe mudam o estado para ´´closed´´ e o líder do produto inclui esta informação nas notas de liberação do produto. Se o item não for consertado então os membros da equipe podem reabrir e atribuí-lo novamente para a pessoa responsável.

Papel Responsável: Líder do Produto

Papéis Envolvidos: Líder do Produto e Equipe de Desenvolvimento

Ferramentas: Planilha do Excel e ferramenta de rastreamento de defeito

Saída: Este processo fornece as informações necessárias sobre a qualidade do produto nas notas de liberação para o usuário final.

A Figura 4.12 mostra as atividades assim como os artefatos que são produzidos neste processo.

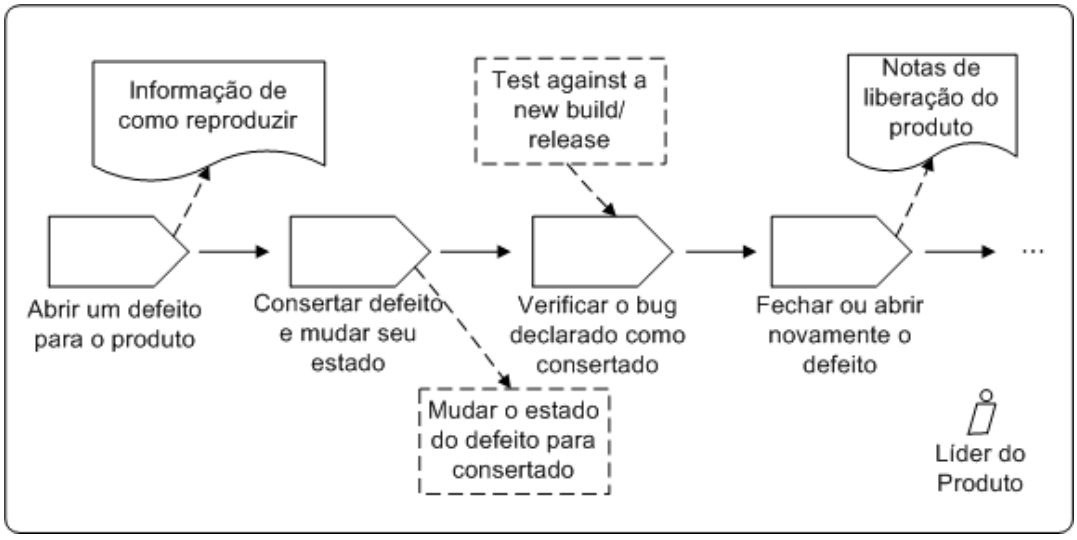

Figura 4.12: Processo para Rastrear os Defeitos do Produto.

4.5.5 Processo para Escolher os Requisitos do Sprint

Objetivo: Este processo para escolher os requisitos do *sprint* fornece atividades para analisar, avaliar e estimar o esforço de implementação das funcionalidades do sistema antes de iniciar um novo *sprint* do projeto.

Padrões e Práticas Ágeis: A prática de planejamento do *sprint* foi integrada e adaptada neste processo.

Fase: Fase de Planejamento

Entrada: Backlog de Produto

Descrição: Este processo para escolher os requisitos do *sprint* fornece atividades para analisar, avaliar e estimar o esforço de implementação das funcionalidades do sistema antes de iniciar um novo *sprint* do projeto. Este processo inicia com uma reunião que tem o propósito de determinar as funcionalidades do sistema que serão implementadas para o próximo *sprint*. O proprietário da plataforma, líder do produto e clientes podem participar desta reunião e escolher as funcionalidades do sistema para o *sprint* baseado nos seguintes

critérios: *(i)* dividir as funcionalidades do sistema dependendo de sua estimativa e da carga de trabalho de cada iteração e *(ii)* dividir por fronteiras de dados, por exemplo, selecionando um subconjunto de operações (p.e., preencher e armazenar um *array* de dados) suportados por uma dada funcionalidade [14].

Codificação de objetos *mock* e *stubs* deveriam também ser usados com o propósito de compensar a ausência de funcionalidade do componente. Pelo fato de que componentes do sistema (p.e., *driver* do display ou teclado) tem pouco acoplamento com o resto do sistema, seria mais fácil construir objetos *mock* para cobrir as entradas e saídas dos componentes. Além disso, quando visto a partir do todo, tendo tais tipos de objetos *mock* disponíveis em tempo para outros subsistemas pode ser essencial para permitir que o resto do sistema cresça de forma ótima. Deste modo, com o propósito de habilitar uma integração incremental em projetos de sistemas embarcados de tempo real, a noção de que o subsistema deveria esperar um nível de retrabalho entre iterações deve ser introduzido.

Finalmente, depois de definir as funcionalidades do sistema que se encaixam em uma iteração, cada membro da equipe deveria escolher as funcionalidades do sistema que ele será responsável por implementar durante o *sprint*. Depois disso, a equipe de desenvolvimento deveria usar um conjunto de ferramentas de projeto para particionar a especificação funcional do sistema em um conjunto de componentes do sistema.

Papel Responsável: Líder do Produto

Papéis Envolvidos: Proprietário da plataforma, líder do produto, cliente e equipe de desenvolvimento.

Ferramentas: Planilha Excel e ferramentas de particionamento de sistema

Saída: As funcionalidades do sistema devem ser incluídas no backlog de *sprint* e elas devem ser particionadas ou em hardware ou em software antes de iniciar o *sprint*.

A Figura 4.13 mostra as atividades assim como os artefatos que são produzidos neste processo.

4.5.6 Processo para Mudar a Prioridade da Implementação

Objetivos: O processo para tratar da interrupção de tarefas fornece atividades para gerenciar qualquer tipo de interrupção que podem impactar os objetivos do projeto.

Padrões e Práticas Ágeis: A prática *firewall* foi integrada e adaptada neste pro-

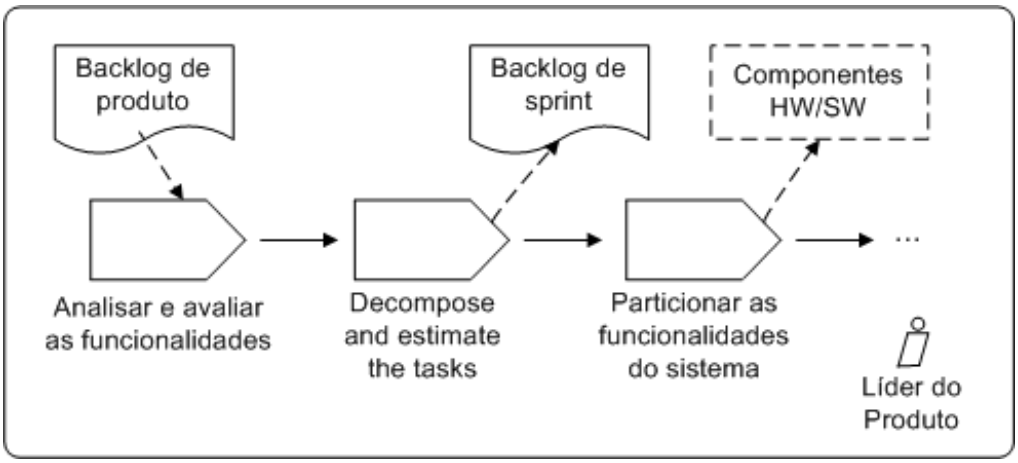

Figura 4.13: Processo para Escolher os Requisitos do Sprint.

cesso.

Fase: Todas as fases do projeto (exploração, planejamento, desenvolvimento, liberação e manutenção).

Entrada: Uma nova tarefa do sistema que deve ser implementada.

Descrição: O processo inicia com uma dada tarefa (p.e., introduzir um novo membro da equipe no projeto, atualizar algum documento do projeto, treinar um novo membro da equipe) a ser realizada enquanto o projeto está em andamento. Deste modo, o líder do produto deve identificar a prioridade das tarefas entre as outras tarefas que estão sendo executadas. Além disso, o líder do produto deve estimar o esforço da tarefa.

Se o líder do produto identifica que o esforço da tarefa é baixo e que isto tem alta prioridade em relação a outra tarefa que está atualmente sendo executada então ele pode atribuir a tarefa para somente uma pessoa. Caso contrário, o líder do produto deve atribuir a tarefa para um pequeno grupo, mas ele deve manter os outros membros da equipe focados nos objetivos do projeto.

É importante salientar que depois de atribuir a tarefa, os membros da equipe envolvidos para realizar a tarefa devem completá-la antes de serem envolvidos em outras tarefas. Se alguma tarefa aparece no meio tempo então os membros da equipe envolvidos na tarefa atual não devem ser interrompidos para resolver esta nova tarefa.

Papel Responsável: Líder do Produto

Papeís Envolvidos: Líder do Produto e Equipe de Desenvolvimento

Ferramentas: Planilha do Excel

Saída: Este processo garante que as tarefas do projeto sejam 100 porcento concluídas

A Figura 4.14 mostra as atividades deste processo.

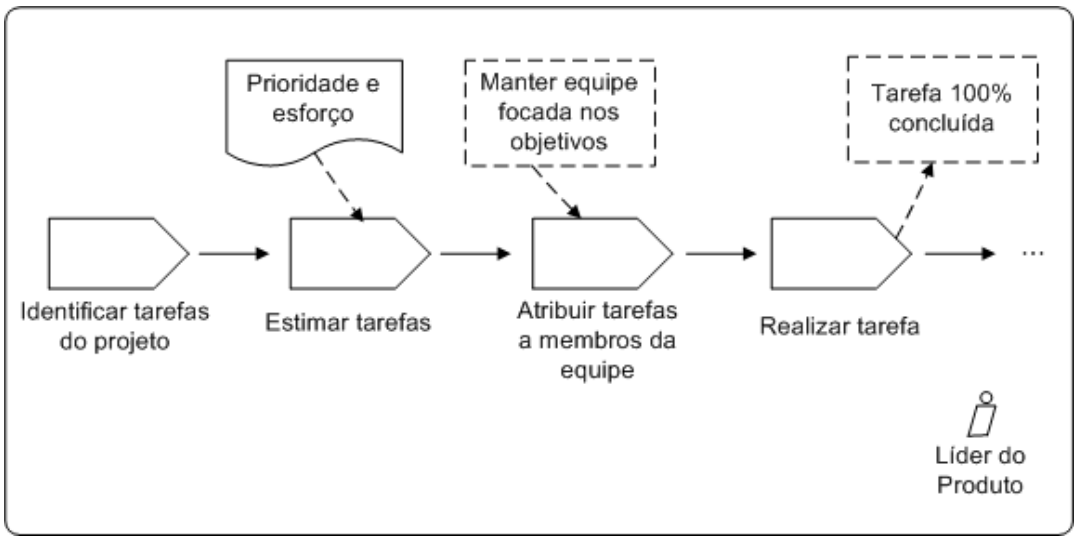

Figura 4.14: Processo para Mudar a Prioridade da Implementação.

4.5.7 Processo para Gerenciar a Linha de Produto

Objetivo: O processo para gerenciar a linha de produto fornece atividades para estruturar o repositório, criar a linha de desenvolvimento do produto, permitir que a equipe de desenvolvimento integre novas tarefas do sistema e liberar novas versões do produto no mercado.

Padrões e Práticas Ágeis: Os padrões *mainline, active development line, task branch, task level commit e release line* foram integrados neste processo.

Fase: Fases de exploração, desenvolvimento, liberação e manutenção.

Entrada: Componentes da Plataforma do Sistema

Descrição: A equipe de desenvolvimento estrutura o repositório do sistema incluindo os componentes da plataforma do sistema que consiste essencialmente da camada alto nível (API da plataforma) e baixo nível (a coleção de *drivers* que controlam os componentes físicos da plataforma). O repositório consiste somente dos componentes do sistema que são necessários para derivar novas linhas de produto. Depois disso, a equipe de desenvolvimento cria no repositório do sistema a linha de desenvolvimento na qual o produto

será desenvolvido. Esta linha de codificação permite que os membros da equipe desenvolvam e otimizem os componentes do produto[5].

Para implementar novas tarefas do sistema que podem incluir novos requisitos, melhorias e consertos de defeitos, cada membro da equipe deve criar um desvio (*branch*) na linha de codificação do produto. Este desvio ajuda os membros da equipe a implementar a tarefa sem ter que interromper outros membros da equipe. Depois de implementar e otimizar todos os componentes do sistema que constituem o produto, a equipe de desenvolvimento cria um desvio para liberação[6] do produto com o intuito de não interferir no desenvolvimento atual do sistema.

Papel Responsável: Líder do Produto

Papéis Envolvidos: Líder do Produto e Equipe de Desenvolvimento

Ferramentas: Planilhas Excel, ferramentas de integração e controle de versão.

Saída: Liberação de novas versões do produto no mercado.

A Figura 4.15 mostra as atividades assim como os artefatos que são produzidos neste processo.

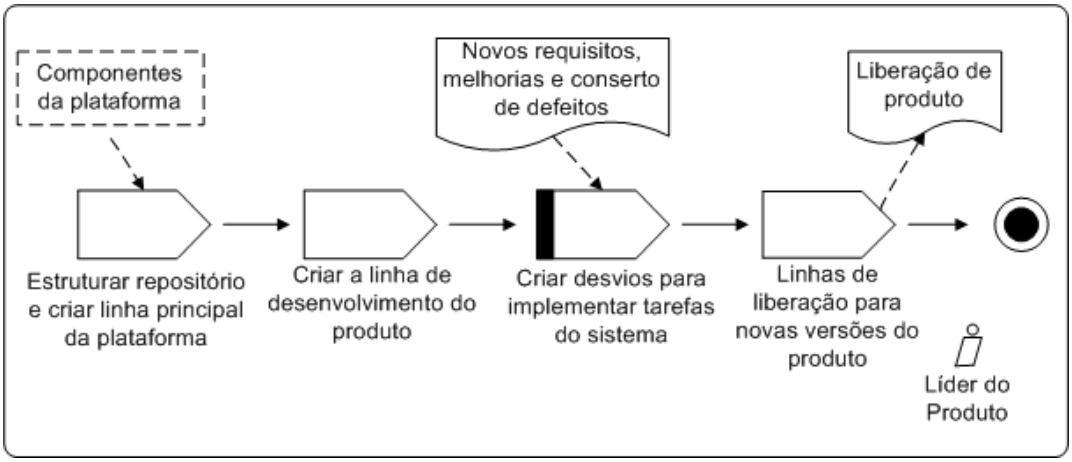

Figura 4.15: Processo para Gerenciar a Linha de Produto.

4.5.8 Processo para Implementar Novas Funcionalidades

Objetivo: O processo para implementar novas funcionalidades do sistema fornece atividades para criar os casos de teste do código antes de realmente implementá-lo. Isto ajuda

[5]A subseção B.4 do apêndice desta dissertação apresenta a infraestrutura de *build* necessária para auxiliar a execução das atividades deste processo.

[6]Conhecido também na língua inglesa como *release branch*.

a melhorar a qualidade do produto e reduzir a criação de funções e métodos complexos.

Padrões e Práticas Ágeis: As práticas padrão de codificação e teste antes do desenvolvimento foram integradas e adaptadas neste processo.

Fase: Fases de planejamento, desenvolvimento, liberação e manutenção.

Entrada: Nova funcionalidade do sistema a ser implementada.

Descrição: O processo inicia tendo uma nova funcionalidade do sistema a ser implementada no sprint atual. Antes de codificar a funcionalidade os membros da equipe devem primeiro criar os casos de teste para a funcionalidade que será implementada. Dependendo da funcionalidade a ser implementada, devem-se escrever testes unitários para cada estágio da computação. Depois de criar o teste unitário, os membros da equipe devem compilar o teste unitário antes de codificar a funcionalidade. Caso ocorram problemas na compilação então o código do teste unitário deve ser corrigido.

No momento da criação dos testes unitário, é importante identificar os diferentes tipos de domínio do sistema (p.e. LCD, comunicação serial, teclado) e isolá-los. Para cada domínio do sistema, deve-se desenvolver uma aplicação e executá-la separadamente de outros módulos com o intuito de identificar a causa raiz do problema. Um outro ponto de extrema importância é distinguir entre componentes que controlam o hardware e componentes que são guiados pelo ambiente.

Para aqueles componentes que controlam o hardware, os casos de teste devem ser executados manualmente na plataforma alvo. Neste caso, deve-se definir quais comandos serão enviados para o driver do hardware e os resultados esperados destes comandos. Por exemplo, uma aplicação que executa na plataforma alvo poderia ser desenvolvida para testar todas as posições de escrita em um LCD 16x2. Um conjunto de testes unitários de hardware junto com uma aplicação pode ser usado para assegurar a corretudo do módulo.

Para aqueles componentes que são guiados pelo ambiente, pode-se utilizar os casos de teste para substituir os eventos que surgem do ambiente. Por exemplo, o módulo “sensor” implementa as funcionalidades de captura dos dados que mensura uma grandeza física no ambiente. Sendo assim, o módulo recebe dados do sensor através de uma interrupção da porta serial e a interface serial define a interação com o hardware. Deste modo, pode-se criar um módulo “sensorTest” que fornece os dados que vem do sensor para o módulo “sensor”.

Depois de criar e compilar os casos de teste com sucesso, o membro da equipe inicia a codificação da funcionalidade seguindo os padrões de codificação definidos no início do projeto. A funcionalidade é completamente implementada somente depois que o membro da equipe executa o teste unitário que foi criado para a funcionalidade. Isto assegura que a funcionalidade esteja em conformidade com sua especificação.

Papel Responsável: Equipe de desenvolvimento

Papéis Envolvidos: Equipe de desenvolvimento

Ferramentas: Planilha Excel e *framework* de teste unitário

Saída: O teste unitário para cada função é criado e assegura que as funções são executadas corretamente.

A Figura 4.16 mostra as atividades deste processo.

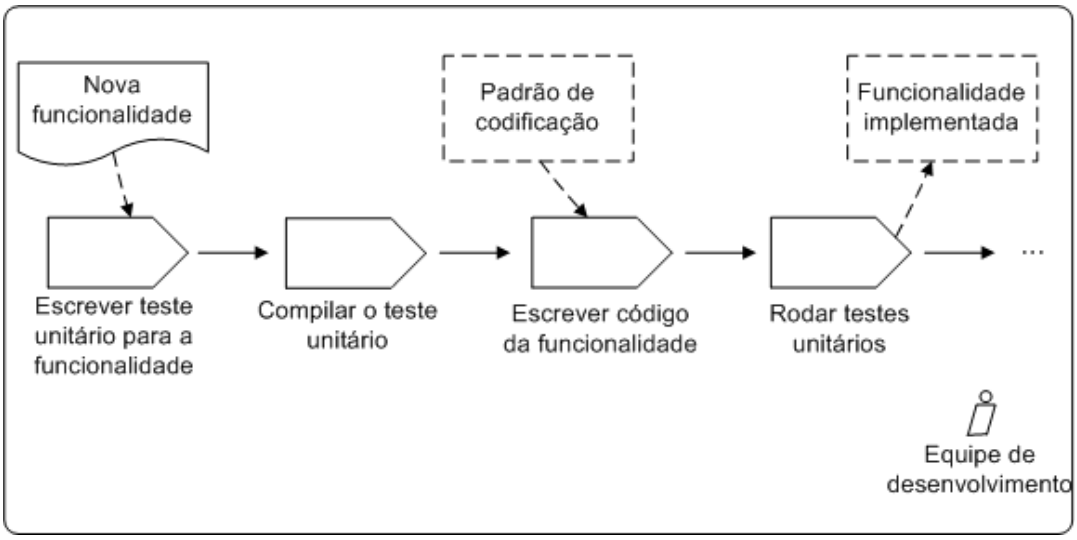

Figura 4.16: Processo para Implementar Novas Funcionalidades.

4.5.9 Processo para Integrar Tarefas do Sistema

Objetivo: O processo para integrar as tarefas do sistema fornece atividades para criar, implementar e integrar novas tarefas no sistema.

Padrões e Práticas Ágeis: As práticas versões diárias, testes de regressão e desvio de tarefa foram integradas e adaptadas neste processo.

Fase: Fases de desenvolvimento, liberação e manutenção.

Entrada: Uma nova tarefa deve ser implementada e integrada no sistema.

Descrição: O membro da equipe que deseja implementar e integrar uma nova tarefa (p.e., requisito, melhoria ou conserto de defeito) no sistema deve primeiramente criar um *branch* no repositório do sistema. Depois disso, ele pode implementar a tarefa no *branch*. Porém, antes de disponibilizar a tarefa no *branch*, o membro da equipe deve compilar e executar o teste unitário com o propósito de checar problemas de semântica e compilação. Se não existirem problemas de compilação e semântica então o membro da equipe pode integrar a mudança na linha principal. De outro modo, o membro da equipe deve resolver o problema com o intuito de continuar com o desenvolvimento do produto. Se o problema for relacionado a integração com a tarefa de um outro membro da equipe então ambos o membros podem juntos resolver o problema de integração.

Papel Responsável: Equipe de desenvolvimento

Papéis Envolvidos: Equipe de desenvolvimento

Ferramentas: Ferramentas de integração e gerenciamento de controle de versão.

Saída: Este processo garante que uma nova tarefa seja implementada e integrada no sistema sem quebrar a linha principal de desenvolvimento.

A Figura 4.17 mostra as atividades deste processo.

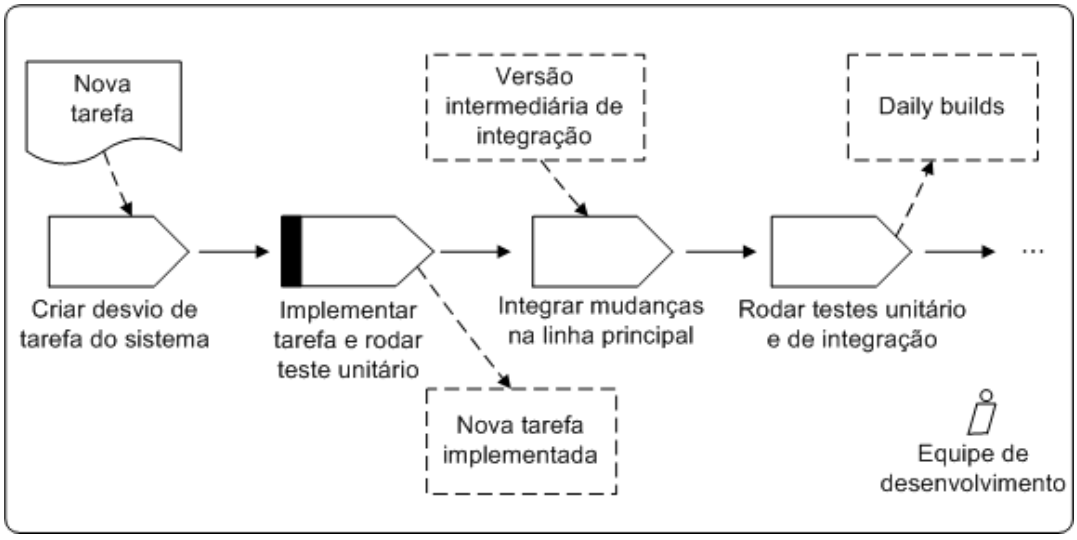

Figura 4.17: Processo para Integrar Tarefas no Sistema.

4.5.10 Processo para Refatoração do Código

Objetivo: O processo para refatorar o código fornece atividades para melhorar o código movendo funções entre objetos, eliminando duplicação e mantendo o número de funções e métodos o mais baixo possível.

Padrões e Práticas Ágeis: As práticas teste unitário, refatoração e integração contínua foram integradas e adaptadas neste processo.

Fase: Fase de desenvolvimento

Entrada: Identificar oportunidade para melhorar o código.

Descrição: Depois de implementar a funcionalidade do sistema, o membro da equipe identifica oportunidade para melhorar um código existente. Deste modo, ele cria um novo *branch* de tarefa no repositório do sistema com o intuito de implementar a melhoria. Antes de implementar a melhoria, o membro da equipe verifica se existe alguma necessidade para atualizar o teste unitário da funcionalidade. Depois disso, ele melhora o código sem alterar o comportamento externo. O processo de refatoração pode solicitar que a função seja movida de um objeto funcional para outro, ou seja, mover o software executando em um dos processadores para um bloco de hardware ou vice-versa. O mesmo pode acontecer de mover um conjunto de funções implementadas em C para assembly com o propósito de melhorar eficiência do sistema.

Depois de refatorar o código, o membro da equipe deve executar o teste unitário para verificar se as mudanças que ele implementou estão funcionando corretamente. Se não existe problema de compilação e o teste unitário não falhou então o membro da equipe pode integrar suas mudanças na linha de desenvolvimento principal do projeto. Depois de integrar o código, os testes de regressão devem ser executados para checar se existem problemas de compilação e semântica. Se não existem problemas então o processo de refatoração está concluído. Caso contrário, o membro da equipe deve investigar a razão do problema e pode solicitar a ajuda de um outro membro da equipe para resolver o problema.

Papel Responsável: Equipe de desenvolvimento

Papéis Envolvidos: Equipe de desenvolvimento

Ferramentas: Ferramentas de particionamento, integração e gerenciamento de controle de versão.

Saída: O código é melhorado sem alterar seu comportamento externo.

A Figura 4.18 mostra as atividades deste processo.

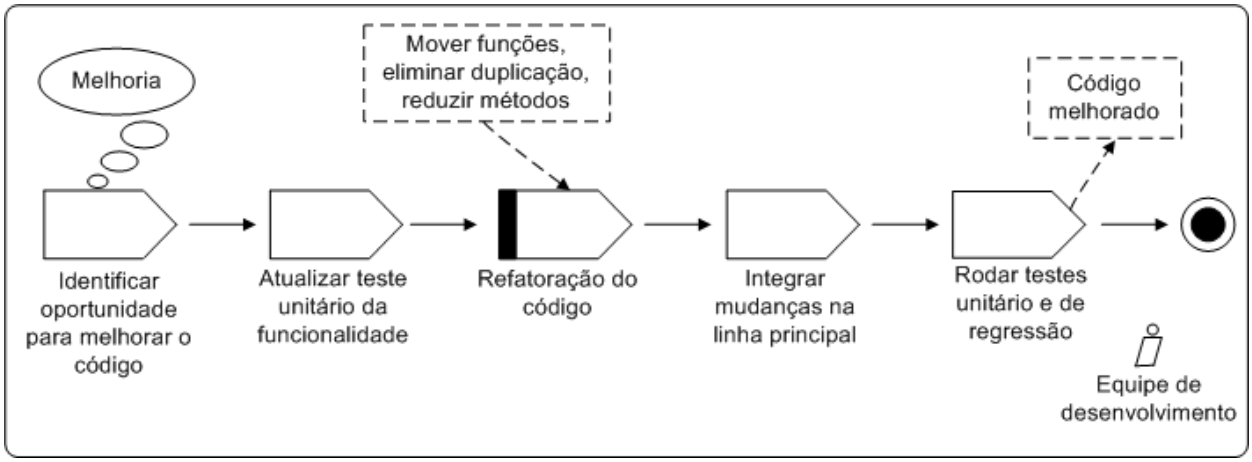

Figura 4.18: Processo para Refatoração do Código.

4.5.11 Processo para Otimização do Sistema

Objetivo: O processo para otimizar o sistema fornece atividades para atender as restrições do sistema e assegurar que a otimização de uma dada métrica não violará a restrição de uma outra.

Padrões e Práticas Ágeis: Práticas de teste unitário, integração contínua e refatoração foram integradas e adaptadas neste processo.

Fase: Fase de desenvolvimento

Entrada: Otimizar alguma variável do sistema (p.e., tempo de execução, consumo de energia, tamanho da memória de dados e programa).

Descrição: O processo inicia através da identificação de alguma variável do sistema que deve ser otimizada com o propósito de atender a restrição da plataforma ou aplicação. Deste modo, o membro da equipe deve estabelecer as métricas e assegurar que o processo de refatoração já tenha sido aplicado. Depois disso, ele pode executar uma ferramenta de *profiler* que monitora o programa e informa onde está consumido tempo, energia e espaço de memória. Nesta maneira, o membro da equipe pode encontrar pequenas partes do código onde estas variáveis podem ser otimizadas.

Depois disso, o membro da equipe deve otimizar as variáveis sob atenção na mão[7]. Como no processo de refatoração, o membro da equipe realiza a mudança em pequenos passos. Depois de cada passo, ele compila, testa e executa a ferramenta de *profiler* novamente. Se a variável não foi otimizada então o membro da equipe deve retornar a mudança no sistema de controle de versão. O processo de otimização continua até que as variáveis atendam as restrições.

Papel Responsável: Equipe de desenvolvimento

Papéis Envolvidos: Equipe de desenvolvimento

Ferramentas: Ferramentas de monitoramento, integração e controle de versão.

Saída: As métricas tempo de execução, consumo de energia, tamanho da memória de dados e programa atendem as restrições da aplicação e plataforma.

A Figura 4.19 mostra as atividades deste processo.

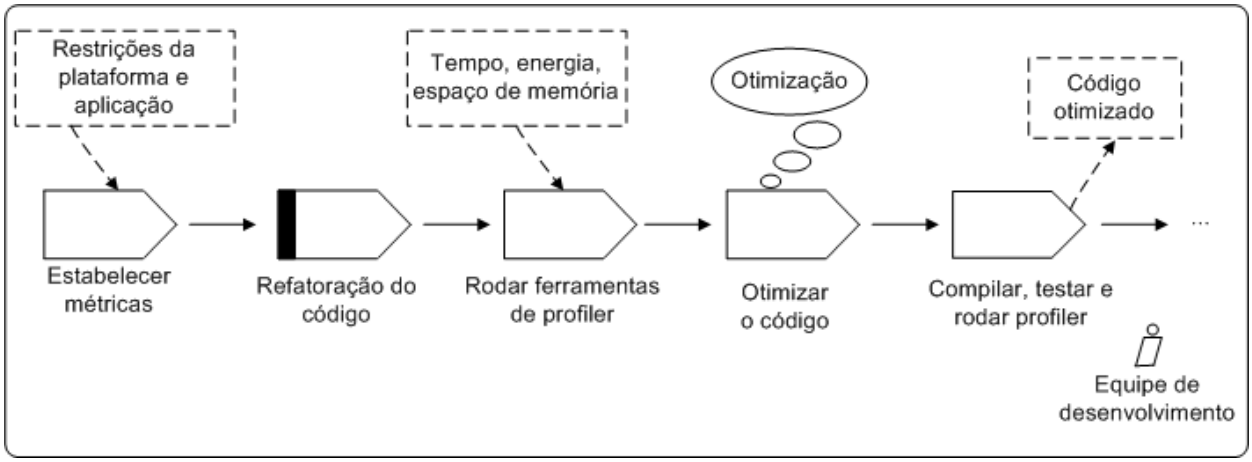

Figura 4.19: Processo para Otimização do Sistema.

4.6 Resumo

Este capítulo apresentou uma visão geral dos processos, papéis e responsabilidade de cada um envolvido nos processos, ciclo de vida e uma descrição detalhada dos processos que constituem a metodologia ágil proposta. Os processos podem ser divididos em três diferentes camadas e incluem: processos de plataforma de sistema, processos de desenvolvimento de produto e processos de gerenciamento de produto.

[7]A Seção D do apêndice desta dissertação descreve algumas técnicas de otimização de código que podem ser aplicados em projetos de sistemas embarcados de tempo real.

O grupo plataforma do sistema fornece os processos que são necessários para instanciar a plataforma para um dado produto. Os processos de desenvolvimento de produto ajudam a equipe de desenvolvimento a desenvolver e otimizar os componentes da aplicação e plataforma assim como integrar estes componentes na plataforma do sistema. O processo de gerenciamento de produto fornece atividades para monitorar e controlar o escopo do produto, tempo, qualidade e custos. Este grupo de processos também assegura que os parâmetros do projeto (escopo, tempo, qualidade e custo) são atendidos durante o desenvolvimento do produto.

A metodologia proposta definiu quatro papéis que podem ser envolvidos nos processos da seguinte maneira: proprietário da plataforma, líder do produto, líder de funcionalidade e equipe de desenvolvimento. O proprietário da plataforma é a pessoa que é oficialmente responsável pelos produtos que derivam de uma dada plataforma. O líder do produto é responsável pela implementação, integração e testes do produto assegurando que os parâmetros qualidade, tempo e custo definidos pelo proprietário da plataforma sejam atendidos. O líder de funcionalidades pode ser envolvido somente em projetos de médio e grande porte e ele é responsável por gerenciar, controlar e coordenar os projetos de subsistema, projetos de integração e parceiros externos que contribuem para um conjunto definido de funcionalidades. A equipe de desenvolvimento que pode consistir de programadores, arquitetos e testadores são responsáveis por trabalhar no desenvolvimento do produto.

Este capítulo também descreveu onze diferentes processos que incluem: requisitos do produto, gerenciamento do projeto, instância da plataforma, rastreamento de defeito, requisitos do *sprint*, prioridade de implementação, linha de produto, implementação de funcionalidade, integração de tarefa, refatoração do sistema e otimização do sistema. O processo de requisitos do produto fornece as atividades para criar, gerenciar e controlar todos os requisitos do sistema (funcional e não funcional). O processo de gerenciamento de projeto fornece as atividades para gerenciar o backlog de produto e sprint, coordenar atividades, produzir versões intermediárias do sistema e rastrear defeitos do produto.

O processo de instância da plataforma fornece atividades para estimar as métricas do sistema, determinar o particionamento hardware e software dos objetos funcionais e escolher a plataforma baseada nos requisitos da aplicação. O processo de rastreamento

de defeito fornece atividades para criar e gerenciar o ciclo de vida dos itens de projeto (defeito, tarefa e melhoria). O processo de requisitos do *sprint* fornece atividades para analisar, avaliar e estimar o esforço de implementação das funcionalidades do sistema antes de iniciar um novo *sprint* do projeto. O processo de prioridade de implementação fornece atividades para gerenciar qualquer tipo de interrupção que pode impactar os objetivos do projeto.

O processo de linha de produto fornece atividades para estruturar o repositório, criar a linha de desenvolvimento do produto, permitir que a equipe de desenvolvimento integre novas tarefas no sistema e libere novas versões do produto no mercado. O processo de implementação de funcionalidades fornece atividades para criar casos de teste para o código antes de implementá-lo. O processo de integração de tarefa fornece atividades para criar, implementar e integrar novas tarefas no sistema. O processo de refatoração do sistema fornece atividades para melhorar o código movendo funções entre componentes, eliminando duplicação e mantendo o número de funções e métodos o mais baixo possível. O processo de otimização do sistema fornece atividades para atender as restrições do sistema e assegurar que a otimização de uma métrica não viole a restrição de uma outra métrica.

Foi também enfatizado que as práticas ágeis XP e Scrum assim como os padrões organizacionais foram integrados e adaptados nestes processos. A seção "experimentos" 7 descreve como estes processos propostos foram aplicados no desenvolvimento dos protótipos. Finalmente, foi mostrado que a metodologia proposta consiste de cinco fases que incluem: Exploração, Planejamento, Desenvolvimento, Liberação e Manutenção. Na metodologia proposta, a duração de cada *sprint* pode variar de 1 a 4 semanas[8]. Para implementar o produto, pode existir aproximadamente de três a oito *sprints* para serem executados na fase de desenvolvimento antes de realmente liberar o produto para o mercado.

[8]Este intervalo ajuda a reduzir riscos e incertezas do projeto.

Capítulo 5

Ferramentas e Plataforma

Este capítulo tem como propósito apresentar as ferramentas que foram usadas nesta dissertação de mestrado. Este capítulo está dividido essencialmente em três seções. A primeira seção descreve a implementação dos algoritmos de particionamento de hardware/software e a ferramenta de captura de log no PC que foram desenvolvidos nos trabalhos [29, 40]. A segunda seção descreve uma ferramenta de *framework* de teste unitário que tem como intuito testar funções escritas em C que executam sob severas restrições no ambiente de execução. A terceira seção apresenta a plataforma de desenvolvimento que foi usada nos estudos de caso desta dissertação.

5.1 Ferramentas Desenvolvidas nesta Dissertação

5.1.1 Particionamento Hardware/Software

Estes algoritmos de particionamento de hardware/software foram implementados com o propósito de auxiliar o projetista do sistema embarcado na aplicação do processo 4.5.3 da metodologia proposta TXM. Sendo assim, a primeira subseção apresenta o algoritmo baseado em programação linear inteira e a subseção seguinte o algoritmo baseado em migração de grupos propostos por [4, 22].

Programação Linear Inteira

Programação linear inteira (ILP - *Integer Linear Programming*) é uma das técnicas que é usada para resolver o problema do particionamento. A técnica ILP define um conjunto de variáveis, um conjunto de inequações lineares que restringem o valor das variáveis e uma função linear simples que serve como uma função objetivo [22]. O objetivo principal é escolher os valores para as variáveis com o propósito de satisfazer todas as inequações e minimizar a função objetivo.

Formalmente, a ILP pode ser definida como segue: Determinar valores positivos para um conjunto de variáveis $x_1, x_2, ..., x_n$ que minimizem uma função objetivo $\sum_{j=1}^{n} k_j \cdot x_j$ onde cada k_j é uma constante e representa a métrica. As variáveis estão sujeitas a n inequações da forma $\sum_{j=1}^{n} k_j \cdot x_j \leq c_j$, onde c_j é uma constante e representa a restrição.

Dado um conjunto de objetos funcionais $(o_1, o_2, ..., o_n)$ então o projetista do sistema deve decidir quais objetos funcionais serão implementados em hardware ou software. Se o objeto funcional o_i for implementado no grupo p_j então a variável de mapeamento x_i é iniciada com o valor j que pode ser hardware ou em software. Em termos gerais, se existem n objetos funcionais e m componentes que constituem o sistema então existem m^n possibilidades de mapeamento. Por exemplo, um sistema embarcado com restrições de custo deve ser desenvolvido. Este sistema é composto por quatro objetos funcionais representado por F = {*f1, f2, f3, f4*}.

Cada funcionalidade possui um custo de implementação em hardware e em software. O custo de implementação do hardware é representado por H_p = {*h1, h2 h3, h4*} e o custo de implementação do software é representado por S_p = {*s1, s2, s3, s4*}. O custo de comunicação do sistema é C = {c1, c2, c3}. A Figura 5.1 apresenta o grafo de tarefa deste sistema embarcado.

Como este sistema embarcado é composto por quatro objetos funcionais então existem quatro variáveis de decisão *x1, x2, x3, x4*. Estas variaveis de decisao são representadas matematicamente como um vetor binário e podem assumir os valores 0 ou 1 (0→hardware ou 1→software). Em outras palavras, estas variáveis definem se o vértice do grafo deve ser implementado em hardware ou em software. É importante notar que se existem n funcionalidades entao existiram 2^n atribuições de variáveis. Para o nosso exemplo, existem 16 diferentes maneiras de implementar o sistema.

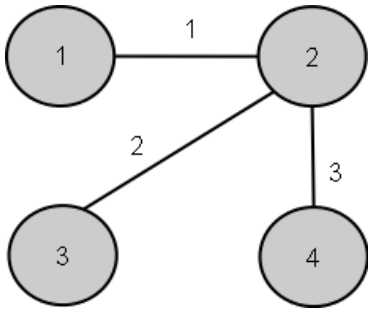

Figura 5.1: Grafo de Tarefa do Sistema.

A comunidade de sistemas embarcados define isso como exploração das alternativas de projeto, pois de todas as possibilidades possíveis devemos escolher somente as alternativas que atendem as restrições do projeto. Para o nosso exemplo, devemos escolher a alternativa que tenha o menor custo de hardware associado. Para isso, deve-se primeiramente calcular a matriz de incidência transposta. Esta matriz é construida da seguinte maneira: (i) define-se o número de linhas e colunas da matriz baseado no número de arestas e vértices do grafo respectivamente, (ii) se a aresta i inicia no vértice j então se atribui -1 ao elemento da matriz E[i,j], (iii) se a aresta i termina no vértice j entao se atribui 1 ao elemento da matriz E[i,j], (iv) se a aresta i não incide no vértice j então se atribui 0 ao elemento da matriz E[i,j].

A equação 5.1 apresenta a matriz de incidência transposta do grafo de tarefa do sistema apresentado na figura 5.1. Como exemplo, a aresta 1 inicia no vértice 1 então se atribui -1 ao elemento E[1,1], a aresta 1 termina no vértice 2 então se atribui 1 ao elemento E[1,2], como a aresta i não inicia e nem termina nos vértices 3 e 4 então se atribui 0 aos elementos E[1,3] e E[1,4] respectivamente.

$$\begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} \tag{5.1}$$

Depois disso, deve-se multiplicar a matriz de incidência transposta pela matriz de decisão. Como a matriz de incidência possui dimensão 3x4 e a matriz de decisão é 4x1 então pode-se efetuar a multiplicação destas duas matrizes resultando em um matriz de tamanho 3x1. Finalmente, aplica-se o resultado da multiplicação da matriz de incidência e

decisão na solução do problema. Deste modo, obtém-se o seguinte problema de otimização:

Minimizar $\sum_{4}^{i=0} h_i \cdot x_i$
Sujeito a $\sum_{4}^{i=0} s\,(1-x) + c\,|E \cdot x| \leq S_0$

Este problema de otimização consiste em minimizar o custo de hardware, mas o sistema está sujeito a uma dada restrição S_0. Para encontrar a solução deste problema, deve-se então aplicar todas as atribuições possíveis (neste caso existem 16 atribuições) e encontrar uma solução que tenha o menor custo de hardware associado e que ao mesmo tempo atenda a restrição do sistema. Como exemplo da aplicação de uma possível atribuição, considera-se o seguinte vetor biário das variáveis de decisão X = {*1,0,0,1*} e calcula-se o custo de software e hardware associado a esta atribuição da seguinte maneira:

$$s_1 - s_1 \cdot x_1 + (-c_1 \cdot x_1 + c_1 \cdot x_2) + s_2 - s_2 \cdot x_2 + (-c_2 \cdot x_2 + c_2 \cdot x_3)$$
$$+s_3 - s_3 \cdot x_3 + (-c_3 \cdot x_2 + c_3 \cdot x_4) + s_4 - s_4 \cdot x_4 \leq S_0$$

Agora, atribuindo x1=1, x2=0, x3=0 e x4=1 na equação acima, tem-se que o custo de hardware H_p = *h1 + h4* e o custo de software S_p = *c1 + c2 + s2 + s3*. O grafo de tarefa do sistema resultante desta atribuição é mostrado na figura 5.2.

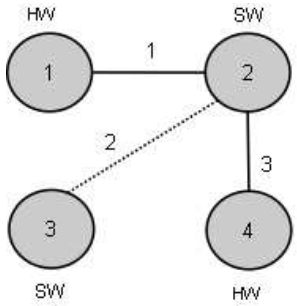

Figura 5.2: Grafo de Tarefa do Sistema para X={1,0,0,1}.

Como mostrado na figura 5.2, pode-se observar que as funcionalidades 2 e 3 serão implementadas em software e as funcionalidades 1 e 4 em hardware. Além disso, como as funcionalidades 2 e 3 estão no mesmo contexto (software) então o custo de comunicação c2, ou seja a aresta c(v2,v3) é desprezada. A ILP produz bons resultados para o problema do particionamento se o sistema consiste apenas de algumas centenas de objetos funcionais e componentes [4]. De outro modo, a ILP requer muito tempo de computação para fornecer

a solução ótima exata do problema. Para particionar grandes sistemas que podem consistir de vários objetos funcionais e componentes do sistema, outras técnicas (p.e. migração de grupo apresentada na próxima seção) devem ser usadas para fornecer uma solução aproximada do problema.

Migração de Grupos

O outro algoritmo que foi implementado nesta dissertação é o migração de grupos (*group migration*) [22]. O algortimo de migração de grupos tem como estratégia de particionamento, para cada objeto, determinar o descréscimo no custo se o objeto for movido para o outro grupo. Então a idéia principal é mover o objeto entre os grupos com o intuito de produzir o menor custo. O algoritmo de migração de grupos é mostrado na figura 5.3.

O procedimento $Move(P, o_i)$ retorna uma nova partição obtida do movimento de um dado objeto para o grupo oposto. Cada objetivo o_i é estendido com a *flag*, movido, que indica se o objeto foi movido ou não. Uma vez movido, o objeto não deveria ser movido novamente. As variáveis *ParticaoAnterior* e *CustoAnterior* representam a partição e o custo anterior para realizar qualquer movimento durante uma iteração do algoritmo. A variável *ObjetoMelhorMovido* é o objeto que, quando movido, produz a melhoria do melhor custo, e a variável *CustoMelhorMovimento* é o custo resultante. A variável MelhorParticaoP representa a partição com o menor custo encontrado durante a movimentação, e a variável *CustoMelhorParticao* representa o custo da partição.

O *loop* mais externo realiza a iteração do algoritmo até que a seqüência gerada de movimentos não melhore o custo. Durante cada iteração é criada uma seqüência de n movimentos, onde n é o número de objetos. Cada movimento é criado movendo temporariamente todos os objetos para ver qual movimentação de objeto resulta no melhor custo, e então movendo o objeto com melhor custo e marcando-o para prevenir de ser movido novamente. Se o custo é o melhor encontrado enquanto criando a seqüência então a partição é salva. Depois de n movimentos terem sidos feitos, o algoritmo verifica se a melhor partição salva é melhor do que a partição que foi iniciada na iteração. Caso seja, então o algoritmo itera novamente. Caso contrário retorna à partição anterior.

A complexidade do algoritmo é dominada pela criação de uma seqüência de n movimentos. Selecionando o melhor objeto para cada movimento requer uma média de $n/2$

```
1 Algoritmo Migração de Grupo
2 P = P_in
3 loop
4     ParticaAnterior = P
5     CustoAnterior = funcObj(P)
6     CustoMelhorParticao = funcObj(P)
7     Para cada (o_i) Faça
8         o_i · movido = FALSO
9     Fim
10    Para cada (i ∈ n) Faça
11        CustoMelhorMovimento = ∞
12        Para cada (o_inaoo_i · movido) Faça
13            Custo = funcObj(Move(P, o_i))
14            Se (Custo < CustoMelhorMovimento) Então
15                CustoMelhorMovimento = Custo
16                ObjetoMelhorMovido = o_i
17            Fim
18        Fim
19        P = Move(P, ObjetoMelhorMovido)
20        ObjetoMelhorMovido · movido = VERDADEIRO
21        Se (CustoMelhorMovimento < CustoMelhorParticao) Então
22            P = MelhorParticaoP = P
23            CustoMelhorParticao = CustoMelhorMovimento
24        Fim
25    Fim
26    Se (CustoMelhorParticao < CustoAnterior) Então
27        P = MelhorParticaoP
28    Senão Retorne ParticaoAnterior
29    Fim
30 Fim
```

Figura 5.3: Algoritmo de Migração de Grupos - *Group Migration* [22]

objetos. Assumindo que a função objetivo também solicita n computações então a complexidade deste algoritmo é $O(n \times n/2 \times n)$, ou $O(n^3)$. Este algoritmo pode ser facilmente estendido para o particionamento de vários componentes. No particionamento de dois componentes como mostrado no algoritmo 5.3, é necessário mover todos os objetos para o seu grupo oposto com o intuito de verificar qual movimentação do objeto produziu o menor custo. Em um particionamento de vários componentes, o algoritmo deve mover o objeto para os outros grupos dentro do sistema. Porém, a complexidade do algoritmo é multiplicada pelo número de grupos. Sabendo que a complexidade do algoritmo para dois componentes é $O(n^2)$, a modificação para vários componentes produz uma complexidade de $O(mn^2)$, onde m é o número de grupos.

5.1.2 Aplicativo para Captura de Log no PC

Com o intuito de analisar as mensagens de texto contidas na memória RAM da plataforma de desenvolvimento dos experimentos, nós desenvolvemos um aplicativo no PC desktop usando a linguagem Java [52]. Para implementar a comunicação entre a plataforma e PC desktop, nós usamos o pacote javax.comm. Este pacote permite que um aplicativo Java comunique com outros periféricos através da porta serial RS-232. A figura 5.4 mostra o método responsável por coletar os bytes através da porta serial.

Depois de capturar os bytes enviados pela plataforma de desenvolvimento através da porta serial, o aplicativo que executa no PC formata estes bytes para serem inseridos em arquivo de texto. Esta formatação leva em consideração o caractere # para formatar os bytes recebidos pela porta serial. Isto significa que quando o nosso aplicativo detecta o símbolo # então significa que uma nova linha deve ser inserida no arquivo de log. O nome do arquivo que fica armazenado no PC tem como identificação a data e horário que foi coletado o log. A extensão deste arquivo de log é ".log" e pode ser aberto por qualquer processador de texto instalado no PC.

5.2 Ferramenta de Terceiros

Esta subseção apresenta o framework de teste unitário e a plataforma de desenvolvimento que foi usada nos estudos de caso desta dissertação.

```
1 public void serialEvent(SerialPortEvent event) {
2     switch (event.getEventType()) {
3     case SerialPortEvent.BI:
4     ⋮
5     case SerialPortEvent.OUTPUT_BUFFER_EMPTY :
6           break;
7     case SerialPortEvent.DATA_AVAILABLE :
8           byte[   ] readBuffer = new byte[5];
9           try {
10                while (inputStream.available() > 0) {
11                      inputStream.read(readBuffer);
12                      readData.CollectData(readBuffer);
13                      inputStream.reset();
14                }
15          } catch (IOException e) {}
16          break;
17    }
18 }
```

Figura 5.4: Aplicativo para Captura de Log no PC

5.2.1 Framework de Teste Unitário

Para realização dos casos de teste implementados nos experimentos desta dissertação, foi utilizada a ferramenta embUnit (Embedded Unit) disponível para download no site [42]. Esta ferramenta nada mais é do que um *framework* de teste unitário para sistemas de software escritos em C. O projeto deste framework é baseado nas ferramentas Junit (para programas escritos em Java) e CUnit (para programas escritos em C/C++) e adaptado para testar software embarcado escrito em C que contém severas restrições no ambiente de execução.

A Figura 5.5 apresenta um exemplo de casos de teste para o código do sensor do oxímetro de pulso. O sensor do oxímetro fornece para o módulo de aquisição os dados de SpO2 e HR. Estes dados são processados pelos módulo e enviado para o micro-controlador através da porta serial RS-232. As funções *fillData* (linha 14) e *clearData* (linha 19) são responsáveis por fornecer os dados provenientes do sensor. Os dados usados nos casos de teste foram obtidos com o uso da ferramenta de captura de log descrita na seção 5.1.2. Com esta ferramenta foi possível coletar uma série de dados reais do sensor. Deste modo,

a função *fillData* simula o sensor coletando dados válidos de SpO2 e HR enquanto que a função *clearData* simula dados inválidos do sensor.

O código de teste apresentado na figura 5.5 possui basicamente dois casos de teste que são *testGetHR* (linha 25) e *testEmptyGetHR* (linha 29). O caso de teste *testGetHR* chama a função *fillData* (linha 26) com o propósito de alimentar as estruturas do componente sensor com os dados de SpO2 e HR. Depois disso, a função *getHR* (linha 27) é chamada através do $TEST\_ASSERT\_EQUAL\_INT$ para comparar o valor real do HR fornecido pelo sensor e o valor de HR fornecido pela função do componente do sensor. Se estes valores forem iguais então o caso de teste passou. Caso contrário, o caso de teste falhou na execução.

A mesma forma de análise é aplicada ao caso de teste *testEmptyGetHR*. A função *clearData* (linha 30) é chamada com o propósito de fornecer dados inválidos do sensor. Depois disso, a função *getHR* (linha 31) é chamada através do $TEST\_ASSERT\_EQUAL\_INT$ para comparar o valor real do HR fornecido pelo sensor e o valor de HR fornecido pela função do componente do sensor. Se estes valores forem iguais então o caso de teste passou. Caso contrário, o caso de teste falhou na execução.

É importante observar que para criar os casos de teste para o componente do sensor, deve-se *(i)* incluir as referências para o *framework* do embUnit e do componente sensor (linhas 2 e 4), *(ii)* incluir, se necessário, código para as funções de inicialização *setUp* e finalização *tearDown* do caso de teste (linhas 7 e 11), *(iii)* incluir todos os casos de teste na função $new\_TestFixture$ adicionando uma string com o nome do teste (linhas 36 e 37) e (iv) retornar com ponteiro para a suíte de teste que acabou de ser criada.

A Figura 5.6 mostra o programa principal para a execução dos casos de teste.

A próxima subseção apresenta os principais componentes da plataforma de desenvolvimento que foi usada nos estudos de caso desta dissertação.

5.3 Plataforma de Desenvolvimento

Para a implementação dos protótipos do oxímetro de pulso, soft-starter digital e simulador do motor, foi usado a plataforma de desenvolvimento 8051NX da Microgênios [37]. Esta plataforma possui o micro-controlador AT89S8252 da ATMEL que é 100 % de compatibil-

```
1 Casos de Teste para o Sensor
2 #include < embUnit/embUnit.h >
3 #/ * embunit : include = + * /
4 #include"../src/drivers/sensor.h"
5 #/ * embunit : include = − * /
6 unsigned int i = 0;
7 static void setUp(void){
8     initSensor();
9     initStatus();
10}
11 static void tearDown(void){
12    /* terminate */
13}
14 static void fillData(void){
15    for(i=0; i<380; i++){
16          collectData(fullCollection[i]);
17    }
18 }
19 static void clearData(void){
20    for(i=0; i<380; i++){
21          collectData(emptyCollection[i]);
22    }
23 }
24 /*embunit:impl=+ */
25 static void testGetHR(void){
26    fillData();
27    TEST_ASSERT_EQUAL_INT(87, getHR());
28 }
29 static void testEmptyGetHR(void){
30    clearData();
31    TEST_ASSERT_EQUAL_INT(0, getHR());
32 }
33 /*embunit:impl=- */
34 TestRef sensorTest_tests(void) {
35    EMB_UNIT_TESTFIXTURES(fixtures) {
36    new_TestFixture("testGetHR", testGetHR),
37    new_TestFixture("testEmptyGetHR", testEmptyGetHR),
38 };
39 EMB_UNIT_TESTCALLER(sensorTest, "sensorTest", setUp, tearDown, fixtures);
40 return(TestRef)&sensorTest;
41 };
```

Figura 5.5: Exemplo de Teste Unitário usando embUnit

```
1 Programa Principal
2 #include < embUnit/embUnit.h >
3 TestRef sensorTest_tests(void);
4 int main (int argc, const char* argv[ ]){
5     TestRunner_start();
6     TestRunner_runTest(sensorTest_tests());
7     TestRunner_end();
7     return 0;
8 }
```

Figura 5.6: Executor de Casos de Teste do Sensor

idade com a família do 8051 [5]. Além disso, esta plataforma de desenvolvimento possui 12 Kbytes de memória flash, 2 Kbytes de memória EEPROM, 256 bytes de memória RAM, 32 portas de entrada e saída e comunicação serial UART.

A plataforma ainda possui 32 Kbytes de memória RAM externa, LCD 16x2 (2 linhas e 16 colunas) com luz de fundo, um relógio de tempo real (RTC) PCF8583 que é acessado via barramento I^2C, um conversor de 4 canais A/D e 1 D/A com resolução de 8 bits PCF8591 também acessado via barramento I^2C e porta para expansão de memória com 34 vias. A Figura 5.7 apresenta a plataforma de desenvolvimento usada nos estudos de caso.

Figura 5.7: Plataforma de Desenvolvimento 8051NX da Microgênios [37].

Esta plataforma de desenvolvimento foi escolhida pelo fato de que os microprocessadores da família 8051 possuem um baixo custo e um poder de processamento adequado para o propósito dos protótipos desenvolvidos nesta dissertação. Além disso, esta plataforma de desenvolvimento possui um conjunto de componentes de hardware interconectados que aceleram o processo de desenvolvimento do produto. Um outro ponto

importante a ser levado em consideração é que esta plataforma possui um bom nível de estabilidade para ser utilizada em projetos de sistemas embarcados de tempo real. A seção 6 descreve em detalhes os principais motivos da escolha desta plataforma para os protótipos que foram desenvolvidos.

5.4 Resumo

Esta seção descreveu as ferramentas que foram desenvolvidas nesta dissertação de mestrado em cooperação com os trabalhos [29, 40]. Nós implementamos dois algoritmos de particionamento conhecidos como programação linear inteira e migração de grupos. O algoritmo baseado em ILP define um conjunto de variáveis, um conjunto de inequações lineares que restringem o valor das variáveis e uma função linear simples que serve como uma função objetivo [22]. O objetivo principal é escolher os valores para as variáveis com o propósito de satisfazer todas as inequações e minimizar a função objetivo.

O algoritmo baseado em migração de grupos tem como estratégia de particionamento, para cada objeto, determinar o descréscimo no custo se o objeto for movido para o outro grupo. Então a idéia principal é mover o objeto entre os grupos com o intuito de produzir o menor custo. Ambos os algoritmos foram implementados usando a linguagem de programação Java [52]. Nós também desenvolvemos um aplicativo no PC com o intuito de analisar as mensagens de texto contidas na memória RAM da plataforma de desenvolvimento. Esta ferramenta também foi desenvolvida usando a linguagem de programação Java [52].

Além disso, esta seção apresentou o framework de teste unitário embUnit usado para testar código em C de sistemas embarcados [50]. EmbUnit não solicita bibliotecas padrões do C. Todos os objetos são alocados para uma área constante da memória do PC desktop. Um exemplo de casos de teste para o código do sensor do oxímetro de pulso foi apresentado com o intuito de mostrar como este framework foi utilizado no desenvolvimento dos protótipos desta dissertação. Finalmente, esta seção apresentou os principais componentes da plataforma de desenvolvimento que foi usada para a implementação dos protótipos do oxímetro de pulso, *soft-starter* digital e simulador do motor de indução.

Capítulo 6

Estudos de Caso

Esta seção tem como objetivo descrever as características, requisitos, arquitetura, os testes unitários e funcionais que foram desenvolvidos para os protótipos desta dissertação de mestrado. Como mencionado na Seção 1, foram desenvolvidos basicamente três protótipos que são: *oxímetro de pulso, soft-starter digital e simulador do motor de indução monofásico*. Para cada protótipo, nós instanciamos a arquitetura e API da plataforma com o intuito de desenvolver o produto baseado nas restrições da aplicação. Para o desenvolvimento dos três protótipos, usou-se a mesma plataforma que foi descrita na Seção 5.3.

Aém disso, como apresentado na Seção 2.3, um dos pontos fortes de métodos ágeis é o enfoque na disciplina de engenharia de software conhecida como testes. As atividades de teste e depuração de programas correspondem a uma grande parcela do seu custo total. No entanto, estas atividades são de extrema importância durante o desenvolvimento de software e, de certa forma, eliminá-las do processo de desenvolvimento pode resultar na diminuição da qualidade final do produto. Sendo assim, esta seção apresenta as técnicas de teste que foram utilizadas para verificar a corretude lógica e temporal das funções assim como validar os requisitos em alto nível do sistema.

6.1 Protótipo do Oxímetro de Pulso

Esta seção descreve a área de aplicação e características do protótipo do oxímetro de pulso que foi desenvolvido com o intuito de validar a metodologia de desenvolvimento TXM. O

protótipo do oxímetro de pulso foi escolhido como estudo de caso pelo fato de representar um exemplo de um sistema embarcado de tempo real crítico. Além disso, o oxímetro pertence ao domínio de aplicações médicas na área de sistemas embarcados. Esta seção descreve ainda a arquitetura do sistema e as práticas de teste que foram usadas durante o desenvolvimento do protótipo.

6.1.1 Características do Protótipo

Em termos gerais, o oxímetro de pulso é um equipamento responsável por mensurar a saturação de oxigênio no sistema sanguíneo do paciente usando um método não-invasivo [1]. Um oxímetro de pulso pode ser usado em muitas circustâncias, como checar se a saturação de oxigênio está abaixo ou não do nível aceitável, quando um paciente é sedado com anestésico para um procedimento cirúrgico [7]. Este equipamento é amplamente usado em unidades de centro cirúrgicos em hospitais. O oxímetro de pulso é freqüentemente anexado a um monitor médico para que o médico possa verificar a oxigenação do paciente, embora equipamentos portáteis sejam também usados. A maioria dos oxímetros de pulso mostram também o batimento cardíaco. Figura 6.1 mostra um oxímetro de pulso portátil.

Figura 6.1: Oxímetro de Pulso (fonte: http://www1.vghtpe.gov.tw).

O diagrama de bloco deste equipamento é mostrado na Figura 6.2. O diagrama de bloco consiste de um micro-controlador, um sensor que é composto de dois diodos emissores de luz (LED - *light emitting diodes*) que serve como fontes de luz e um fotodiodo (photodiode) que atua como um receptor de luz, um *display* para mostrar a quantidade de

[1] O termo não-invasivo significa um procedimento médico que não penetra na pele do paciente.

oxigênio do sangue do paciente, e um teclado para entrar com as informações necessárias para configurar o equipamento. O sensor é posicionado de tal maneira que o LED e fotodiodo se opõe de forma que a luz atravessa o dedo.

Como mostrado na Figura 6.2, o sensor pode também solicitar uma interface para amplificar e filtrar o sinal. O micro-controlador controla a sincronização e amplitude do sinal e envia uma série de pulsos não simultâneos para o sensor. Ambos os LEDs do sensor geram, respectivamente, pulsos de radiação vermelha e infra-vermelho que atravessam o dedo do paciente. Depois de cruzar o dedo, o fotodiodo captura o nível de radiação e o envia para o micro-controlador.

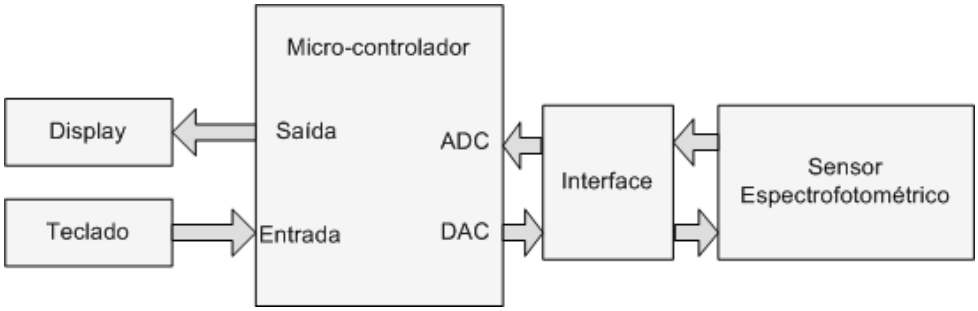

Figura 6.2: Diagrama de Bloco do Oxímetro de Pulso.

Finalmente, o micro-controlador realiza o cálculo baseado na variação da absorção entre o oxigênio rico e pobre no sangue do paciente. Esta diferença possibilita o micro-controlador calcular o nível de saturação do oxigênio e mostrar o resultado em um display. Devido a sua simplicidade e velocidade (basta colocar no dedo e os resultados são mostrados no *display* dentro de poucos segundos), oxímetros de pulso são de importância crítica em medicina de emergência e também são bastante usados em pacientes com problemas cardíacos e respiratórios assim como pilotos operando em aviões não pressurizados acima de 3048 metros, onde oxigênio suplementar é necessário [7].

Em linhas gerais, as principais características que foram implementadas para o oxímetro de pulso incluem: *(i)* o nível de SpO_2 e HR deve ser mostrado de acordo com a taxa de amostragem definida pelo usuário, *(ii)* o usuário deve ser capaz de salvar os valores de SpO_2 e HR na memória do dispositivo, *(iii)* a interface com o usuário deve ter um teclado e display, *(iv)* o *projeto* do sistema deve ser altamente otimizado para o ciclo de vida e eficiência, *(v)* o número de defeitos deve ser o menor possível, e *(vi)* a dissipação de potência do sistema final deveria ser aproximadamente 500 mW. Para uma lista completa

dos requisitos funcionais dos drivers, interface com o usuário, aplicação e os requisitos não funcionais, refira-se ao apêndice B.1.

6.1.2 Arquitetura do Sistema

Como mencionado no capítulo 2.1, a solução de um sistema embarcado geralmente envolve a definição de componentes de hardware e software. As Figuras 6.3 e 6.4 apresentam uma visão geral da arquitetura de hardware e software deste experimento respectivamente. Como pode ser observado na Figura 6.3, a solução do hardware consiste essencialmente da plataforma de aquisição de dados (OEM III da Nonin) e da plataforma desenvolvimento[2].

Esta plataforma de aquisição de dados OEM III da Nonin fornece uma maneira simples de incorporar a oximetria de pulso em dispositivos médicos com um baixo custo e tempo para mercado reduzido [27]. Este módulo OEM III é um projeto compactado, com baixa dissipação de potência (aproximadamente 29 mW) e fornece máxima flexibilidade para ser integrado em outros tipos de sistemas. Além disso, este módulo suporta um amplo espectro de sensores e é adequado para ser usado em ambientes com movimentos.

O sensor do oxímetro de pulso (também da Nonin) é conectado ao módulo de aquisição de dados (OEM III da Nonin) e fornece os dados de saturação do oxigênio (SpO_2) e batimento cardíaco (HR) [27]. Estes dados são adquiridos e processados pelo módulo OEM III e posteriormente mostrados no display do dispositivo. Este módulo possui uma interface de comunicação com sensor, um componente ASIC que é responsável por toda a inteligência do módulo, e uma interface de comunicação com PC através da porta serial RS-232. Sendo assim, o componente ASIC captura os dados do sensor, calcula os valores de SpO_2 e HR e finalmente fornece estes dados na porta RS-232 do módulo.

A plataforma de desenvolvimento possui um conjunto de componentes de hardware que acelera o processo de desenvolvimento do sistema e reduz custos do projeto. Esta plataforma possui um conversor serial RS-232, micro-controlador AT89S8252, relógio de tempo real PCF8583, conversor de 4 canais A/D e 1 D/A com resoluções de 8 bits e 4 portas de expansão (32 portas de E/S). Além disso, esta plataforma possui 12 KB de memória flash (memória de programa) e 32KB de memória RAM (memória de dados). A plataforma de desenvolvimento se comunica com a plataforma de aquisição de dados

[2]Esta plataforma é baseada no micro-controlador AT89S8252 da família do 8051.

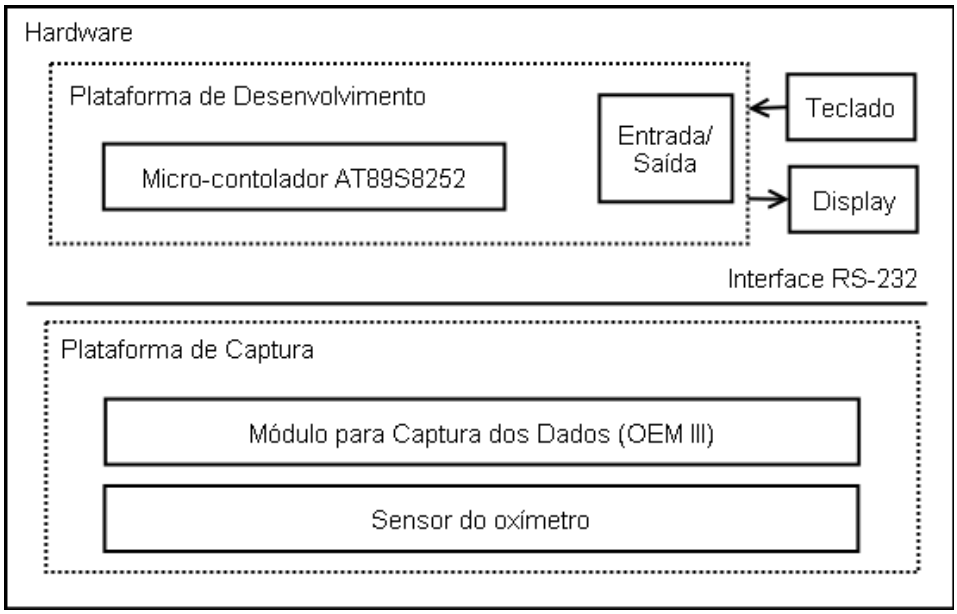

Figura 6.3: Arquitetura do Hardware - Experimento 1.

através da porta serial RS-232 em uma taxa de comunicação de 9600 bps. Com o propósito de definir uma interface com o usuário, foram utilizados um LCD 16x2 (16 colunas e 2 linhas) e um teclado de 5 botões (Inicia (*Start*), Para (*Stop*), Incrementa (*Up*), Decrementa (*Down*) e Seleciona (*Select*)).

Como mostrado na Figura 6.4, a arquitetura de software é composta basicamente dos drivers da plataforma (serial, LCD, teclado, temporizador e sensor), um componente de software que permite depurar o código através do armazenamento de dados na memória RAM (Log do Sistema), uma API que possibilita a camada de aplicação realizar chamadas nos drivers da plataforma (API da plataforma) e a camada de aplicação propriamente dita (Aplicativo). Estes drivers da plataforma definem as classes de software que são dependentes do hardware. A aplicação do oxímetro de pulso se comunica com os drivers dos dispositivos através da API da plataforma.

Esta API é uma abstração dos periféricos (drivers dos dispositivos) e representa uma abstração única da arquitetura da plataforma. Com esta API assim definida, o software de aplicação pode reusar esta API para cada instância da plataforma (maximização de reuso - leia processo 4.5.3). Deste ponto de vista, a API pode ser considerada com uma plataforma em si. Sendo assim, a união da camada inferior (o conjunto de componentes que definem a arquitetura do hardware) com a camada superior (o conjunto de compo-

nentes que estão abaixo da API da plataforma) pode ser considerada como uma única plataforma, a plataforma do sistema, como definido na seção 4.1.

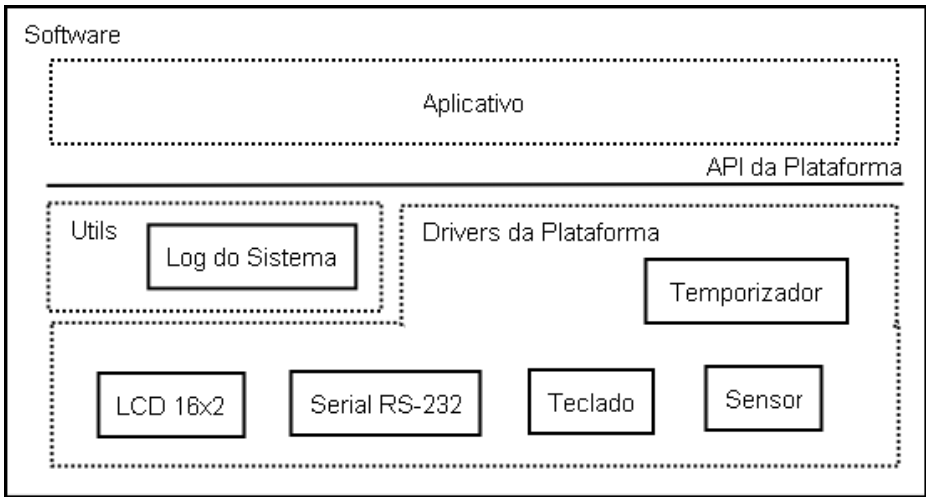

Figura 6.4: Arquitetura do Software.

A Figura 6.5 apresenta o diagrama de componentes[3] do oxímetro de pulso. O subsistema "Drivers da plataforma" consiste de 5 componentes (cada driver pode ser visto como um componente de software) que são: *display, teclado, serial, sensor e temporizador.* O componente do display fornece funções para realizar escrita de dados do LCD 16x2 acoplado a plataforma de desenvolvimento. O componente do teclado fornece funções para detectar as teclas que foram pressionadas pelo usuário. O componente da serial fornece meios de acessar o canal de comunicação entre a plataforma de desenvolvimento e o mundo externo através da porta RS-232. O componente do sensor possibilita a camada de aplicação acessar os dados de SpO2 e HR. O componente do temporizador fornece funções para disparar um serviço de interrupção da plataforma em tempos determinados.

O componente "Log do Sistema" foi desenvolvido com o propósito de depurar o código e armazenar conteúdo de variáveis na memória. Este componente faz uso de serviços fornecidos pelos drivers da plataforma. Uma API para a plataforma de desenvolvimento deste experimento foi desenvolvida com o propósito de maximizar o reuso do software em outras aplicações que derivam desta plataforma. Sendo assim, o software da camada de aplicação acessa os componentes de hardware da plataforma através de uma API em um alto nível de abstração. Serviços como mudar a taxa de transmissão da serial,

[3]A definição de componente, neste trabalho, significa a implementação de uma ou mais responsabilidades fortemente relacionadas e cada componente tem interfaces claramente definidas.

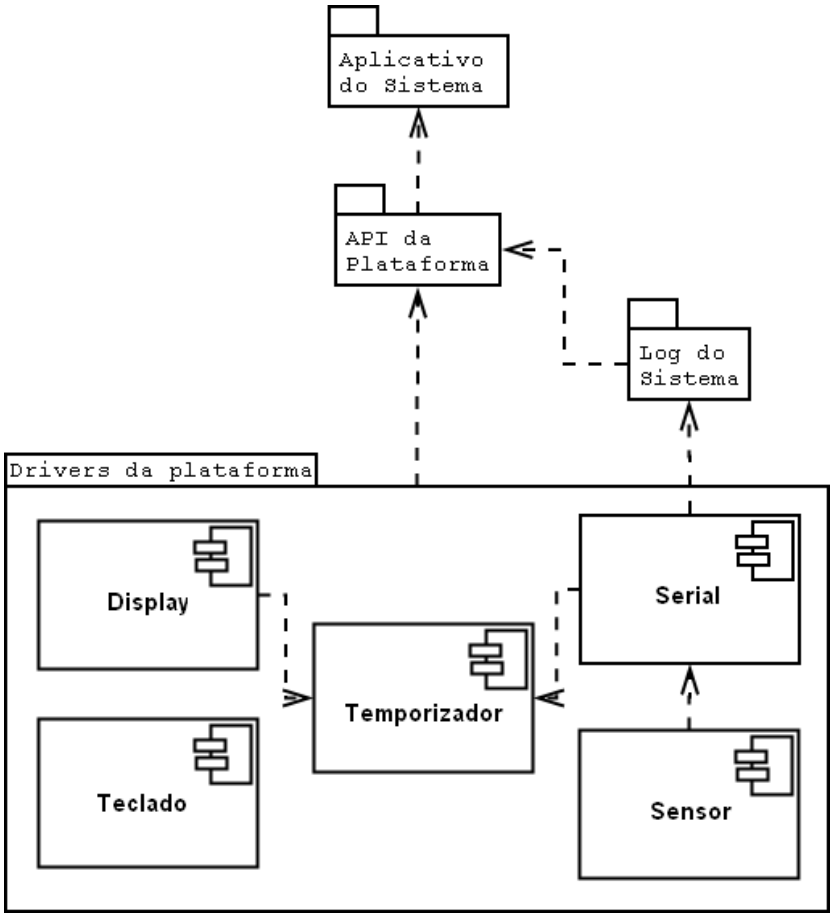

Figura 6.5: Diagrama de Componentes do Sistema.

depurar código, escrever mensagens de texto no LCD, receber eventos do mundo externo, enviar/receber pacote de dados pela serial entre outros podem facilmente serem realizados através da chamada de funções desta API.

A Figura 6.6 mostra o diagrama de estados da interface com o usuário do oxímetro de pulso. Como pode ser observado nesta figura, o software de aplicação inicia no item "Ajustar tempo de amostragem". Caso o usuário deseje alterar a taxa de amostragem do sinal então ele deve pressionar as teclas incrementa/decrementa disponíveis no teclado do dispositivo. Se o usuário pressiona a tecla início então o sistema começa a mostrar os dados de SpO_2 e HR no display. Se o usuário pressionar a tecla "Seleciona" então o software de aplicação passa para o estado "Habilita armazenamento". Esta opção permite que o usuário salve dados de SpO_2 e HR na memória RAM do micro-controlador. O processo para habilitar/desabilitar é realizado pressionando os botões incremento/decremento respectivamente.

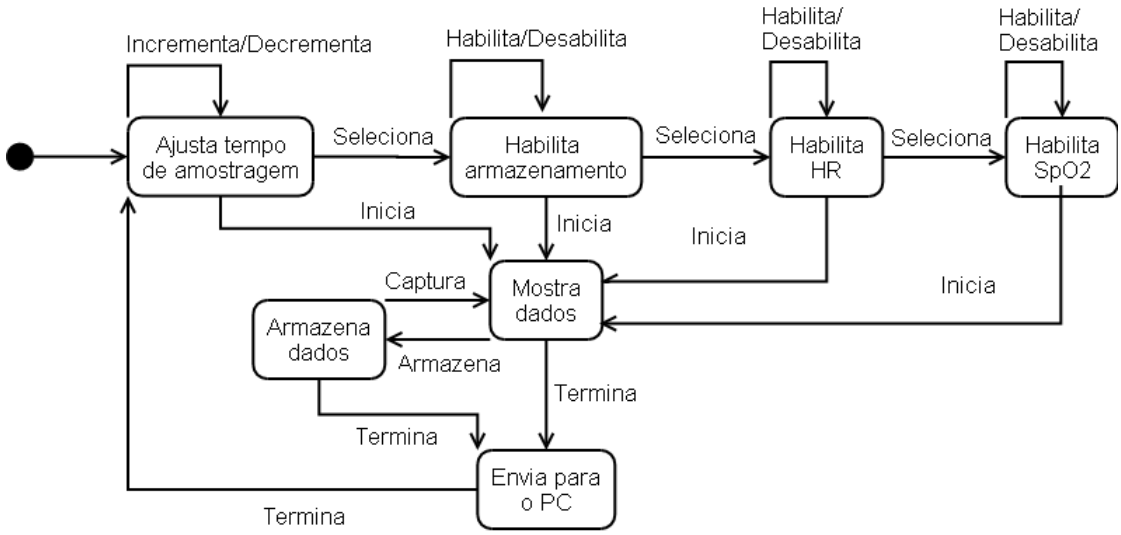

Figura 6.6: Diagrama de Estados.

O usuário passa do estado "Habilita armazenamento" para "Habilita SpO_2" pressionando a tecla "Seleciona". Os botões incremento/decremento possibilitam que o usuário habilite/desabilite a exibição dos dados de SpO_2 no display do dispositivo. A mesma ação acontece para o item "Habilita HR". É importante notar que o usuário pode passar para o estado de exibição dos dados a partir de qualquer estado do diagrama. Todos os itens iniciam habilitados e o tempo de amostragem é ajustado para o valor de 1 segundo no início da aplicação. Um outro ponto importante a ser mencionado é que o usuário pode parar o software de aplicação em qualquer estado do sistema. Depois de pressionar a tecla "termina", o sistema vai para o estado inicial que é "Ajustar tempo de amostragem". Esta situação foi omitida do diagrama com o intuito de não poluir visualmente o diagrama.

No momento em que o usuário pressiona a tecla "termina", o sistema solicita que o usuário conecte o cabo serial do PC no oxímetro de pulso. Depois de confirmar a conexão do cabo, o usuário pressiona a tecla "início" e o processo de transferência de dados é iniciado. Foi implementado um mecanismo para fornecer o status do progresso de transferência para o usuário. Desta forma, o progresso é mostrado em termos percentuais. Quando o buffer circular implementado no componente de log é esvaziado então o sistema volta para o estado inicial.

A Seção C.1 do apêndice desta dissertação descreve as funções públicas dos módulos dos drivers (componentes) e o software de aplicação desenvolvido para o projeto do oxímetro de pulso. A próxima subseção descreve as técnicas de teste que foram uti-

```
1 #define TARGET 1 /*1- >code will run on the target platform*/
2                  /*0- >code will run on the PC*/
3 void serial_init(){
4    #if(TARGET)
5         SCON = 0x50;
6         TMOD = 0x20;
7         TR1 = 1;
8         IE = 0x90;
9         TH1 = 253;
10   #endif
11 }
```

Figura 6.7: Técnica para rodar o código na plataforma alvo e PC

lizadas para verificar a corretude lógica e temporal do oxímetro de pulso assim como validar requisitos do sistema.

6.1.3 Testes Unitários e Funcionais

Como o projeto do oxímetro de pulso possui aproximadamente 80 funções, apenas um pequeno conjunto de testes serão explorados nesta seção. A seleção dos casos de teste foram realizadas baseada na importância da função para o sistema, na complexidade da função e no grau de adaptação[4] das técnicas de teste utilizadas para verificação da função e validação do requisito.

Para executar os casos de teste em uma maneira automatizada, nós também tivemos que implementar um mecanismo para executar o software embarcado tanto no PC desktop quanto na plataforma alvo descrita na Seção 5.3. Para este propósito foram utilizados *#if* and *#else* com o propósito de isolar o código que depende de hardware, e contorná-lo quando o software estivesse executando no PC. A Figura 6.7 mostra um exemplo de aplicação desta técnica.

Para aqueles componentes de software que tocavam no hardware, nós os dividimos em duas diferentes classes: componentes que controlam o hardware e componentes que são guiados pelo ambiente. Para este último, a estratégia adotada foi substituir dados

[4]Tendo em vista que as técnicas de teste propostas pelos métodos ágeis focam em aplicações de PC, algumas adaptações foram necessárias para testar o software embarcado dos experimentos desenvolvidos nesta dissertação.

coletados a partir do sensor em tempo real por dados contidos em arquivo de acesso seqüêncial. A figura 6.8 mostra um exemplo de aplicação desta estratégia que foi utilizada neste experimento.

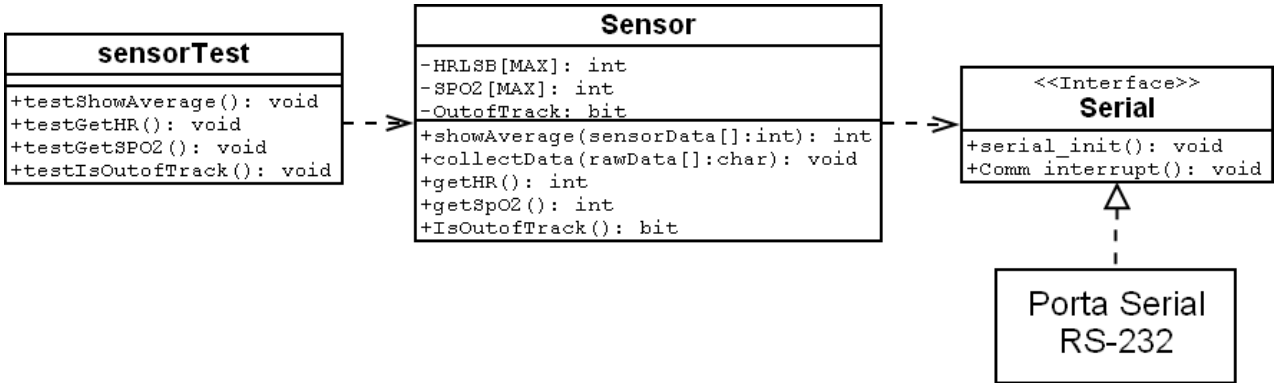

Figura 6.8: Componente controlado pelo ambiente.

Conforme mostrado na Figura 6.8 , o módulo Sensor implementa as funcionalidades de captura dos dados do sensor que está acoplado à plataforma de aquisição. Este módulo recebe dados do sensor através de uma interrupção produzida pela porta serial que ocorre com uma taxa de transferência de 9600 bps. Sendo assim, a interface serial define a interação entre o hardware e o software da plataforma. Em vez de utilizar os dados que vem do sensor através desta interface, desenvolve-se um módulo, neste caso chamado de *sensorTest*, que fornece os dados provenientes do sensor. Esta implementação elimina a necessidade de termos o hardware do sensor acoplado a plataforma de desenvolvimento. Além disso, este tipo de caso de teste pode ser executado de forma automatizada.

No entanto, para o caso onde o componente de software controla o hardware, nós tivemos que executar os casos de teste manualmente na plataforma alvo. Por exemplo, o componente de software do *display* tem essencialmente duas funções que são *lcd_clean* e *lcd_printf*. A implementação de ambas funções são dependentes de hardware. Sendo assim, nós definimos um conjunto de comandos para ser enviado para o hardware do display e os resultados esperados destes comandos. A aplicação que contém os comandos deve executar na plataforma alvo com o propósito de testar todas as posições de escrita do LCD. A figura 6.9 mostra esta técnica de teste aplicada para testar o componente de software do display.

As próximas seções fornecem exemplos de testes unitários e funcionais implementados

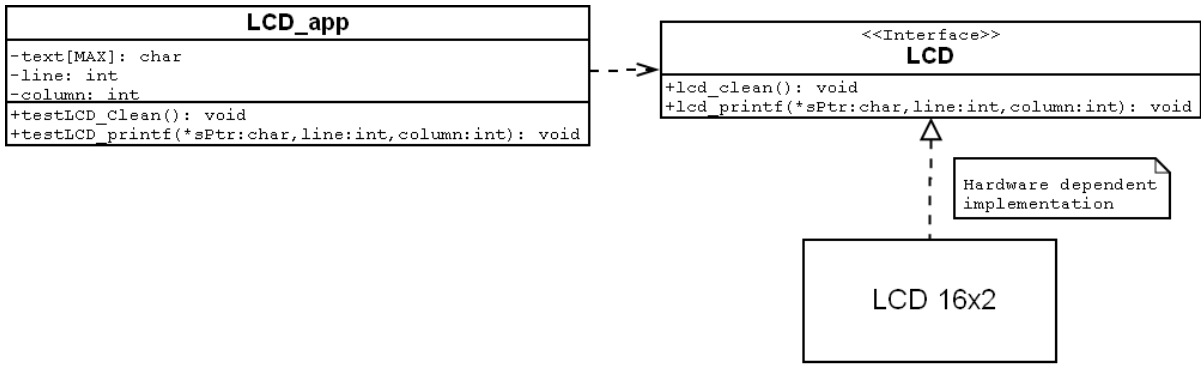

Figura 6.9: Componente que controla o hardware do display.

com a ferramenta de framework de teste unitário conhecida como embUnit [50].

Testes Unitários

Módulo Sensor

getHR: Esta função fornece o valor do batimento cardíaco no modo padrão.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se

1. Fornecer um conjunto de bytes que contenham os dados coletados pelo sensor (p.e., HR);

2. Calcular o valor esperado de HR e comparar este valor com o valor retornado pela função *getHR*;

3. Mensurar o tempo gasto desde a captura do dado através da porta serial até o cálculo do valor de HR.

Código de Teste:

```
static void fillData(void){
    for(i=0; i<380; i++){
        collectData(fullCollection[i]);
    }
}
```

```
static void testGetHR(void){
    fillData();
    TEST_ASSERT_EQUAL_INT(87, getHR());
}
```

Tempos Medidos:

Mínimo: 0.238 ms Máximo: 0.256 ms Média: 0.251 ms

Avaliação:

Para testar a corretude lógica da função, foi necessário capturar os dados do sensor e armazenar em um arquivo para ser utilizado na execução do caso de teste. O código de teste acima mostra que a função *fillData* fornece os dados do sensor para a estrutura *dataFormat*, que está contida no componente sensor, através da chamada de função *collectData*. A função *testGetHR* calcula e retorna o dado de HR disponível na estrutura de dados do componente sensor com o intuito de ser comparado com o valor esperado da variável. Para este caso, foram realizadas três coletas distintas para alimentar os dados do caso de teste. Em todos os casos, a função passou no teste realizado.

Para testar a corretude temporal da função, foi necessário capturar o valor do temporizador no início da função *collectData* e logo após preencher a estrutura de dados que contém os dados de HR na função *fillArrays*. Para saber o tempo gasto nesta função, bastou-se subtrair o tempo final do inicial. Foram realizadas medidas durante 1 minuto desta função com o sistema em pleno funcionamento. De acordo com a taxa de comunicação da porta serial de 9600 bps, tem-se que o intervalo de tempo entre duas recepções de byte pela serial do 8051 é de 0.833 ms. O maior tempo medido para esta função foi de 0.256 ms. Portanto, esta função atendeu os *deadlines* da comunicação serial.

IsOutofTrack: Esta função verifica a ausência de bons sinais de pulso de saturação de oxigênio e batimento cardícado.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se:

1. Fornecer um conjunto de bytes que não contenham bons sinais de pulso do sensor;

2. Verificar se a função retorna verdadeiro, ou seja, se o sensor não está realmente coletando bons sinais de pulso;

3. Mensurar o tempo gasto desde a captura do dado através da porta serial até a verificação do status do sinal.

Código de Teste:

```
static void clearData(void){
    for(i=0; i<380; i++){
        collectData(emptyCollection[i]);
    }
}
static void testIsOutofTrack(void){
    clearData();
    TEST_ASSERT_EQUAL_INT(1, IsOutofTrack());
}
```

Tempos Medidos:

Mínimo: 0.268 ms Máximo: 0.268 ms Média: 0.268 ms

Avaliação:

Para testar a corretude lógica desta função, foi necessário capturar sinais de pulso com baixa qualidade. Estes dados foram armazenados em um arquivo para ser usado pelo código de teste mostrado acima. A função *clearData* fornece sinais de pulso com baixa qualidade para a estrutura *statusByte* do componente sensor. Sendo assim, a função *IsOutofTrack*, que é chamada logo em seguida, verifica o byte de status do sensor e retorna verdadeiro caso não exista bons sinais de pulso. Para este caso, foram realizadas três coletas distintas para alimentar os dados do caso de teste. Em todos os casos, a função passou no teste realizado.

Para testar a corretude temporal da função, foi necessário capturar o valor do temporizador no início da função *collectData* e logo após a função *checkStatus* que preence a estrutura *statusByte*. É importante observar que o byte de status do sensor é sempre o primeiro byte enviado pela porta serial. Ou seja, de todos os pacotes enviados pelo sensor através da serial, o byte de status está sempre contido e é sempre o primeiro. Sendo assim, a função *IsOutofTrack* teve um tempo medido para todos os casos de 0.268 ms.

Foram realizadas medidas durante 1 minuto desta função com o sistema em pleno

funcionamento. De acordo com a taxa de comunicação da porta serial de 9600 bps, tem-se um *deadline* de 0.833 ms para recepção de pacotes. De acordo com os tempo medidos, esta função atendeu os *deadlines* para recepção dos dados.

Módulo Teclado

checkPressedButton: Esta função verifica se uma dada tecla foi pressionada pelo usuário do oxímetro.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se

1. Emular a ação de pressionar um botão do teclado;

2. Verificar se a função retorna a tecla pressionada;

3. Mensurar o tempo gasto desde a capturar do dado através da porta serial até a verificação do status do sinal.

Código de Teste:

```
enum Key_Value {startButton=1, stopButton, upButton, downButton, selectButton};
enum Key_State {START=0xFE, STOP=0xFD, UP=0xEF, DOWN=0xDF, SELECT=0xBF};

static void testStartButton(void){

    TEST_ASSERT_EQUAL_INT(startButton, checkPressedButton(START));
    TEST_ASSERT_EQUAL_INT(stopButton, checkPressedButton(STOP));
    TEST_ASSERT_EQUAL_INT(upButton, checkPressedButton(UP));
    TEST_ASSERT_EQUAL_INT(downButton, checkPressedButton(DOWN));
    TEST_ASSERT_EQUAL_INT(selectButton, checkPressedButton(SELECT));
}
```

Tempos Medidos:

Mínimo: 0.041 ms Máximo: 0.064 ms Média: 0.053 ms

Avaliação:

Para testar a corretude lógica desta função, criaram-se casos de teste para simular a ação do usuário de pressionar botões do teclado. Como cada botão do teclado está

conectado à porta de E/S da plataforma, a ação de pressionar um botão faz com que altere o valor das portas E/S. Sendo assim, a função *checkPressedButton* é chamada com o valor da porta que representa a tecla pressionada. Para a execução dos casos de teste, a função *checkPressedButton* foi chamada 5 vezes (número de botões do teclado) passando como parâmetro o valor da porta correspondente ao botão pressionado. A função passou para todos os casos de teste executados.

Para testar a corretude temporal da função, foi necessário capturar o valor do temporizador no início da função *checkPressedButton* e logo antes de chamar a função correspondente a tecla pressionada. O temporizador da plataforma foi configurado para verificar o valor das portas de E/S a cada 400 ms. O tempo de execução mensurado para o pior caso da função *checkPressedButton* foi de 0.064ms. Sendo assim, o tempo total para verificar uma tecla pressionada é de 400.064 ms. Este intervalo de tempo é ideal para balancear o desempenho do processador e o tempo para reagir a um evento externo do usuário.

Módulo Temporizador

initTimer0s: Esta função tem como objetivo configurar o temporizador para ser disparado em uma escala de segundos de acordo com o valor passado na chamada da função.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se

1. Configurar o temporizador 0 para disparar a cada segundo;
2. Mensurar o tempo de disparo do temporizador na aplicação;

Código de Teste:

```
static void testInitTimer0s1(void){
    unsigned int times=1;
    initTimers();
    TEST_ASSERT_EQUAL_INT(20, initTimer0s(times));
}
```

Tempos Medidos:

Mínimo: 1.0000794 ms Máximo: 1.0000794 ms Média: 1.0000794 ms

Avaliação:

Para testar a corretude lógica desta função, criaram-se vários casos de teste considerando tempos distintos de disparo do temporizador. Para mostrar o processo de teste desta função, o código acima mostra apenas o exemplo para o tempo de disparo de 1 segundo do temporizador. A função *initTimer0s* retorna o número de vezes que a rotina de interrupção deve ser tratada antes de enviar o sinal para a aplicação. Para termos medidos em segundos, a função *initTimer0s* configura o temporizador para disparar a cada 50 ms (caso não tenha nenhum valor já previamente configurado pela função *initTimer0ms*). Sendo assim, para obter o tempo de disparo de 1 segundo, o componente temporizador deve aguardar por 20 chamadas da rotina que trata o temporizador. Depois destas 20 chamadas, um sinal é enviado para a camada de aplicação com o propósito de executar um conjunto de ações definidas em tempo de compilação. A função passou para todos os casos de teste executados.

Para testar a corretude temporal desta função, foi necessário capturar o valor do temporizador no início da função *system_tick* que trata a interrupção do temporizador. Sendo assim, para calcular o tempo de disparo, bastou-se subtrair o valor que é carregado o temporizador pelo valor lido no início da função *system_tick*. Durante a execução deste caso de teste foi detectado que o temporizador estava disparando a cada 1.083 segundos (um erro relativamente alto considerando a criticidade da aplicação). Para resolver este problema, foi alterado o valor de carga do temporizador para um valor que minimizasse o erro nos disparos. Sendo assim, foi obtido no final destes ajustes um tempo de disparo de 1.0000794 ms.

Módulo Log do Sistema

logd: Esta função é usada para inserir um elemento no buffer circular.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se:

1. Ajustar o tamanho do buffer para um número inteiro finito X e inserir a mesma quantidade X de elementos no buffer para depois removê-los um a um;

2. Mensurar o tempo necessário para inserir um elemento no buffer.

Código de Teste:

```
static void testCircularBuffer(void)
    int senData[ ]={1, -128, 98, 88, 59, 1, -128, 90, 0, -37};
    int i;
    initLog(5);
    for(i=0; i<10; i++)
        insertLogElement(senData[i]);
    }
    for(i=5; i<10; i++)
        TEST_ASSERT_EQUAL_INT(senData[i], removeLogElement());
    }
}
```

Tempos Medidos: 0.086 ms

Mínimo: 0.086 ms Máximo: 0.086 ms Média: 0.086 ms

Avaliação:

Para testar a corretude lógica desta função, criou-se um caso de teste para inserir e remover elementos do buffer circular. O código de teste acima mostra os passos necessários para a execução do teste. Nesta situação, o buffer circular foi inicializado e depois foi inserido um conjunto de elementos no buffer. O retorno da função *removeLogElement* foi comparada com os valores inseridos no buffer circular. A função passou em todos os casos de teste executados.

Para verificar a corretude temporal desta função, foi apenas mensurado o tempo necessário para inserir um elemento no buffer circular. Sendo assim, foi necessário capturar o valor do temporizador no início e no fim da função *insertLogElement*. Como o tamanho de um caractere na tabela ASCII é constante, então obteve-se um tempo de 0.086 ms para inserção de um elemento no buffer. Este tempo de inserção é útil para avaliar situações onde é necessário usar o log, mas têm-se severas restrições de tempo real no componente onde se deseja observar o comportamento.

Testes Funcionais

Requisito a ser validado:

/RF4/ O sistema deve possibilitar o usuário de armazenar os dados de freqüência cardíaca e saturação do oxigênio na memória RAM do micro-controlador.

Procedimento:

Para validar este requisito, deve-se:

1. Selecionar e habilitar o item "Store data" na lista de comandos;
2. Iniciar a aplicação para coletar os dados do sensor pressionando o botão "start".

Algoritmo de Teste:

```
static void testDataLog(void){
    selectItem();
    TEST_ASSERT_EQUAL_INT(TRUE, KeyUp());
    startApp();
}
```

Tempo Medido: 0.189 ms

Avaliação:

Para validar este requisito, foi necessário utilizar as funções do teclado de selecionar item, incremento, e início da aplicação. Quando a aplicação é iniciada, o primeiro item da lista de comandos é o tempo de amostragem dos dados. Deste modo, a função *selectItem* mostrada no código de teste acima é chamada com o propósito de selecionar o próximo item da lista de comandos que é "Store data". Neste item, o usuário pode habilitar ou desabilitar o armazenamento de dados na memória RAM do micro-controlador.

Para o propósito deste teste, o item "Store data" foi desabilitado. Sendo assim, o usuário deve pressionar o botão de incremento com o propósito de habilitar esta funcionalidade. Após habilitar o armazenamento de dados, o usuário pode iniciar o processo de captura de dados do sensor para ser mostrado no display do oxímetro. É importante salientar que este teste foi executado automaticamente usando o código de teste mostrado acima. Este caso de teste foi executado com sucesso e foi mensurado um tempo de 0.189 ms para a execução do código acima.

Requisito a ser validado:

/RF15/ O usuário deve ser capaz de ajustar a freqüência de amostragem dos dados do sensor que serão mostrados no display do micro-controlador.

Procedimento:

Para validar este requisito, deve-se:

1. Selecionar o item da lista de comando "Set Sample Time";
2. Incrementar o tempo de amostragem do sinal;
3. E depois decrementar até chegar no valor inicial que é 1 segundo.

Código de Teste:

```
static void testSETSAMPLETIME(void){
    int i;
    for(i=0; i<3; i++){
        selectItem();
    }
    TEST_ASSERT_EQUAL_INT(SETSAMPLETIME, selectItem());
    TEST_ASSERT_EQUAL_INT(2, KeyUp());
    TEST_ASSERT_EQUAL_INT(3, KeyUp());
    TEST_ASSERT_EQUAL_INT(4, KeyUp());
    TEST_ASSERT_EQUAL_INT(3, KeyDown());
    TEST_ASSERT_EQUAL_INT(2, KeyDown());
    TEST_ASSERT_EQUAL_INT(1, KeyDown());
    startApp();
}
```

Tempo Medido: 0.586 ms

Avaliação:

Para validar este requisito, simulou-se a situação onde o usuário pressiona diversas vezes o botão de selecionar item até encontrar o tempo de amostragem do sinal. Sendo assim, o código acima mostra que o usuário percorreu todos os itens da lista até encontrar o item "sample time". Depois disso, o usuário pressiona a tecla de incremento e decremento

3 vezes cada uma. Ou seja, ele ajusta o tempo de amostragem para 4 e depois decide voltar para o valor inicial que é 1 segundo. Finalmente, quando o usuário decide o valor do tempo de amostragem, ele inicia a coleta de dados do sensor pressionando a tecla *start*. Este caso de teste foi executado com sucesso de forma automática e foi mensurado um tempo de 0.586 ms para a execução do código acima.

Requisito a ser validado:

/RF24/ O log armazenado na memória RAM do micro-controlador deve ser enviado para o PC através da porta serial do micro-controlador.

Procedimento:

Para validar este requisito, deve-se:

1. Criar uma mensagem de tamanho fixo X;
2. Ajustar o tamanho do buffer para o valor X;
3. Inserir a mensagem no buffer circular;
4. E finalmente enviar a mensagem de texto para o PC.

Código de Teste:

```
static void testSendLog2PC(void){
    int i;
    size_t size;
    char message[100];
    strcpy(message,"SERIAL- >serial.c:init_serial(12), Error while writing in register");
    size = strlen(message);
    initLog(100);
    logd(message);
    TEST_ASSERT_EQUAL_INT(0, sendLog2PC());
}
```

Tempo Medido: 42.41 ms

Avaliação:

Para validar este requisito, criou-se uma mensagem de tamanho fixo e a mesma foi

armazenada no buffer circular através da função *logd*. Depois disso, a função *sendLog2PC* foi chamada com o propósito de enviar a mensagem de texto para o PC através da porta serial RS-232. Este caso de teste foi também executado automaticamente e passou com sucesso. Foi mensurado um tempo de 42.41 ms para a execução do código acima.

6.2 Protótipo do Soft-Starter Digital

Esta seção descreve a área de aplicação e características do protótipo *soft-starter* digital que foi desenvolvido com o intuito de validar a metodologia de desenvolvimento ágil proposta. O protótipo do *soft-starter* digital foi escolhido como estudo de caso devido ao fato de existir severas restrições impostas pelo hardware no desenvolvimento do software (p.e., tempo real e tamanho de código). Além disso, este protótipo foi desenvolvido por uma equipe de desenvolvimento com o intuito de validar as práticas de gerenciamento de projeto propostas na metodologia TXM. Esta seção descreve ainda a arquitetura do sistema e as práticas de teste que foram usadas para a construção do protótipo.

6.2.1 Características do Protótipo

O *soft-starter* digital é um equipamento que adota um método eficiente de partida do motor com baixo consumo de energia e ajuste de parâmetros adaptativos. Um *soft-starter* ótimo é aquele que inicia o motor com uma tensão mínima e corrente de partida reduzida [41]. Este equipamento é geralmente equipado com teclado e display e possibilita os usuário visualizar a tensão e corrente que está sendo aplicada nos terminais do motor de indução. Além disso, a maioria dos *soft-starter* digitais indicam falhas do sistema para o usuário e permitem que o log da aplicação seja salvo na memória de dados do dispositivo. A Figura 6.10 mostra um soft-starter digital comercial fornecido pela Baldor [6].

O *soft-starter* digital é basicamente composto pelos seguintes elementos:

- *Inversores:* Os inversores são usados para converter a corrente alterna (normalmente disponível) em corrente contínua e por fim em corrente alternada novamente (conversão CA-CC-CA).

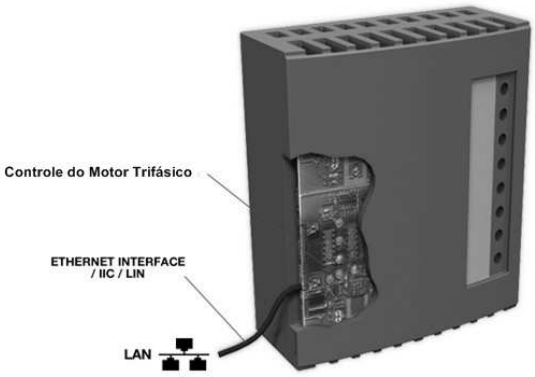

Figura 6.10: Soft-Starter Digital.

- *Retificador:* O retificador permite que uma tensão ou corrente alternada seja retificada, sendo transformada em contínua.

- *Filtro:* Em geral, a saída do retificador contém harmônicas elevadas da freqüência fundamental da fonte CA e estas são convenientemente eliminadas pelo uso de filtros.

Após o sinal da fonte de alimentação passar pelo retificador e filtro, o mesmo torna-se essencialmente CC. Deste modo, a função do inversor consiste em gerar uma nova fonte de tensão que, em geral, possui as propriedades de freqüência variável, tensão ajustável e, mesmo de fase ajustável. Os inversores geralmente usam transistores ou tiristores GTO (*Gate Turn Off*) como chaves, pois eles podem operar com freqüências de chaveamento muito mais alta do que aqueles que usam SCRs (Silicon Controlled Rectifiers) convencionais [2]. A Figura 6.10 mostra um inversor monofásico, em ponte com BJTs (Bipolar Junction Transistors) como chaves, usado para gerar uma tensão monofásica nos terminais de um motor de indução.

A freqüência de tensão na saída do inversor que se alterna é determinada pela taxa de variação do chaveamento. Se o período do chaveamento for de T segundos, a freqüência f será:

$$f = 1/T \tag{6.1}$$

Para que se obtenha uma tensão de saída senoidal, pode-se adotar basicamente dois métodos. O primeiro método consiste em empregar um circuito filtro no lado da saída do inversor [53]. Esse filtro deve ser capaz de deixar passar a grande potência de saída

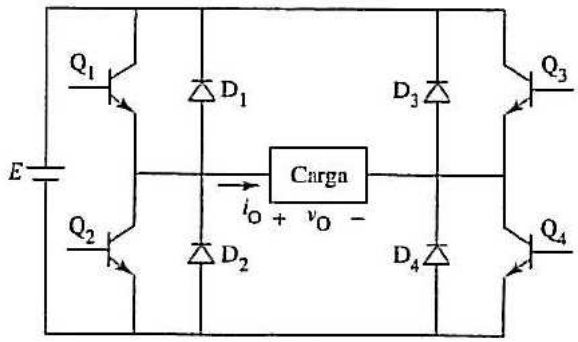

Figura 6.11: Inversor Monofásico [2].

do dispositivo, o que significa ter um tamanho adequado. Isto aumenta o custo e peso do inversor. Mais ainda, a eficiência ficará reduzida por causa das perdas adicionais da potência do filtro. O segundo método, utilizado neste experimento, modulação por largura de pulso (PWM - *Pulse Width Modulation*), usa um esquema de chaveamento no inversor para modificar a forma de onda de saída.

Em um inversor modulado por largura de pulso, a forma de onda da tensão de saída tem uma amplitude constante, cuja polaridade se inverte de maneira periódica, de modo a fornecer a freqüência fundamental na saída. A fonte de tensão é chaveada a intervalos regulares para fornecer uma tensão de saída variável. A tensão de saída do inversor é controlada pela variação da largura de pulso de cada ciclo da tensão de saída. Como mostrado na Figura 6.10, as chaves Q3 e Q4, do lado direito do inversor, passam para o estado ligado após um ângulo α (disparo dos transistores), em relação à passagem para o estado ligado das chaves Q1 e Q2 do lado esquerdo.

Em um inversor modulado por largura de pulso, a forma de onda da tensão de saída tem uma amplitude constante. Nesta situação, a fonte de tensão é chaveada em intervalos regulares para fornecer uma tensão de saída variável. A tensão de saída do inversor é controlada basicamente pela variação da largura de pulso de cada ciclo de tensão da saída. A Figura 6.12 mostra a tensão de saída do inversor monofásico mostrado na Figura 6.11.

Conforme pode ser observado na Figura 6.12, a tensão de saída resultante V_0 tem uma largura de pulso tw de α. A mudança do ângulo de deslocamento α pode alterar a tensão de saída do inversor [2]. A tabela 6.1 mostra a seqüência de chaveamento assim como a tensão resultante na saída do inversor.

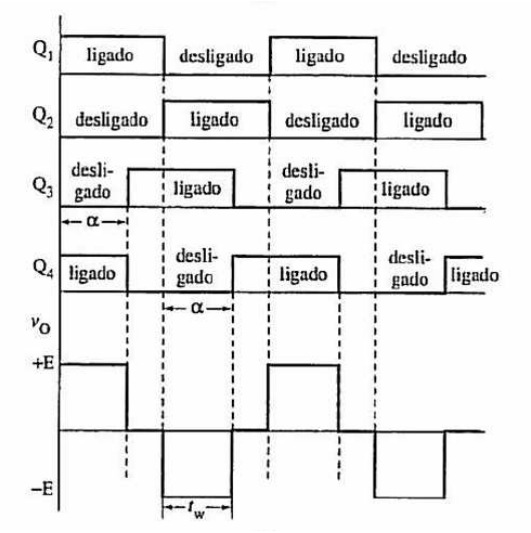

Figura 6.12: Tensão de saída do inversor monofásico [2].

Tabela 6.1: Seqüência de Chaveamento [2]

Q_1	Q_2	Q_3	Q_4	V_0
ligado	deligado	desligado	ligado	+E
ligado	deligado	ligado	desligado	0
desligado	ligado	ligado	desligado	−E
desligado	ligado	desligado	ligado	0
ligado	desligado	desligado	ligado	+E
ligado	desligado	ligado	desligado	0

Em linhas gerais, as principais características que foram implementadas para o *soft-starter* digital: *(i)* o sistema deve controlar automaticamente a partida do motor monofásico, *(ii)* o sistema deve ler o sinal de tensão fornecido pelo sensor através do conversor analógico-digital, *(iii)* o sinal PWM gerado nos pinos do micro-controlador deve atender os requisitos temporais da aplicação, *(iv)* uma interface homem-máquina (display e teclado) deve estar presente na solução final de modo que o usuário possa interagir com o sistema, *(v)* o *projeto* do sistema deve ser altamente otimizado para o ciclo de vida e eficiência , e *(vi)* o número de defeitos do sistema deve ser o menor possível. A Figura 6.13 apresenta uma visão geral do protótipo *soft-starter* digital.

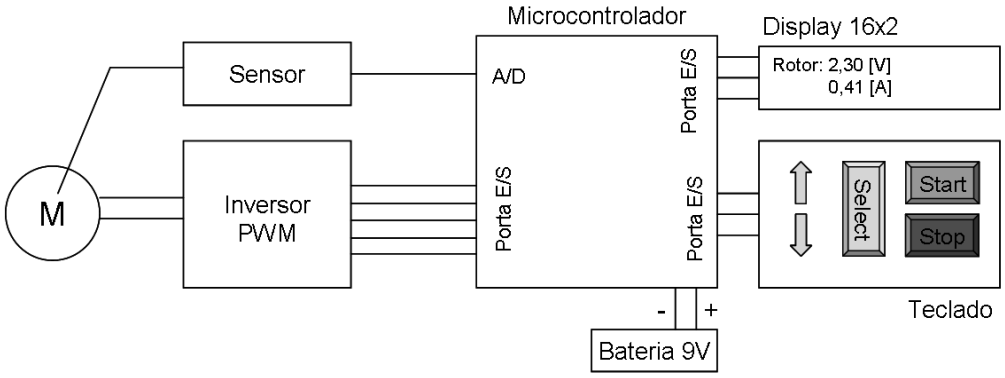

Figura 6.13: Visão Geral do *Soft-Starter* Digital.

6.2.2 Arquitetura do Sistema

A arquitetura do *soft-starter* digital consiste essencialmente de componentes de hardware e software conforme mostrado na Figura 6.14. Estes componentes de hardware e software estão conectados de tal forma para implementar um conjunto de funcionalidades enquanto satisfaz um conjunto de restrições (p.e., tempo de execução, uso de memória e consumo de energia). A arquitetura de software consiste dos drivers dos dispositivos (*serial, temporizador, teclado, display, conversor A/D*) e dos componentes *log do sistema, gerador PWM e conversor de unidades*. Além disso, existe uma camada de abstração que possibilita o software de aplicação utilizar os serviços fornecidos pela plataforma de desenvolvimento.

O componente do display fornece funções para realizar a escrita de dados do LCD 16x2 acoplado a plataforma de desenvolvimento. O componente do teclado fornece funções para detectar as teclas que foram pressionadas pelo o usuário. O componente da serial fornece meios de acessar o canal de comunicação entre plataforma de desenvolvimento e mundo externo através da porta RS-232. O componente do conversor A/D possibilita realizar leituras do nível de tensão dos terminais do motor de indução através de um sensor imaginário. O componente do temporizador fornece funções para disparar um serviço de interrupção da plataforma em tempos determinados.

É importante enfatizar que a comunicação entre o software de controle embarcado e o conversor A/D (PCF8591) é realizada através do protocolo de comunicação I^2C [47]. Este protocolo possui basicamente duas linhas de comunicação: o **SCL** que define o *clock* para sincronismo dos dados e **SDA** onde o dado é realmente transmitido. Para

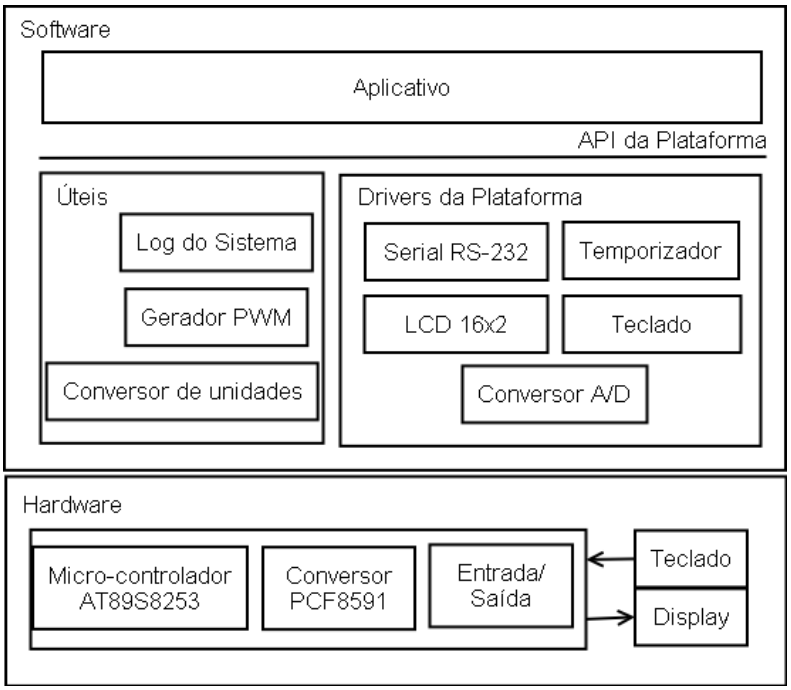

Figura 6.14: Arquitetura do *Soft-Starter* Digital.

o software embarcado do soft-starter digital, o micro-controlador (AT89S8253) funciona como mestre e o conversor A/D (PCF8591) como escravo. Ambos podem transferir dados, mas o controle é sempre do mestre. O mestre sempre inicia a conversa com uma seqüência de *start*. Além disso, o *start* e o *stop* são os únicos momentos onde a linha de dados muda quando a linha de clock está em nível alto. Um outro ponto importante a ser mencionado é que os dados são transferidos a 8 bits (MSB primeiro) e para cada 8 bits tem um bit de ACK (H-nok,L-ok).

O componente "Log do Sistema" foi desenvolvido com o propósito de depurar o código e armazenar conteúdo de variáveis na memória. O componente "gerador PWM" é responsável por gerar os pulsos a serem aplicados nos tiristores do circuito de controle do motor de indução. Estes dois componentes fazem uso de serviços fornecidos pelos drivers da plataforma. O componente "conversor de unidade" fornece funções para de leitura do conversor D/A onde o intervalo de valores do sensor imaginário é abstraído para a camada de aplicação.

A arquitetura de hardware possui um conjunto de componentes interconectados que acelera o processo de desenvolvimento do sistema. A arquitetura da plataforma possui conversor serial RS-232, micro-controlador AT89S8253, relógio de tempo real PCF8583,

conversor de 4 canais A/D e 1 D/A com resoluções de 8 bits e 4 portas de expansão (32 portas de E/S). Além disso, esta plataforma possui 12 KB de memória flash (memória de programa) e 32KB de memória RAM (memória de dados). As próximas seções descrevem brevemente os componentes de software do *soft-starter* digital. É importante salientar que os componentes em comum com o protótipo do oxímetro de pulso não serão discutidos nesta seção com o intuito de minimizar redundância na dissertação de mestrado.

A Seção C.2 do apêndice desta dissertação descreve as funções públicas dos módulos dos drivers (componentes) e o software de aplicação desenvolvido para o projeto do *soft-starter* digital. A próxima subseção descreve as técnicas de teste que foram utilizadas para verificar a corretude lógica e temporal do soft-starter digital.

6.2.3 Testes Unitários e Funcionais

Esta seção descreve as técnicas de teste que foram utilizadas para verificar a corretude lógica e temporal do soft-starter digital. Adotando a mesma estratégia do projeto do oxímetro de pulso, apenas alguns testes serão explorados nesta seção. A seleção dos casos de teste foi realizada baseada na importância da função para o sistema, na complexidade da função e no grau de adaptação das técnicas de teste utilizadas para verificação da função.

Testes Unitários

Módulo Gerador PWM

externalInterrupt: Esta função muda a flag para nível lógico alto e lê o valor de tensão a partir do conversor A/D.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se:

1. Ajustar o conversor A/D para um valor de leitura conhecido;
2. Simular a chamada da função *externalInterrupt*;
3. Comparar o valor real do flag e do conversor A/D com o valor esperado.

Código de Teste:

```
static void testExternalInterrupt(void) {
    ADValue = 100;
    externalInterrupt();
    TEST_ASSERT_EQUAL_INT(1, signal);
    TEST_ASSERT_EQUAL_INT(100, ADValue);
}
```

Tempo Medido: 223.51 μs

Avaliação:

Para testar a corretude lógica desta função, foi necessário declarar as variáveis *signal* e *ADValue* como global. Além disso, nós tivemos que simular a chamada da função *externalInterrupt* conforme mostrado no código acima. Sendo assim, o valor das variáveis *signal* e *ADValue* puderam ser analisadas com uso do comando *assert*. Esta função passou com sucesso nos casos de teste. Para verificar a corretude temporal, nós tivemos que executar este caso de teste na plataforma de desenvolvimento. Sendo assim, nós obtivemos um tempo de execução de aproximadamente 223.51 μs para a função *externalInterrupt()*.

timerTick: Esta função configura o temporizador para ser disparado a cada 100μs e incrementa o valor dos contadores dos pinos Q_1 e Q_3 com o intuito de controlar a geração dos sinais PWM.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se:

1. Chamar a função *timerTick* um determinado número de vezes;
2. Comparar o valor real a ser carregado no temporizador com o valor esperado;
3. Comparar os contadores dos pinos Q_1 e Q_3 com o valor esperado;
4. Atualizar os contadores depois de executar os procedimentos 1, 2 e 3.

Código de Teste:

```
static void testTimerTick(void) {
    int i, rate = 1, counter01 = 100, counter02 = 0;
    q3Enable = 0;
    counterQ1 = 0; counterQ3 = 0;
```

```
    P2_1 = 0;
    for (i = 0; i < TEST_NUMBER; i++) {
        timerTick();
        TEST_ASSERT_EQUAL_INT(TLOW, TL0);
        TEST_ASSERT_EQUAL_INT(THIGH, TH0);
        TEST_ASSERT_EQUAL_INT(counter01, counterQ1);
        TEST_ASSERT_EQUAL_INT(counter02, counterQ3);
        TEST_ASSERT_EQUAL_INT(rate, P2_1);
        TEST_ASSERT_EQUAL_INT(1, TR0);
        if (i > 40)
            q3Enable = 100;
        rate = !rate;
        counter01 + = TICK;
        counter02 + = q3Enable;
    }
}
```

Tempo Medido: 74.865 μs

Avaliação:

Para testar a corretude lógica desta função, foi necessário declarar as variáveis *counterQ1*, *counterQ3* e *q3Enable* como global. Além disso, nós tivemos que simular a chamada da função *timerTick* 100 vezes conforme mostrado no código acima. Sendo assim, o valor das variáveis *counterQ1* e *counterQ3* puderam ser analisadas com uso do comando *assert*. Um outro ponto importante é que utilizamos o pino P2_1 para efeito de debug da função *timerTick*. Este pino possibilitou monitorar com o uso de um osciloscópio a freqüência das chamadas da função.

Nós simulamos também a situação onde o sinal do pino Q_3 fosse disparado depois de um ângulo α em relação a Q_1. Deste modo, depois de habilitar o disparo do pino Q_3, a variável de teste *counter02* é incrementada e depois comparada com o valor da variável *counterQ3* que é modificada dentro da função *timerTick()*. Todos os casos de teste foram executados com sucesso. Para verificar a corretude temporal, nós tivemos que executar estes casos de teste na plataforma de desenvolvimento. Sendo assim, nós obtivemos um

tempo de execução de aproximadamente 74.865 μs para a função *timerTick()*.

generateSignal: Esta função usa os valores do contador para comparar com os respectivos períodos e gerar os sinais dos pinos de controle Q_1, Q_2, Q_3 e Q_4.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se:

1. Preencher o array de inteiros *signalData01* com todas as combinações possíveis para os pinos de controle Q_1, Q_2, Q_3 e Q_4;

2. Inicializar os contadores dos pinos Q_1 e Q_3 assim como o valor do ângulo α de disparo;

3. Chamar a função *generateSignal* um determinado número de vezes com o intuito de simular a geração do sinal PWM;

4. Comparar o valor real dos pinos Q_1, Q_2, Q_3 e Q_4 com o valor esperado no *array signalData01*.

Código de Teste:

```
static void setUp(void) {
    fillData("signal01.txt", signalData01);
}

static void testGenerateSignalNormal(void) {
    int i;
    Q1 = 0; Q2 = 1; Q3 = 0; Q4 = 1;
    ton = 5;
    q3Enable = 0;
    counterQ1 = 0;
    counterQ3 = 0;
    MAX_T_2 = 8333;
    for (i = 0; i < SIGNAL_NUMBER; i++) {
        generateSignal();
        counterQ1 += TICK;
        counterQ3 += q3Enable;
```

$TEST\_ASSERT\_EQUAL\_INT$(signalData01[$4 * i$], Q1);
$TEST\_ASSERT\_EQUAL\_INT$(signalData01[$4 * i + 1$], Q2);
$TEST\_ASSERT\_EQUAL\_INT$(signalData01[4 * i + 2], Q3);
$TEST\_ASSERT\_EQUAL\_INT$(signalData01[$4 * i + 3$], Q4);
}
}

Tempo Medidos:

Mínimo: 74.865 μs Máximo: 107.415 μs Média: 78.554 μs

Avaliação:

Para testar a corretude lógica desta função, foi necessário implementar a função *setUp* com o propósito de alimentar o *array signalData01* com possíveis valores dos pinos Q_1, Q_2, Q_3 e Q_4. Além disso, nós tivemos também que declarar as variáveis *counterQ1*, *counterQ3* e *q3Enable* como global e inicializá-las com um determinado valor no início da execução dos casos de teste. É importante observar que a freqüência do sinal PWM gerado é de 60 Hz. Deste modo, inicializamos a variável MAX_T_2 com o valor máximo que é metade do período do sinal. Este valor indica a condição de contorno da geração PWM e também foi exercitado em outros casos de teste.

Nós tivemos também que simular a chamada da função *generateSignal* 100 vezes conforme mostrado no código acima. Sendo assim, o valor das variáveis *counterQ1* e *counterQ3* puderam ser alteradas pela função *generateSignal* com o intuito de simular a geração do sinal. Deste modo, a função *generateSignal* muda os valores dos pinos Q_1, Q_2, Q_3 e Q_4 de acordo com os valores dos contadores, permitindo assim analisá-las com o uso do comando *assert*. Esta análise é feita comparando o valor real do pino com o valor esperado no *array signalData01*.

Todos os casos de teste foram executados com sucesso. Para verificar a corretude temporal, nós tivemos que executar estes casos de teste na plataforma de desenvolvimento. Sendo assim, nós obtivemos um tempo de execução para o pior caso de aproximadamente 107.415 μs para a função *generateSignal()*.

configureIteration: Esta função tem como objetivo configurar a geração de cada sinal aplicado nos pinos Q_1, Q_2, Q_3 e Q_4 do circuito de controle do motor.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se:

1. Chamar a função *configureIteration* um determinado número de vezes com o intuito de configurar as variáveis de controle para geração do sinal PWM;

2. Verificar o valor do ângulo de disparo α para a próxima iteração da geração de sinal PWM;

3. Verificar se o valor de tensão RMS já está na condição de contorno. Caso esteja, verificar a *flag* que indica a condição de contorno para geração dos sinais.

4. Comparar o valor real das variáveis globais de controle para geração do sinal PWM (*counterQ1, counterQ3, q3Enable, signal e alert*) com o valor esperado.

Código de Teste:

```
static void testConfigureIteration(void) {
    int i;
    increasing = 1;
    for (i = 0; i < TEST_NUMBER; i++) {
        ton = i;
        configureIteration();
        TEST_ASSERT_EQUAL_INT(((i + 1) >= NUM_VRMS) ?
                    NUM_VRMS − 1: i + 1, ton);
        if ((i + 1) >= NUM_VRMS) {
            TEST_ASSERT_EQUAL_INT( 0, increasing);
        } else {
            TEST_ASSERT_EQUAL_INT( 1, increasing);
        }
        TEST_ASSERT_EQUAL_INT(0, counterQ1);
        TEST_ASSERT_EQUAL_INT(0, counterQ3);
        TEST_ASSERT_EQUAL_INT(0, q3Enable);
        TEST_ASSERT_EQUAL_INT(0, signal);
        TEST_ASSERT_EQUAL_INT(0, alert);
        TEST_ASSERT_EQUAL_INT(1, EX1);
```

```
        TEST_ASSERT_EQUAL_INT(1, IT1);
        TEST_ASSERT_EQUAL_INT(0, IE1);
        TEST_ASSERT_EQUAL_INT(1, EA);
        TEST_ASSERT_EQUAL_INT(1, ET0);
        TEST_ASSERT_EQUAL_INT(1, ET0);
        TEST_ASSERT_EQUAL_INT(1, TR0);
    }
}
```

Tempo Medido: 43.787 ms

Avaliação:

Para testar a corretude lógica desta função, nós tivemos que declarar as variáveis *counterQ1*, *counterQ3*, *q3Enable*, *ton*, *signal* e *alert* como globais para possibilitar a análise dos caminhos do código. Além disso, a função *configureIteration* foi chamada aproximadamente 100 vezes com o intuito de configurar as variáveis de controle e com isso exercitar os caminhos de código da função. Em cada iteração do loop *for* mostrado no código acima, foi verificado se o valor da tensão RMS estava na condição de contorno. Caso estivesse, era verificado se a *flag increasing*, que indica a condição de contorno, estava habilitada.

É importante observar que os valores inteiros carregados nos registradores do 8051 são analisados em cada iteração do loop *for* pelo uso do comando $TEST\_ASSERT\_EQUAL\_INT$. Depois de calcular o valor do ângulo de disparo, a função *configureIteration* inicializa as variáveis *counterQ1*, *counterQ3*, *q3Enable*, *signal* e *alert* com zero. Sendo assim, o código de teste acima também verifica se estas variáveis são inicializadas apropriadamente. Todos os casos de teste foram executados com sucesso. Para verificar a corretude temporal, nós tivemos que executar estes casos de teste na plataforma de desenvolvimento. Sendo assim, nós obtivemos um tempo de execução para o pior caso de aproximadamente 43.787 ms para a função *configureIteration*.

6.3 Protótipo do Motor de Indução Monofásico

Esta seção descreve a área de aplicação e características do protótipo motor de indução monofásico que foi desenvolvido com o intuito de validar a metodologia de desenvolvimento ágil proposta. O protótipo do simulador do motor de indução monofásico foi escolhido como estudo de caso devido ao fato de existir severas restrições impostas pelo hardware no desenvolvimento do software (p.e., tempo real e tamanho de código). Além disso, este protótipo foi desenvolvido por uma equipes de desenvolvimento com o intuito de validar as práticas de gerenciamento de projeto propostas na metodologia TXM. Esta seção descreve ainda a arquitetura do sistema e as práticas de teste que foram usadas para a construção do protótipo.

6.3.1 Características do Protótipo

A grande maioria dos motores de indução monofásicos é construída numa faixa de fração de HP (*Horsepower*). Motores monofásicos podem ser encontrados em inúmeras aplicações desempenhando todo tipo de função em residências, lojas, escritórios e fazenda. Vários tipos diferentes de motores monofásicos foram desenvolvidos principalmente por duas razões [53]. A primeira, as necessidades de torque dos instrumentos e aplicações nas quais eles são empregados variam amplamente. A segunda, é desejável usar o motor de menor preço que acionará uma dada carga satisfatoriamente. A Figura 6.15 mostra um motor de indução em uma aplicação industrial.

Figura 6.15: Motor de Indução.

Uma característica importante que distingue os motores de indução é que eles são máquinas com *excitação única*. Embora tais máquinas sejam equipadas tanto com um enrolamento de campo como um enrolamento de armadura, em condições normais de utilização a fonte de energia é conectada a um único enrolamento, o enrolamento de

campo. As correntes circulam no enrolamento de armadura por indução, o que cria uma distribuição ampère-condutor que interage com a distribuição de campo para produzir um torque líquido unidirecional. A grande vantagem do motor de indução é a sua capacidade de operar sem necessidade de contato com os enrolamentos do rotor. Além disso, o motor de indução produz torque a qualquer velocidade abaixo da velocidade síncrona.

Devido a sua geometria e modo de operação, pode-se considerar o circuito equivalente do motor de indução idêntico ao circuito do transformador [53]. A diferença do circuito equivalente do motor de indução e do transformador consiste essencialmente no módulo dos parâmetros devido ao entreferro presente no motor de indução em vez de um núcleo de ferro. A Figura 6.16 mostra o circuito equivalente do motor de indução monofásico baseado em componentes simétricos (este circuito equivalente é deduzido na referência [53]).

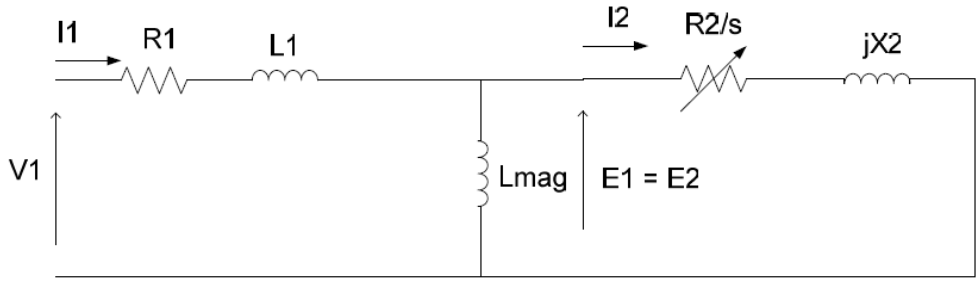

Figura 6.16: Circuito equivalente associados com os campos de seqüência positiva e negativa.

É importante salientar que o motor monofásico necessita de um dispositivo auxiliar de partida, a fim de que o mesmo possa ser utilizado. Os dispositivos de auxílio atuam basicamente no sentido de criar um desequilíbrio no campo do estator. Uma vez que o motor começa a girar observa-se que o torque fornecido pelo motor no sentido de rotação é maior que o torque exercido no sentido contrário, ou seja, o motor passa a fornecer um torque acelerante. A forma mais usual de partida é o emprego de um enrolamento auxiliar, o qual pode atuar apenas na partida ou ainda ser conectado para funcionamento permanente. Um outro ponto importante é que os motores monofásicos são em geral maiores e possuem rendimentos menores que motores trifásicos de mesma potência [53].

Em linhas gerais, as principais características que foram implementadas para o simulador do motor de indução monofásico incluem: *(i)* o sistema deve simular o compor-

tamento do motor monofásico, *(ii)* o sistema deve reproduzir o sinal PWM fornecido (pelo *soft-starter* digital) através das portas de E/S do micro-controlador, *(iii)* o sistema deve calcular e mostrar no display o valor de tensão e corrente baseado no sinal PWM fornecido pelo *soft-starter*, *(iv)* uma interface homem-máquina (display e teclado) deve estar presente na solução final de modo que o usuário possa interagir com o sistema, *(v)* o *projeto* do sistema deve ser altamente otimizado para o ciclo de vida e eficiência, e *(vi)* o número de defeitos do sistema deve ser o menor possível. A Figura 6.17 apresenta uma visão geral do protótipo simulador de motor de indução monofásico.

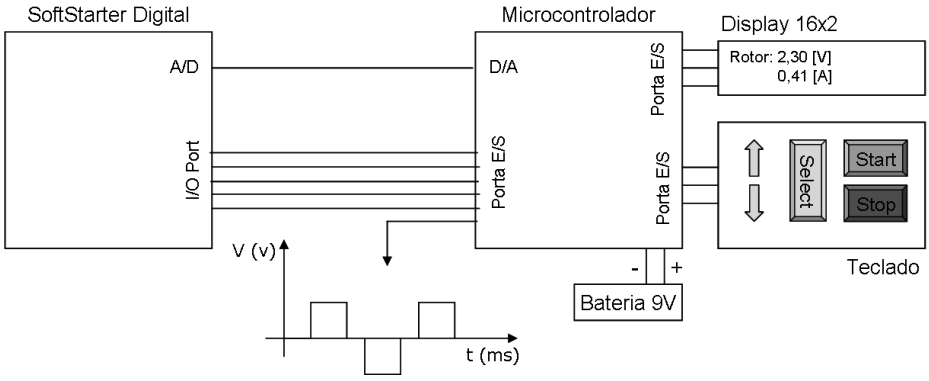

Figura 6.17: Visão Geral do Experimento 2.

6.3.2 Arquitetura do Sistema

A arquitetura do simulador do motor de indução monofásico consiste essencialmente de componentes de hardware e software conforme mostrado na Figura 6.18. Assim como nos protótipos do oxímetro de pulso e *soft-starter* digital, estes componentes de hardware e software estão conectados de tal forma para implementar um conjunto de funcionalidades enquanto satisfazem um conjunto de restrições (p.e., tempo de execução, uso de memória e consumo de energia). A arquitetura de software consiste basicamente dos drivers dos dispositivos (*serial, temporizador, teclado, display, conversor D/A*) e dos componentes *log do sistema, tratador PWM e simulador do motor*. Além disso, existe uma camada de abstração que possibilita o software de aplicação utilizar os serviços fornecidos pela plataforma de desenvolvimento.

Os drivers dos dispositivos da serial, temporizador, teclado e display são os mesmos do protótipo do *soft-starter* digital assim como o componente de log do sistema. A

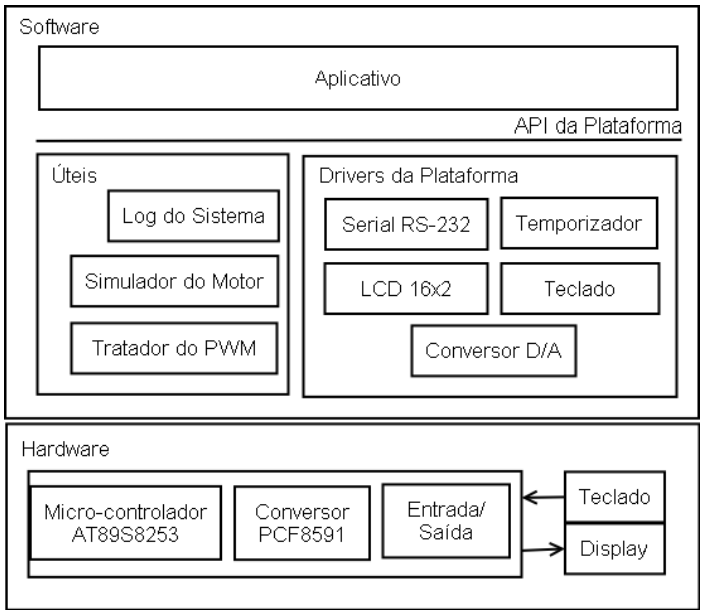

Figura 6.18: Soft-Starter Digital.

diferença desta arquitetura quando comparada com o *soft-starter* digital está nos componentes de software que são responsáveis pela simulação do motor, tratamento do sinal PWM e na conversão do sinal digital para analógico D/A. O componente simulador do motor é responsável por implementar o modelo do motor de indução monofásico através de funções matemáticas. Os parâmetros do motor (p.e., resistência, reatância, escorregamento) podem ser facilmente alterados através da camada de aplicação chamando as funções disponíveis da nossa API proprietária. Esta API da plataforma representa uma abstração única dos drivers dos dispositivos e de outros componentes de software do sistema.

O componente tratador do sinal PWM é responsável por receber os sinais gerados pelo *soft-starter* digital e calcular o valor da tensão RMS que está sendo aplicada nos terminais do motor de indução monofásico. Além disso, após calculado o valor de tensão representado no sinal PWM, este módulo informa o *soft-starter* digital que um novo sinal PWM pode ser enviado para o simulador do motor. O driver do conversor D/A fornece funções para representar o estado do sensor. Este sensor é imaginário e estaria conectado nos terminais do motor, ou seja, no enrolamento de campo. Sendo assim, o tratador do sinal PWM calcula o valor de tensão RMS, aplica este valor no modelo matemático do motor e calcula a corrente e tensão induzida no enrolamento de armadura do motor. Estes

valores de tensão e corrente são fornecidos para os usuários do simulador do motor.

O valor de tensão no enrolamento de campo é fornecido para o *soft-starter* digital através de um canal do conversor D/A representado o sensor imaginário. Assim como no *soft-starter* digital, o conversor D/A é acessado através do barramento de comunicação I^2C. É importante salientar que os componentes em comum com os protótipos do oxímetro de pulso e *soft-starter* digital não serão discutidos nesta seção com o intuito minimizar redundância neste texto.

A Seção C.3 do apêndice desta dissertação descreve as funções públicas dos módulos dos drivers (componentes) e o software de aplicação desenvolvido para o projeto do motor de indução monofásico. A próxima subseção descreve as técnicas de teste que foram utilizadas para verificar a corretude lógica e temporal deste protótipo.

6.3.3 Testes Unitários e Funcionais

Esta seção descreve as técnicas de teste que foram utilizadas para verificar a corretude lógica e temporal do simulador do motor de indução monofásico. Adotando a mesma estratégia do projeto do oxímetro de pulso e soft-starter digital, apenas alguns testes serão explorados nesta seção. A seleção dos casos de teste foi realizada baseada na importância da função para o sistema, na complexidade da função e no grau de adaptação das técnicas de teste utilizadas para verificação da função.

Testes Unitários

handleINT1: Esta função configura o temporizador 0 para disparar a cada 65 ms com o intuito de mensurar os períodos T_{on} e T do sinal PWM.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se:

1. Preencher a matriz de inteiros *signals* com todas as combinações possíveis para os pinos de controle Q_1, Q_2, Q_3 e Q_4;

2. Inicializar o array *times* e o valor a ser carregado nos registradores;

3. Chamar a função *handleINT1* um determinado número de vezes com o intuito de simular a interrupção gerada pelo *soft-starter* digital;

4. Comparar o valor real contido no *array times* com o valor esperado.

Código de Teste:

```
static void TestHandlerINT1(void) {
    int i;
    times[0] = 0;
    times[1] = 0;
    TH0 = 1;
    TL0 = 0;
    TR0 = 0;
    for(i=0; i < LINES; i++) {
        Q1 = signals[i][0];
        Q2 = signals[i][1];
        Q3 = signals[i][2];
        Q4 = signals[i][3];
        Q5 = signals[i][4];
        handleINT1();
        if ((times[0] != 0) && (times[1] != 0)) {
            TEST_ASSERT_EQUAL_INT(256, times[0]);
            TEST_ASSERT_EQUAL_INT(256, times[1]);
            times[0]=times[1]=0;
            TR0=0;
        }
    }
}
```

Tempos Medidos:

Mínimo: 13.02 μs Máximo: 335.265 μs Média: 141.592 μs

Avaliação:

Para testar a corretude lógica desta função, foi necessário implementar a função *setUp* com o propósito de alimentar o *array signals* com possíveis valores dos pinos Q_1, Q_2, Q_3 e Q_4. Além disso, nós tivemos também que declarar o *array times* como global e inicializá-

lo com um determinado valor no início da execução dos casos de teste. É importante observar que foi necessário chamar a função *handleINT1* um determinado número de vezes com o intuito de simular a interrupção produzida pelo soft-starter digital. Antes de simular a interrupação, foi também necessário atualizar o estado dos pinos de controle Q_1, Q_2, Q_3 e Q_4 com o valor contido no *array signals*.

É importante mencionar que foram criados casos de teste para a condição de contorno do sistema, ou seja, simulamos a geração de um sinal onde T_{on} está bem próximo de T. Como mostrado no código de teste acima, para cada chamada da função *handleINT1*, era verificado o valor contido no *array times*. Caso o valor contido nestas variáveis fossem diferente a zero então a função *assert* comparava o valor real com o esperado.

Para verificar a corretude lógica, nós executamos os testes no PC desktop e comparamos o valor real com 255, que é o valor de carga do temporizador. Este teste serviu para exercitar os caminhos do código da função. Todos os casos de teste foram executados com sucesso. Para verificar a corretude temporal, nós tivemos que executar estes casos de teste na plataforma de desenvolvimento. Sendo assim, nós obtivemos um tempo de execução para o pior caso de aproximadamente 335.265 μs para a função *handleINT1*.

showVRMS: Esta função imprime o valor da tensão RMS no display, escreve este valor no conversor D/A e finalmente envia um sinal para o soft-starter digital.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se chamar a função *showVRMS* com diferentes valores de tensão RMS.

Código de Teste:

```
static void TestShowVRMS(void) {
    for(i=50; i < 127; i++) {
        TEST_ASSERT_EQUAL_INT(1, showVRMS(i));
    }
}
```

Tempo Medido: 265.3 μs

Avaliação:

Para verificar a corretude lógica, nós tivemos que alterar a função *showVRMS* para retornar um valor conhecido em caso de sucesso na execução. Sendo assim, nós chamamos esta função passando como parâmetro todos os valores possíveis de tensão RMS. Como mostrado no código acima, o retorno da função *showVRMS* é comparado com 1 através da função $TEST\_ASSERT\_EQUAL\_INT$. Todos os casos de teste foram executados com sucesso. Para verificar a corretude temporal, nós tivemos que executar estes casos de teste na plataforma de desenvolvimento. Sendo assim, nós obtivemos um tempo de execução para o pior caso de aproximadamente 265.3 μs para a função *handleINT1*.

write2DA: Esta função escreve o valor de tensão que varia de 50 a 127 volts no canal do conversor D/A.

Procedimento de Teste:

Para verificar a corretude lógica e temporal desta função, deve-se:

1. Ajustar os valores de tensão entre 50 e 127 volts para a função *write2DA*;

2. Ajustar valores de tensão que estejam fora do intervalo de operação do sistema;

3. Comparar o valor retornado pela função *write2DA* com o valor esperado. Se a função for executada com sucesso então o inteiro 1 é retornado. Caso contrário, -1 é retornado.

Código de Teste:

```
static void testWrite2DA(void) {
    unsigned char temp = 0;
    for(i=50; i<=127; i++) {
        temp = write2DA(i);
        TEST_ASSERT_EQUAL_INT(1, temp);
    }
    temp = write2DA(20);
    TEST_ASSERT_EQUAL_INT(-1, temp);
    temp = write2DA(200);
    TEST_ASSERT_EQUAL_INT(-1, temp);
}
```

Tempos Medidos:

Mínimo: 81.981 μs Máximo: 86.71 μs Média: 84.789 μs

Avaliação:

Para verificar a corretude lógica, nós tivemos que exercitar todos os caminhos de execução do código da função. Sendo assim, a função *write2DA* foi chamada passando como parâmetro todos os valores que estão no intervalo de 50 a 127 volts. Este intervalo define a tensão de operação do motor de indução monofásico. Além disso, nós tivemos que simular cenários de falha, ou seja, valores de tensão que estão fora do intervalo de operação do motor.

Deste modo, nós usamos valores acima e abaixo do intervalo de tensão do motor. Todos os casos de teste foram executados com sucesso. Para verificar a corretude temporal, nós tivemos que executar estes casos de teste na plataforma de desenvolvimento. Sendo assim, nós obtivemos um tempo de execução para o pior caso de aproximadamente 86.71 μs para a função *write2DA*.

6.4 Resumo

Esta seção mostrou em linhas gerais que o oxímetro de pulso é um equipamento responsável por mensurar a saturação de oxigênio no sistema sanguíneo do paciente usando um método não-invasivo. Foi apresentando também as principais funcionalidades implementadas para o oxímetro de pulso assim como a arquitetura de hardware e software. Um outro protótipo apresentado foi o *soft-starter* digital que é um equipamento que adota um método eficiente de partida do motor com baixo consumo de energia e ajuste de parâmetros adaptativos. Para o *soft-starter* digital foi apresentado apenas os componentes de hardware e software que diferenciavam do projeto do oxímetro de pulso. A mesma estratégia foi adotada para o simulador do motor de indução que é encontrado em inúmeras aplicações desempenhando todo tipo de função em residências, lojas, escritórios e fazendas.

Para os três protótipos desenvolvidos nesta dissertação, utilizou-se o paradigma de projeto baseado em plataforma. Caracterizou-se a plataforma por componentes programáveis. Sendo assim, cada instância da plataforma proveniente a partir da arquitetura da plataforma manteve suficiente flexibilidade para suportar o espaço de projeto da

aplicação. Além disso, considerou-se que o software de aplicação dos três protótipos se comunica com os drivers dos dispositivos através da API proprietária da plataforma. Esta API representa uma abstração única da arquitetura da plataforma. Com esta API assim definida, o software de aplicação pode reusar esta API para cada instância da plataforma (maximização de reuso).

Além disso, esta seção descreveu os casos de teste mais relevantes para os projetos do oxímetro de pulso, *soft-starter* digital e simulador do motor de indução. A seleção dos casos de teste foi realizada baseada na importância da função para o sistema, na complexidade da função e no grau de adaptação das técnicas de teste utilizadas para verificação da corretude lógica e temporal da função assim como validação dos requisitos. A Tabela 6.2 mostra as funções que foram descritas neste capítulo assim como o tempo de execução do pior caso mensurado.

Tabela 6.2: Funções e Tempo de Execução (ms)

Projeto	Módulo	Função	Tempo de Execução
Oxímetro	Sensor	*getHR*	0.256
Oxímetro	Sensor	*IsOutofTrack*	0.268
Oxímetro	Teclado	*checkPressedButton*	0.064
Oxímetro	Temporizador	*initTimer0s*	1000.0794
Oxímetro	Log	*logd*	0.086
Soft-Starter	Gerador PWM	*externalInterrupt*	0.2235
Soft-Starter	Gerador PWM	*timerTick*	0.074865
Soft-Starter	Gerador PWM	*generateSignal*	0.1074
Soft-Starter	Gerador PWM	*configureIteration*	0.04378
Simulador	Tratador PWM	*handleINT1*	0.33526
Simulador	Tratador PWM	*showVRMS*	0.2653
Simulador	write2DA	*showVRMS*	0.8671

Foi mostrado que para executar os casos de teste em uma maneira automatizada utilizando o *framework* de teste unitário [50], nós tivemos que implementar um mecanismo para executar o software embarcado tanto no PC desktop quanto na plataforma alvo. Além disso, nós dividimos os componentes de software para teste em duas categorias: código independente de hardware e código específico de hardware. Para o código puro, nós usamos as técnicas convencionais de teste em engenharia de software descritas em [48].

Para testar o código específico de hardware, nós tivemos que dividi-lo em duas subcat-

egorias (código que controla o hardware e código guiado pelo ambiente) e propor algumas adaptações na maneira convencional de testar o software. Para aqueles componentes de software que tocavam no hardware, nós tivemos que executar os casos de teste manualmente na plataforma alvo. Para componentes que são guiados pelo ambiente, a estratégia adotada foi substituir dados coletados a partir do sensor em tempo real por dados contidos em arquivo de acesso seqüencial. Para todos os casos de teste, foi realizada uma avaliação da corretude lógica e temporal.

Capítulo 7

Resultados Experimentais

Este capítulo tem como objetivo apresentar os resultados da aplicação da metodologia proposta no desenvolvimento dos protótipos oxímetro de pulso, *soft-starter* digital e simulador do motor de indução. Com este objetivo em mente, esta seção apresenta os valores das métricas de projeto mensuradas durante o desenvolvimento dos protótipos. Estas métricas incluem esforço estimado e mensurado do desenvolvimento, valores de negócio, velocidade do *sprint*, complexidade ciclomática, número de linhas de código da aplicação e teste, uso de memória e consumo de energia do sistema. Os resultados obtidos com o desenvolvimento dos protótipos são apresentados e discutidos os pontos fortes e fracos da metodologia proposta.

7.1 Oxímetro de Pulso

Esta seção apresenta os resultados da metodologia proposta aplicada ao desenvolvimento do oxímetro de pulso. O projeto foi dividido em 3 diferentes sprints e desenvolvido por um engenheiro de sistema embarcado. O proprietário da plataforma, que também foi envolvido no desenvolvimento, estava responsável pela definição da qualidade, cronograma, custo e requisitos do produto. No início do projeto, nós criamos uma lista de novas funcionalidades e requisitos com o intuito de unir todas as nossas necessidades com relação ao desenvolvimento do produto. Estas informações foram incluídas no backlog de produto conforme descrito na Seção 4.5.1.

Baseado nos valores de negócio dos requisitos, nós escolhemos a partir do backlog

de produto um conjunto de funcionalidades para serem implementadas nos *sprints* do projeto. O primeiro e segundo *sprint* levou aproximadamente um mês cada um e o terceiro *sprint* levou duas semanas para ser concluído. Como uma estratégia de gerenciamento de requisito, nós colocamos mais ênfase para entregar funcionalidades do sistema com alto valor de negócio agregado no início dos sprints. A Tabela 7.1 mostra o esforço mensurado e estimado assim como o valor de negócio e velocidade do desenvolvimento para cada *sprint*.

Tabela 7.1: Esforço estimado e mensurado (em horas)

	Sprint 1	Sprint 2	Sprint 3
Esforço estimado	110	122	60
Esforço medido	128	127	63
Valor de negócio	22	27	16
Velocidade do *sprint*	0.171	0.212	0.253

A Figura 7.1 mostra o gráfico *burndown* do primeiro *sprint* do projeto. Como pode ser visto nesta figura, o trabalho inicial foi estimado em 110 horas no início do *sprint*. Nos dias 14 e 15, as horas restantes estimadas aumentaram, pois novas tarefas foram descobertas e como resultado o esforço restante estimado foi também revisado. Esta situação ocorreu devido ao fato de que nós ainda estávamos aprendendo a tecnologia envolvida, o ambiente de desenvolvimento, e o domínio da aplicação. Para os outros dois *sprints*, nós melhoramos nossa estimativa e a velocidade de desenvolvimento, mas mesmo assim ainda não alcançamos um gráfico linear do backlog de *sprint*.

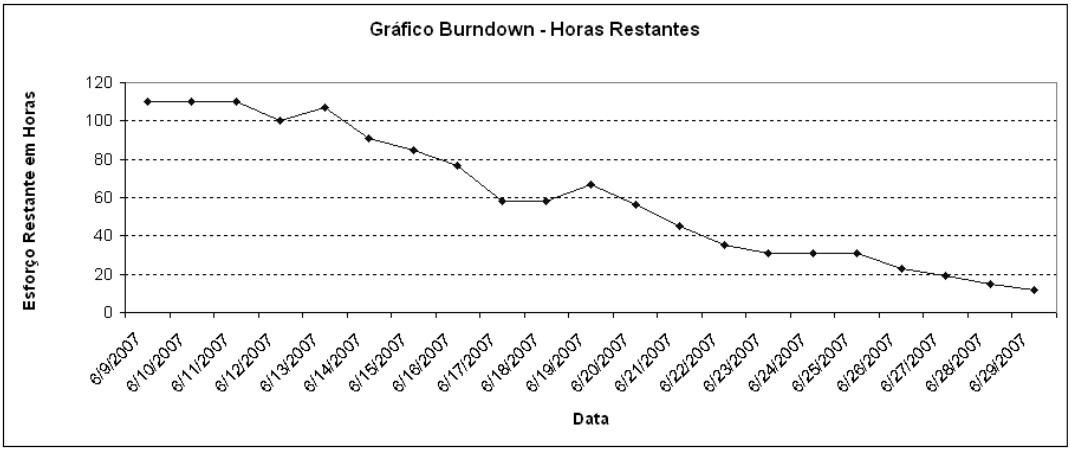

Figura 7.1: Gráfico Burndown do Sprint 1.

Como mencionado na Seção 6.1.3, nós tivemos código independente de hardware e código específico de hardware para o projeto do oxímetro de pulso. Para o código puro, foi aplicada as técnicas de teste propostas no *processo para implementar novas funcionalidades* que tem como objetivo checar não somente a lógica mas também os requisitos temporais da aplicação. Porém, para código específico de hardware, nós tivemos que contorná-lo quando estávamos executando a aplicação no PC desktop. Para aquelas classes de software que forneciam dados do sensor, nós somente substituímos por dados que representavam valores lidos do sensor. Deste modo, isto foi muito melhor do que dados coletados do sensor em tempo de execução, pois nós tivemos a oportunidade de exercitar os caminhos do código rodando o sistema no PC.

Com as técnicas de teste propostas para assegurar a corretude lógica e temporal, nós tivemos aproximadamente uma linha de teste a cada linha de código. A Figura 7.2 mostra a relação entre linhas de código de teste e da aplicação no projeto do oxímetro de pulso. A tabela 7.2 mostra os valores usados para esboçar o gráfico da Figura 7.2. O tamanho final do software embarcado ficou abaixo das 2000 linhas, excluindo comentários e espaços em branco.

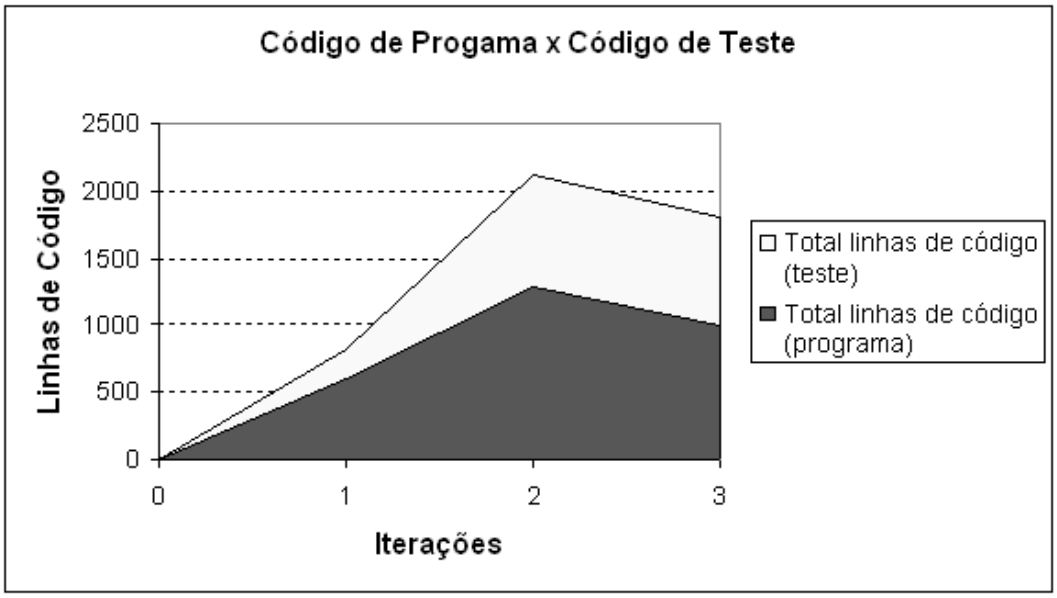

Figura 7.2: Total de Linhas de Código por Linhas de Teste - Experimento 1.

O hardware do sensor fornece três pacotes de dados (cada pacote tem 75 frames e cada frame consiste de 5 bytes) a cada segundo para a plataforma de desenvolvimento.

Tabela 7.2: Total de Linhas de Código (Programa e Teste)

Iteração	1	2	3
Total linhas de código (programa)	595	1296	1074
Total linhas de código (teste)	216	825	684

Sendo assim, o software embarcado do oxímetro de pulso recebe o pacote de dados, aplica um conjunto de funções para mostrar os valores de SpO_2 e HR na unidade do *display*. As funções que são responsáveis para calcular e mostrar os dados do sensor devem ser executadas de tal maneira que o *deadline* para recebimento do próximo pacote não seja perdido.

Para este propósito, foi necessário escrever algumas funções que estavam no caminho crítico na linguagem Assembly com o intuito de diminuir o tempo de execução. A Figura 7.3 mostra a quantidade de linhas de código em Assembly e C para cada sprint do projeto excluindo comentários e linhas em branco. A Tabela 7.3 mostra os valores usados para esboçar o gráfico da Figura 7.3. A terceira coluna mostra o número total de linhas de código contidas na solução final do oxímetro de pulso.

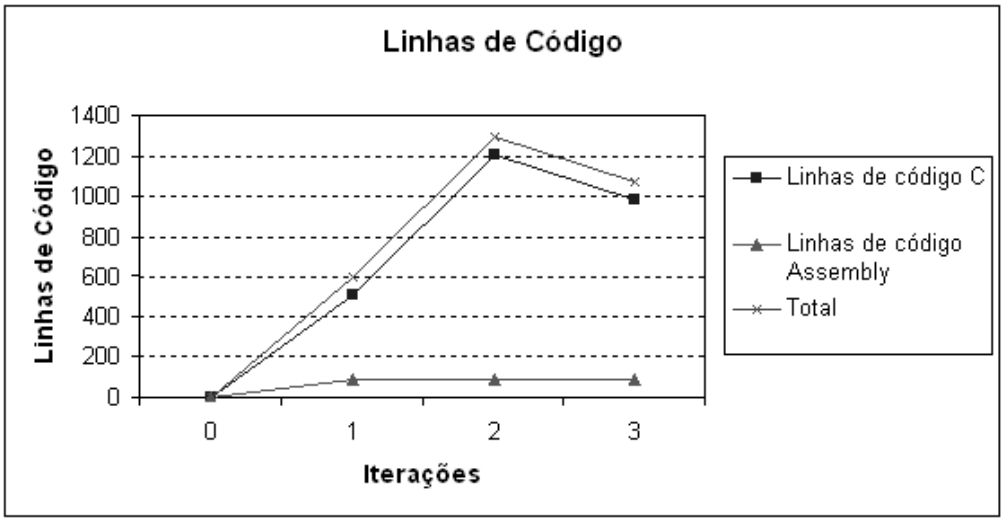

Figura 7.3: Quantidade Total de Linhas de Código - Experimento 1.

Foi necessário executar o software embarcado do oxímetro de pulso em um ambiente com restrições de memória e consumo de energia. Nossa plataforma de desenvolvimento tinha somente 12 KBytes de memória flash e tínhamos que dissipar uma potência máxima do sistema de 500mW. Deste modo, nós usamos o *Big Visible Chart* (BVC) proposto

Tabela 7.3: Total de Linhas de Código

Iteração	1	2	3
Linhas de código C	508	1209	987
Linhas de código Assembly	87	87	87
Total	595	1296	1074

por [9] com o propósito de rastrear as métricas de uso de memória e dissipação de potência. Ambos os gráficos foram atualizados regularmente e mantidos visíveis com o intuito de identificar tendências do sistema. A dissipação de potência foi mensurada conectando um amperímetro em série com a fonte de alimentação. O valor lido de corrente foi multiplicado pelo nível de tensão da fonte. Esta potência é fornecida para os componentes analógicos e digitais do oxímetro de pulso. As Figuras 7.4 e 7.5 mostram o uso de memória e dissipação de potência respectivamente.

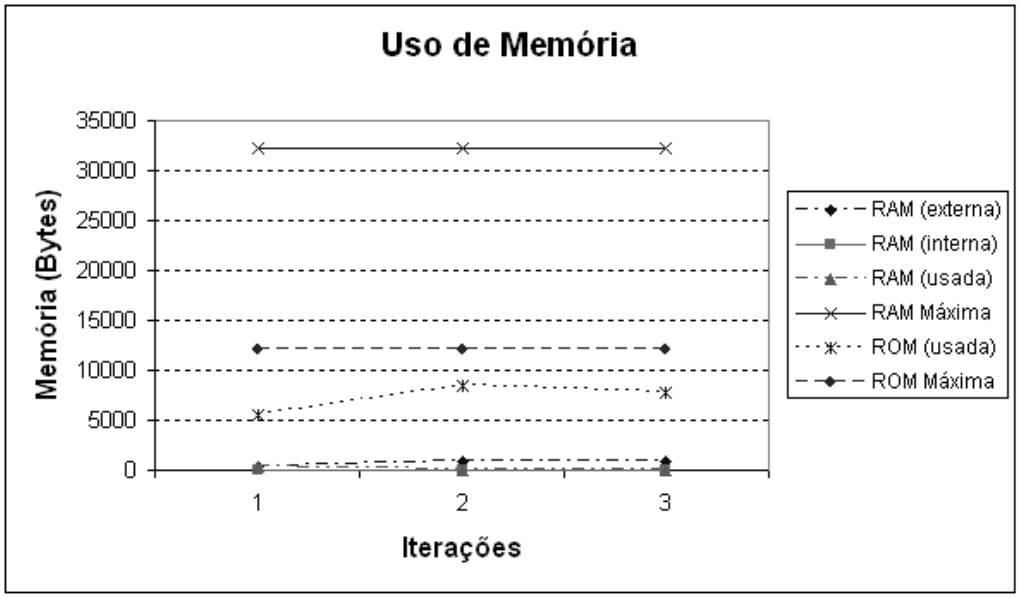

Figura 7.4: Uso de Memória - Experimento 1.

Tabela 7.4: Uso de Memória em cada Iteração (Bytes)

Iteração	1	2	3
RAM (externa)	339	864	795
RAM (interna)	15.1	28.7	21.3
RAM (usada)	354.1	892.7	816.3
ROM (usada)	5574	8513	7711

As Tabelas 7.4 e 7.5 mostram os valores usados para esboçar o gráfico das Figuras 7.4 e 7.5. No *sprint* 3 nós implementamos um conjunto de novas funcionalidades, mas também enfatizamos a otimização do código seguindo as técnicas de otimizações descritas nos processos da metodologia de desenvolvimento proposta. A coluna 4 mostra que o uso de memória e dissipação de potência foram substancialmente reduzidas no *sprint* 3.

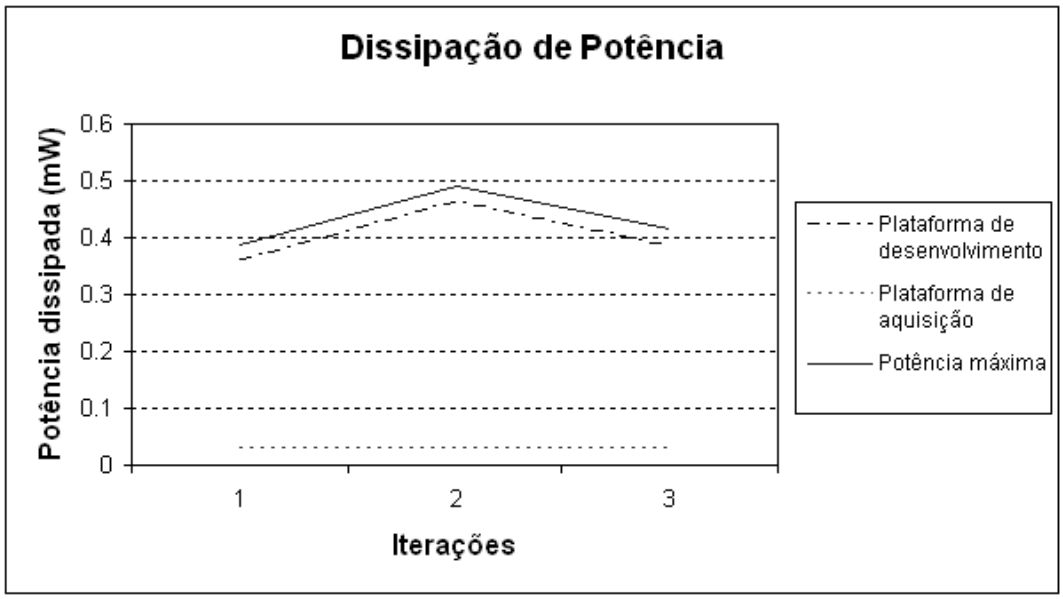

Figura 7.5: Dissipação de Potência - Experimento 1.

Tabela 7.5: Potência Dissipada em cada Iteração (mW)

Iteração	1	2	3
Plataforma de desenvolvimento	360	462	385
Plataforma de aquisição	29	29	29
Potência máxima	389	491	414

A Figura 7.6 mostra a evolução do backlog de produto. Como pode ser observado nesta figura, iniciamos o backlog de produto com um conjunto de requisitos funcionais e não funcionais totalizando um valor de negócio de aproximadamente 69. No sprint 2 nós identificamos mais requisitos que poderiam ser implementados para o projeto do oxímetro de pulso. Sendo que no sprint 3 nós mantivemos o mesmo número de requisitos para o sistema. Um outro ponto importante a ser mencionado deste gráfico é que no final do projeto tivemos que descartar alguns requisitos do backlog de produto.

Nós descartamos os requisitos referentes à análise da curva pletimosgráfica do sensor do oxímetro devido à restrição de tempo do projeto. Esta curva fornece informações de movimentos do paciente durante a aquisição do sinal de HR e SpO_2. No final do sprint 3 nós tivemos 65 valores de negócio implementados e testados. Somente 9 valores de negócio foram descartados. A Tabela 7.6 mostra os valores que foram utilizados para esboçar o gráfico da Figura 7.6. Como pode ser observado nesta tabela, os valores de negócio implementados ficaram bem próximos dos planejados.

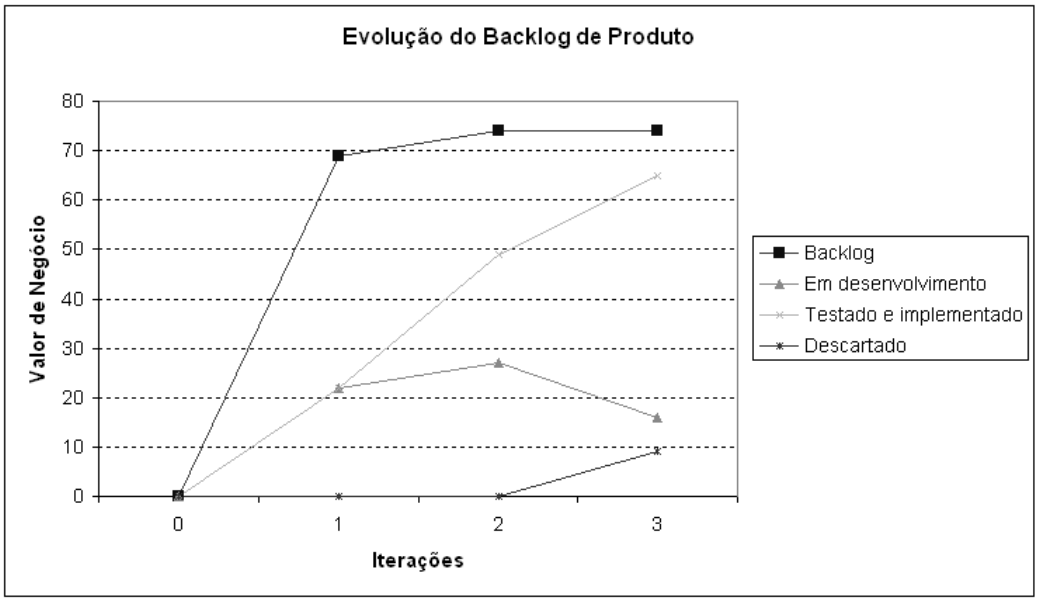

Figura 7.6: Evolução do Backlog de Produto.

Tabela 7.6: Evolução do Backlog de Produto

Iteração	1	2	3
Backlog	69	74	74
Em desenvolvimento	22	27	16
Testado e implementado	22	49	65
Testado e implementado	22	49	65
Descartado	0	0	9

As técnicas de teste descritas na Seção 4.5.8 formaram um meio perfeito de modularizar o software embarcado do oxímetro de pulso. A Figura 7.7 mostra a complexidade ciclomática do sistema para todos os sprint do projeto. Complexidade ciclomática nada mais é do que uma medida que indica a complexidade lógica de um algoritmo. Em outras

palavras, esta medida indica o número de caminhos linearmente independentes e, por conseguinte, o número mínimo que deveria ser razoavelmente testado para eliminar erros no algoritmo [28].

A Tabela 7.7 mostra os valores usados para esboçar o gráfico da Figura 7.7. Dados sobre o tamanho do código fonte, número de funções e complexidade ciclomática foram obtidos usando a ferramenta CCCC que analisa arquivos C/C++ e produz um relatório no formato HTML [49]. Todos os *sprints* foram observados e o resultado foi uma complexidade ciclomática média de 3 no fim do terceiro *sprint*. Para mais informações desta métrica no software embarcado do oxímetro, refira-se a [28].

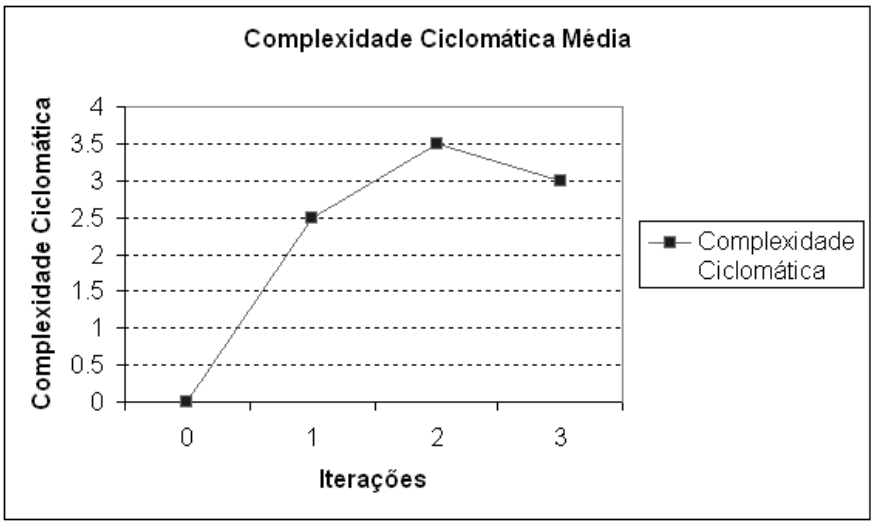

Figura 7.7: Complexidade Ciclomática - Experimento 1.

Tabela 7.7: Complexidade Ciclomática

Iteração	1	2	3
Complexidade Ciclomática	2.5	3.5	3

7.2 Soft-Starter Digital

Esta seção apresenta os resultados da metodologia proposta aplicada ao desenvolvimento do *soft-starter digital*. O projeto foi dividido em 2 diferentes sprints e desenvolvido por três alunos de mestrado sendo dois engenheiros de desenvolvimento e um líder de produto. O proprietário da plataforma, que também foi envolvido no desenvolvimento, estava

responsável pela definição da qualidade, cronograma, custo e requisitos do produto. No início do projeto, nós criamos uma lista de novas funcionalidades e requisitos com o intuito de unir todas as nossas necessidades com relação do desenvolvimento do produto. Estas informações foram incluídas no backlog de produto conforme descrito na seção 4.5.1.

Baseado nos valores de negócio dos requisitos, nós escolhemos a partir do backlog de produto um conjunto de funcionalidades para serem implementadas nos *sprints*do projeto. O primeiro *sprint* levou aproximadamente um mês e o segundo *sprint* levou 3 semanas para ser concluído. Como uma estratégia de gerenciamento de requisito, nós colocamos mais ênfase para entregar funcionalidades do sistema com alto valor de negócio agregado no início dos sprints. A Tabela 7.8 mostra o esforço mensurado, o valor de negócio e velocidade do desenvolvimento para cada *sprint*.

Tabela 7.8: Esforço estimado e mensurado (em horas)

	Sprint 1	Sprint 2
Esforço medido	70	84
Valor de negócio	11	16
Velocidade do *sprint*	0.157	0.19

A Figura 7.8 mostra o gráfico *burndown* do segundo *sprint* do projeto. Como pode ser visto nesta figura, o trabalho inicial foi estimado em 84 horas no início do *sprint*. Nos dias 16 e 23, as horas restantes estimadas aumentaram, pois novas tarefas foram descobertas e como resultado o esforço restante estimado foi também revisado. Um ponto de extrema importância a ser mencionado é que o backlog de sprint não foi atualizado diariamente pelos engenheiros de desenvolvimento. Isto se deve ao fato de que nós não estávamos trabalhando diariamente no projeto. Além disso, alguns membros da equipe estavam ocupados com as disciplinas sendo ministradas no mestrado e o projeto de dissertação.

A implementação final do *soft-starter* digital contém aproximadamente 1420 linhas de código em C e 195 linhas de script de build. Estes scripts de build auxiliaram na tarefa de compilação dos componentes de software do sistema. Além disso, com as técnicas de teste propostas para assegurar a corretude lógica e temporal, nós tivemos aproximadamente 854 linhas de teste usando o framework de teste unitário EmbUnit descrito na Seção 5.2.1. Isto significa uma linha de código de teste a cada duas linhas de código da aplicação.

Foi necessário executar o software embarcado do *soft-starter* digital em um ambiente

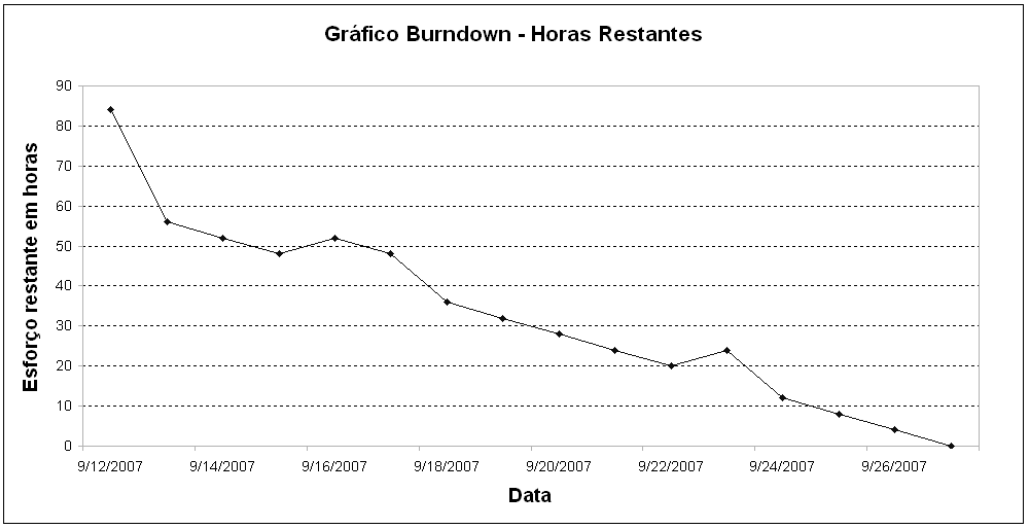

Figura 7.8: Gráfico Burndown do Sprint 2.

com restrições de memória e consumo de energia. Nossa plataforma de desenvolvimento tinha somente 12K bytes de memória flash e tínhamos que dissipar uma potência máxima do sistema de 1W. Depois de aplicar as técnicas de otimização do código descritas na Seção D, nós obtivemos na solução final uma dissipação de potência de 855 mW e um consumo de memória de aproximadamente 6.3K bytes.

7.3 Motor de Indução Monofásico

Esta seção apresenta os resultados da metodologia proposta aplicada ao desenvolvimento do simulador do motor de indução monofásico. O projeto foi dividido em 2 diferentes sprints e desenvolvido por três alunos de mestrado sendo dois engenheiros de desenvolvimento e um líder de produto. O proprietário da plataforma, que também foi envolvido no desenvolvimento, estava responsável pela definição da qualidade, cronograma, custo e requisitos do produto. No início do projeto, nós criamos uma lista de novas funcionalidades e requisitos com o intuito de unir todas as nossas necessidades com relação do desenvolvimento do produto. Estas informações foram incluídas no backlog de produto conforme descrito na seção 4.5.1.

Baseado nos valores de negócio dos requisitos, nós escolhemos a partir do backlog de produto um conjunto de funcionalidades para serem implementadas nos *sprints* do projeto. O primeiro *sprint* levou aproximadamente um mês e o segundo *sprint* levou 3 semanas

para ser concluído. Como uma estratégia de gerenciamento de requisito, nós colocamos mais ênfase para entregar funcionalidades do sistema com alto valor de negócio agregado no início dos sprints. A Tabela 7.9 mostra o esforço mensurado, o valor de negócio e velocidade do desenvolvimento para cada *sprint*.

Tabela 7.9: Esforço estimado e mensurado (em horas)

	Sprint 1	Sprint 2
Esforço medido	125	219
Valor de negócio	13	14
Velocidade do *sprint*	0.104	0.0639

A Figura 7.9 mostra o gráfico *burndown* do segundo *sprint* do projeto. Como pode ser visto nesta figura, o trabalho inicial foi estimado em 84 horas no início do *sprint*. Um ponto de extrema importância a ser mencionado é que o backlog de sprint não foi atualizado diariamente pelos engenheiros de desenvolvimento. Isto se deve ao fato de que nós não estávamos trabalhando diariamente no projetos. Além disso, alguns membros da equipe estavam ocupados com as disciplinas sendo ministradas no mestrado e os respectivos projetos de dissertação.

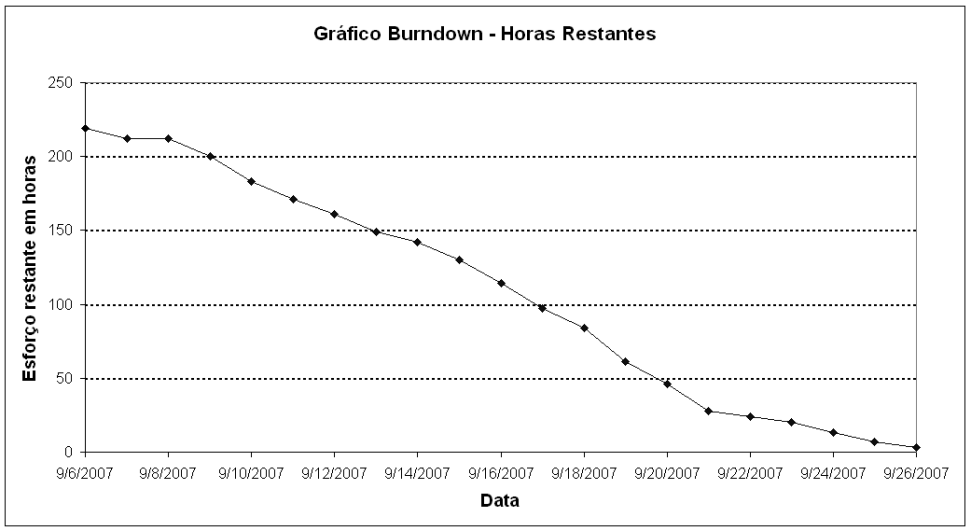

Figura 7.9: Gráfico Burndown do Sprint 2 do Simulador do Motor.

A implementação final do simulador do motor de indução contém aproximadamente 828 linhas de código em C e 129 linhas de script de build. Estes scripts de build auxiliaram na tarefa de compilação dos componentes de software do sistema. Além disso, com as

técnicas de teste propostas para assegurar a corretude lógica e temporal, nós tivemos aproximadamente 243 linhas de teste usando o framework de teste unitário EmbUnit descrito na Seção 5.2.1.

Foi necessário executar o software embarcado do simulador do motor de indução em um ambiente com restrições de memória e consumo de energia. Nossa plataforma de desenvolvimento tinha somente 12K bytes de memória flash e tínhamos que dissipar uma potência máxima do sistema de 1W. Depois de aplicar as técnicas de otimização do código descritas na Seção D, nós obtemos na solução final uma dissipação de potência de 774 mW e um consumo de memória de aproximadamente 3.9K bytes.

7.4 Resumo

Esta seção apresentou os resultados obtidos no desenvolvimento dos protótipos oxímetro de pulso, *soft-starter* digital e simulador do motor de indução. Para o projeto do oxímetro de pulso assim como *soft-starter* digital, nós obtivemos um aumento na velocidade do *sprint* a medida que o projeto avançava. Isto é justificado pelo fato de melhorarmos o nosso entendimento no domínio da aplicação, o ambiente de desenvolvimento e tecnologia envolvida. Porém, o mesmo não aconteceu para o projeto do simulador do motor de indução onde obtivemos um decréscimo da velocidade do sprint a medida que avançamos no projeto.

Outro ponto de extrema importância é que obtivemos bons resultados em termos de modularidade do software. Para o projeto do oxímetro de pulso, nós conseguimos obter uma complexidade ciclomática de valor 3. Além disso, adotando as práticas de otimização proposta pelo processo 4.5.11, nós conseguimos melhorar de forma significativa o tempo de execução, consumo de energia e uso de memória do sistema.

Como proposto pelo processo 4.5.2, nós focamos em primeiro implementar os requisitos funcionais de um dado sprint e depois os não-funcionais. Isso também nos ajudou a obter melhores resultados em termos de otimização do sistema e minimizou o problema da sub-otimização[1] descrito por [18].

As práticas de teste propostas na metodologia de desenvolvimento forneceram os pas-

[1]Este problema conhecido do inglês como "suboptimization" afirma que quando nós tentamos otimizar apenas partes do sistema separadamente, isso pode não levar a otimização global do mesmo.

sos necessários para verificar a corretude lógica e temporal das funções do sistema. Cada função do sistema foi testada usando todos os valores possíveis do parâmetro de entrada com o intuito de exercitar todos os caminhos do código. Além disso, nós também verificamos através do sistema de log a corretude temporal de funções críticas do sistema. Este sistema de log permitiu identificar perdas de *deadlines* das tarefas do sistema.

Capítulo 8

Conclusões e Trabalhos Futuros

Esta dissertação de mestrado descreveu uma metodologia de desenvolvimento baseada em princípios ágeis e sua aplicação no desenvolvimento dos estudos de caso do oxímetro de pulso, *soft-starter* digital e simulador de motor de indução. Com o propósito de criar a metodologia, nós escolhemos dois métodos ágeis XP e Scrum assim como padrões organizacionais nomeados neste trabalho como padrões ágeis. XP é uma coleção de práticas de desenvolvimento de software para aplicativos que executam em PC. Scrum é explicitamente voltado para o propósito de gerenciamento de projetos de software ágil. Os padrões ágeis fornecem meios para estruturar o processo de desenvolvimento de software das organizações.

Quando XP, Scrum e padrões ágeis são combinados, eles cobrem várias áreas do ciclo de desenvolvimento do sistema. Porém, a combinação do Scrum, XP e padrões ágeis não significam que eles podem ser diretamente usado para desenvolver sistemas embarcados. No entanto, este trabalhou teve um foco especial em solucionar os desafios adicionais de sistemas embarcados que são: *(i)* restrições no ambiente de execução (p.e., tempo de execução, consumo de energia, tamanho da memória de dados e programa) *(ii)* o custo de uma simples unidade de sistema embarcado deve ser a menor possível com o propósito de reduzir custos de produção, *(iii)* a solução do sistemas geralmente envolve componentes de hardware e software, *(iv)*a plataforma de desenvolvimento é muito caro e muitas vezes não está disponível no início do processo de desenvolvimento.

Para tratar destes desafios adicionais, pequenas mudanças foram necessárias para: *(i)* adotar modelos, processos e ferramentas para otimizar o projeto do produto em vez de ir

por caminhos que podem não satisfazer as restrições do sistema, *(ii)* apoiar o desenvolvimento de hardware e software através de um fluxo compreensivo desde a especificação até implementação, *(iii)* instanciar a plataforma do sistema baseado nas restrições da aplicação em vez de adotar uma instância de plataforma para um dado produto que possua muito mais recursos que o necessário, e *(iv)* usar plataforma do sistema para conduzir várias explorações de espaço de projeto com o intuito de analisar o desempenho do sistema.

Para ilustrar o uso dos processos e ferramentas de metodologia proposta, nós mostramos como a metodologia foi aplicada para desenvolver os projetos do oxímetro de pulso, *soft-starter* digital e o simulador do motor de indução. Para estes projetos, o uso das plataformas de desenvolvimento e aquisição (para o projeto do oxímetro de pulso) reduziram substancialmente o tempo de desenvolvimento do produto. Além disso, nós aplicamos um conjunto de técnicas de teste com o propósito de garantir a corretude lógica e temporal dos sistemas desenvolvidos.

Estas técnicas levaram a uma melhor modularidade do sistema de software. Como trabalhos futuros, deve-se ainda pesquisar modelos que possam conter informações suficientes sobre a implementação física do sistema em um alto nível de abstração e alcançar melhores resultados em termos de corretude. Além disso, deve-se realizar estudos experimentais em condições ideais com os processos de gerenciamento de projeto. Depois disso, pode-se iniciar o processo de transferência da metodologia proposta fase após fase na indústria através do uso de técnicas de melhorias de processos.

8.1 Contribuições

A metodologia de desenvolvimento ágil proposta nesta dissertação de mestrado tem como objetivo definir papéis e responsabilidades e fornecer processos, práticas e ferramentas para ser aplicado em projetos de sistema embarcado de tempo real. A metodologia é composta por três diferentes grupos de processo que são: *(i)* o grupo de processos de plataforma do sistema que objetiva instanciar a plataforma para um dado produto, *(ii)* o grupo de processos do desenvolvimento do produto que oferece práticas para desenvolver os componentes da aplicação e integrá-los na plataforma e finalmente *(iii)* O grupo de

processos de gerenciamento de produto que monitora e controla os parâmetros de custo, qualidade, tempo e escopo do produto.

Além disso, a metodologia proposta define quatro papéis que são: *Proprietário da Plataforma, Líder de Produto, Líder de Funcionalidade e Equipe de desenvolvimento.* Para conduzir os processos da metodologia de desenvolvimento ágil proposta, a mesma define o ciclo de vida dos processos em cinco fases: *Exploração, Planejamento, Desenvolvimento, Liberação e Manutenção.* Sendo assim, uma das principais contribuições deste trabalho pode ser definida da seguinte maneira:

a) Adotar modelos, processos e ferramentas para otimizar o projeto do produto em vez de ir por caminhos que podem não satisfazer as restrições do sistema.

b) Apoiar o desenvolvimento de hardware e software através de um fluxo compreensivo desde a especificação até implementação.

c) Instanciar a plataforma do sistema baseado nas restrições da aplicação em vez de adotar uma instância de plataforma para um dado produto que possua muito mais recursos que o necessário.

d) Usar plataforma do sistema para conduzir várias explorações de espaço de projeto com o intuito de analisar o desempenho do sistema.

De acordo com a revisão literária realizada durante este trabalho, nós não identificamos nenhum trabalho na área de métodos ágeis que propõe processos, práticas e uso de ferramentas em uma maneira sistemática para o desenvolvimento de sistemas embarcados. Além disso, este trabalho fornece uma parcela de contribuição às idéias e objetivos do paradigma de projeto baseado em plataforma proposto por [55, 54]. Embora a metodologia proposta por Vicentelli e Martin seja totalmente conceitual, ela serviu como base para o desenvolvimento da metodologia proposta.

Outra contribuição significante com a realização deste trabalho são as técnicas de teste que foram usadas para assegurar a corretude lógica e temporal do sistema de software. A automação de testes unitários e funcionais forma a base das metodologias de desenvolvimento ágil, como por exemplo, Scrum e XP. No entanto, as práticas propostas por estes

métodos focam em aplicativos que adotam o paradigma de orientação à objeto e executam no PC desktop. Este trabalho contribuiu no sentido de exercitar tanto caminhos do código puro quanto o código específico de hardware.

Foi mostrado que para executar os casos de teste em uma maneira automatizada utilizando o framework de teste unitário [50], nós tivemos que implementar um mecanismo para rodar o software embarcado tanto no PC desktop quanto na plataforma alvo. Além disso, nós dividimos os componentes de software para teste em duas categorias: código puro e código específico de hardware. Para o código puro, nós usamos as técnicas convencionais de teste.

Para testar o código específico de hardware, nós tivemos que dividi-lo em duas subcategorias (código que controla o hardware e código guiado pelo ambiente) e propor algumas adaptações na maneira convencional de testar o software. Para aqueles componentes de software que controlavam o hardware, nós tivemos que executar os casos de teste manualmente na plataforma alvo. Para aqueles componentes que são guiados pelo ambiente, a estratégia adotada foi substituir dados coletados a partir do ambiente por dados contidos em um arquivo. Para estas duas subcategorias, nós procuramos exercitar ao máximo todos os caminhos do código. Além disso, para todos os casos de teste, foi realizada uma avaliação da corretude temporal.

8.2 Experiência

A metodologia de desenvolvimento ágil proposta fornece práticas de desenvolvimento de software embarcado que podem ser usadas desde a concepção de um produto até a fase de manutenção. O conhecimento adquirido nesta dissertação foi de fundamental importância para a melhoria das minhas atividades no campo profissional. Algumas práticas que são fornecidas na metodologia, já estão sendo aplicadas no meu dia a dia de trabalho. Por exemplo, a prática teste antes do desenvolvimento foi usada para desenvolver um device driver no Linux para uma plataforma de TV digital da NXP semicondutores (antiga Philips semicondutores). Este driver contém aproximadamente 34 funções IOCTL que fornecem um conjunto de funcionalidades para controlar o componente de demodulação e decodificação de um sinal digital terrestre.

Outros princípios ágeis como desenvolvimento iterativo e incremental e planejamento adaptativo também estão sendo utilizados em um projeto de software embarcado que eu estou participando atualmente. O desenvolvimento iterativo e incremental está ajudando a reduzir a complexidade do desenvolvimento e reduzir riscos do projeto. O planejamento adaptativo permite que, depois de definido os requisitos do sistema para uma dada iteração, os mesmo não sejam mais alterados. Isto ajudou a manter o foco nas atividades que foram planejadas para uma dada iteração e facilita também o gerenciamento do projeto.

Um outro ponto de extrema importância em termos de aprendizado com a metodologia proposta é o conhecimento na área de sistemas embarcados. Eu pude melhorar de forma significativa os meus conhecimentos nesta área mesmo embora eu já tenha estudado durante a minha graduação em engenharia elétrica e também esteja trabalhando com o desenvolvimento de tais sistemas. Não somente na área de metodologias de desenvolvimento como também em aspectos formais aplicados ao software embarcado, mesmo não tendo trabalhado com este assunto na minha dissertação de mestrado. O conteúdo das disciplinas para área de concentração em sistemas embarcados permitiu que eu ampliasse meus conhecimentos substancialmente, podendo assim ser utilizado para a minha futura tese de doutorado na área de verificação formal para sistemas embarcados.

8.3 Problemas

Um dos problemas cruciais que eu tive na minha dissertação de mestrado foi a validação das práticas de gerenciamento de projeto. Como a metodologia proposta fornece práticas que vão desde a concepção do produto até a fase de manutenção, nós tivemos que pensar cuidadosamente na execução dos experimentos com o propósito de validar o que estava sendo proposto. Sendo assim, três estudos de caso foram propostos na minha dissertação com o intuito de exercitar tanto a parte de desenvolvimento e gerenciamento quanto atacar diferentes áreas de aplicação de sistemas embarcados, que incluem: *dispositivos médicos, controle de processo, comunicação, eletro-eletrônicos* e assim por diante.

O primeiro experimento a ser conduzido na minha dissertação foi o desenvolvimento do oxímetro de pulso. A principal dificuldade envolvida na construção deste equipamento

foi a falta de estrutura de laboratórios do curso de sistemas embarcados do mestrado da UFAM. Para o desenvolvimento de qualquer tipo de sistema embarcado, existe sempre a necessidade de analisar um sinal, soldar um fio, construir uma placa de interfaceamento e assim por diante. Além disso, a necessidade de ter componentes de hardware no laboratório é constante. Por exemplo, precisávamos ter a conexão de um teclado a plataforma de desenvolvimento. Nós tivemos que realizar esta tarefa emendando dois diferentes cabos emprestados de um professor.

Para o segundo e terceiro experimento tivemos ainda mais dificuldades. Estes dois experimentos consistem basicamente no desenvolvimento do soft-starter digital e simulador do motor de indução. Estes dois experimentos visavam explorar as práticas de gerenciamento de projeto da metodologia proposta. Sendo assim, nós começamos os dois projetos com oito alunos de mestrado, quatro para cada equipe. Nós também definimos o escopo. tempo e métricas de qualidade de projeto. No entanto, os requisitos planejados para o primeiro sprint do projeto não foram alcançados e quatros alunos desistiram. Além disso, os outros quatro alunos remanescentes estavam focados nas discplinas do mestrado, ou seja, não tinham tempo disponível para trabalhar nos projetos.

Depois disso, nós decidimos continuar os projetos com os quatro alunos de mestrado em um segundo sprint. Sendo assim, as equipes ficaram dividas em duas equipes de dois alunos cada uma. Neste segundo sprint, os alunos ainda estavam envolvidos nas disciplinas do mestrado sendo liberados somente no final do mês de agosto. Uma outra dificuldade crucial que nós tivemos na execução destes dois experimentos foi a falta de laboratório de eletrônica do curso de sistemas embarcado. Para solucionar este problema, nós tivemos que solicitar ao coordenador do curso de engenharia elétrica permissão para o uso do laboratório.

Nas primeiras seções de laboratório tivemos alguns problemas de comunicação para agendar um horário para o uso do laboratório. Porém, isto foi resolvido depois. No entanto, continuamos tendo dificuldades para obter materiais para a execução dos experimentos. Por exemplo, os dois micro-controladores (um do projeto soft-starter e outro do simulador) precisavam estabelecer uma comunicação através das portas de entrada e saída da plataforma de desenvolvimento. Para resolver esta situação, um dos membros da equipe teve que desmontar um PC desktop para remover os fios a serem utilizados

nos experimentos. Em cada extremidade do fio nós isolamos com uma fita isolante, mas mesmo assim não ficou algo robusto para a comunicação dos micro-controladores. Mesmo com todas estas dificuldades, nós conseguimos no final do sprint 3 alcançar os requisitos de maior valor de negócio agregado.

8.4 Limitações

Uma das principais limitações da metodologia proposta são as práticas de gerenciamento de projeto. Tendo em vista que nós tínhamos uma equipe reduzida e que não estava 100% focada nas atividades do projeto, inviabilizou bastante a validação das práticas. Por exemplo, a metodologia proposta define que sejam feitas reuniões diárias com o intuito de monitorar as atividades sendo desenvolvidas pelos membros da equipe. Como nós não estávamos trabalhando diariamente no projeto então ficava difícil de monitorar as atividades. Além disso, a metodologia proposta define que o backlog de sprint seja preenchido diariamente. No entanto, os membros da equipe costumavam a preencher o backlog somente no final do sprint.

Uma outra prática definida na metodologia proposta é que o proprietário da plataforma participe de forma ativa dos projetos através da definição de requisitos para o sistema, métricas de qualidade, revisão dos sprints entre outras atividades. Como o nosso proprietário da plataforma estava distante da equipe de desenvolvimento, ficou difícil envolvê-lo em algumas atividades como, por exemplo, a revisão do sprint. Nesta reunião, o proprietário da plataforma avalia as funcionalidades que foram implementadas e fornece, para um próximo sprint, comentários para melhorar o produto e requisitos. Além disso, o proprietário da plataforma geralmente está bem próximo do cliente com o intuito de definir requisitos para o sistema.

Sendo assim, deve-se experimentar a metodologia proposta em um projeto onde se tenha investimento para fornecer uma boa estrutura aos membros da equipe e que as pessoas estejam 100% dedicadas ao projeto. Um outro ponto de extrema importância a ser mencionado é com relação ao uso de ferramentas para suportar os processos da metodologia proposta. Os engenheiros devem usar ferramentas para lidar com severeas restrições de desempenho, custo, consumo de energia que são típicos em sistemas embarcados. Além

disso, deve-se garantir a confiabilidade do sistema a ser desenvolvido.

Com o uso da metodologia proposta, o engenheiro pode usar algumas práticas para contornar aspectos de tempo de execução, segurança e custo do sistema. No entanto, nós não conseguimos propor uma maneira de analisar o consumo de energia em nível de função. Ou seja, nós tivemos que analisar o consumo de energia monitorando a corrente na entrada do sistema através de um amperímetro. Porém, é interessante termos ferramentas que possibilitem o engenheiro analisar quais funções do sistema são responsáveis por grande parte do consumo e então otimizá-las uma a uma em processo iterativo e incremental.

Além disso, é interessante desenvolver uma ferramenta que possibilite o engenheiro de verificar certas propriedades do sistema. Embora a metodologia proposta forneça várias práticas para verificar a corretude lógica e temporal do sistema, deve-se usar uma ferramenta de checagem de modelo (*model checking*) para garantir que certas propriedades estejam realmente presentes no modelo. Esta ferramenta verificaria o modelo e geraria um contra-exemplo caso o modelo não satisfaça a especificação. Sendo assim, com o uso desta ferramenta, pode-se melhorar ainda mais a confiabilidade do sistema desenvolvido.

8.5 Trabalhos Futuros

Para continuação deste trabalho de pesquisa, há uma série de propostas que podem ser analisadas com o propósito de iniciar novos trabalhos de graduação, dissertações de mestrado e teses de doutorado. Entre estas propostas destacam-se:

- Desenvolver algoritmos para realizar a análise do consumo de energia, tempo de execução e memória de sistemas de software usando os cenários dos casos de teste da aplicação;
- Propor uma metodologia para realizar a verificação formal do sistema de software usando, por exemplo, técnicas de checagem de modelo e verificação dirigida por cobertura;
- Usar modelos de plataforma escritos em SystemC, VHDL ou Verilog com o intuito de possibilitar o desenvolvimento do sistema sem uso da plataforma física que muitas

vezes é um recurso escasso e caro para ser compartilhado entre os membros da equipe.

A próxima subseção fornece o comentário final desta dissertação de mestrado.

8.6 Comentário Final

Existe essencialmente duas área de conhecimento no desenvolvimento de sistemas embarcados. A primeira área é compartilhada com outros projetos de desenvolvimento de software de todos os tipos. De posse dos requisitos iniciais, uma estimativa é realizada e uma data de entrega do projeto é definida. Apesar das dificuldades encontradas para validar as práticas de gerenciamento de projeto, nós acreditamos que a metodologia proposta representa uma abordagem adequada para gerenciar projetos de sistemas embarcados.

A segunda área de conhecimento é única ao desenvolvimento de sistemas embarcados. A essência deste tipo de sistema é implementar um conjunto de funcionalidades enquanto um conjunto de restrições (p.e., consumo de energia, desempenho, custo) é satisfeito. Para esta área de conhecimento, a metodologia proposta fornece uma abordagem sistemática para construção deste tipo de sistema. Porém, foram realizados experimentos somente na área de dispositivos médicos e sistemas de controle. Experimentos em outras áreas de aplicação de sistemas embarcados devem ser executados.

Referências Bibliográficas

[1] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen. New directions on agile methods: A comparative analysis. *IEEE Software*, pages 244–254, 2003.

[2] A. Ahmed. *Eletrônica de Potência*. Prentice Hall, Inc., 2000.

[3] Agile Alliance. *Manifesto for Agile Software Development*. Disponível em www.agilemanifesto.org. Última visita no dia 29 de Outubro, 2007.

[4] S. Arató, P; Juhász, A. Z. Mann, A. Orbán, and D. Papp. Hardware-software partitioning in embedded system design. *IEEE International Symposium on Intelligent Signal Processing*, pages 197–202, 2003.

[5] Atmel. At89s8252 datasheet. *Disponível em http://www.atmel.com/atmel/acrobat/doc0401.pdf. Última visita no dia 26 de Outubro*, 2007.

[6] Inc. Baldor. *Digital Soft-Start: Installation and Operating Manual*. Disponível em www.baldor.com. Última visita no dia 30 de Outubro, 2007.

[7] R. Barreto. *A Time Petri Net-Based Methodology for Embedded Hard Real-Time Systems Software Synthesis*. Phd. thesis, Centro de Informática. Universidade Federal de Pernambuco, April 2005.

[8] E. Barros, S. Cavalcante, M. Lima, and C. Valderrama. *Hardware/Software Co-design: Projetando Hardware e Software Concorrentemente*. Escola de computação, IME-USP, São Paulo., 2000.

[9] K. Beck and C. Andres. *Extreme Programming Explained - Embrace Change*. Second Edition, Addison-Wesley, 1999.

[10] S. Berczuk and B. Appleton. *Software Configuration Management Patterns.* First Edition, Addison-Wesley, 2002.

[11] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages.* Harlow: Addison-Wesley, 2001.

[12] P. L. Carloni, F. Bernardinis, C. Pinello, A. L. S. Vicentelli, and M. Sgroi. *Platform-Based Design for Embedded Systems.* CRC Press, The Embedded Systems Handbook. Disponível em www.cs.columbia.edu/ luca/research/pbdes.pdf. Última visita no dia 30 de Outubro, 2007.

[13] R. Chen, M. Sgroi, L. Lavagno, Martinm G., A. Sangiovanni-Vincentelli, and J. Rabaey. *UML e Projeto Baseado em Plataforma.* University of California at Berkeley and Cadence Design Systems, 2004.

[14] M. Cohn. *Agile Estimating and Planning.* Robert Martin Series, Prentice Hall, 2005.

[15] J. E. Cooling. *Software Engineering for Real-Time Systems.* First Edition, Addison-Wesley., 2003.

[16] J. O. Coplien and D.C. Schmidt. *Organizational Patterns of Agile Software Development.* First Edition, Prentice Hall, 2004.

[17] L.C. Cordeiro. *Pulse Oximeter Source Code.* Available at http://https://sourceforge.net/cvs. Last visit on November 6th, 2007.

[18] Principia Cybernetica. *Suboptimization Problem.* Available at http://pespmc1.vub.ac.be/SUBOPTIM.html. Last visit on 26th December, 2006.

[19] M. Fowler. Is design dead?. *Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.*, 2001.

[20] D. Gajski and F. Vahid. Especification and design of embedded hardware-software systems. *IEEE Design and Test of Computers*, 1994.

[21] D. Gajski, F. Vahid, and S. Narayan. A system-design methodology: Executable-specification refinement. *European Conference on Design Automation, Paris, France*, 1994.

[22] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and design of embedded systems*. Specification and design of embedded systems. Prentice Hall., 1994.

[23] K. Ghosh. *Writing Efficient C and C Code Optimization*. The Code Project. Disponível em http://www.codeproject.com. Última visita no dia 30 de Outubro, 2007.

[24] E. M. Goldratt. *Theory of Constraints*. North River Press, 1999.

[25] B. Greene. Agile methods applied to embedded software development. *Proceeding of the Agile Development Conference (ADC'04).*, 2004.

[26] G. Hu and G. Greenwood. Evolutionary approach to hardware/software partitioning. *IEEE Proceedings of the Comput. Digit. Tech.,*, 145(3):203–209, 1994.

[27] Nonin Medical Devices Inc. Oem iii module: Internal pulse oximetry. *Disponível em http://www.nonin.com/products.asp. Última visita no dia 27 de Outubro*, 2007.

[28] Software Engineering Institute. *Cyclomatic Complexity*. Published at the Carnegie Mellon University. Available at www.sei.cmu.edu/str. Last visit on 29th October, 2007.

[29] P. Kimura, R. S. Barreto, and L. C. Cordeiro. *Projeto e Implementação de um Plug-in Baseado no Framework do OSGi para Particionamento de Hardware/Software*. Trabalho de Iniciação Científica. Universidade Federal do Amazonas. Conselho Nacional de Desenvolvimento Científico e Tecnológico., 2007.

[30] P. Koopman. Embedded system design issues (the rest of the story). *Proceedings of the International Conference on Computer Design (ICCD96)*, pages 310–317, 1996.

[31] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 2002.

[32] P. Kukkala, J. Riihimäki, M. Hännikäinen, T. Hämäläinen, and K. Kronlöf. Uml 2.0 profile for embedded system design. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05).*, 2005.

[33] C. Larman. *Agile and iterative development: a manager's guide*. First Edition, Agile Software Development Series, Addison-Wesley, 2004.

[34] M. Lindvall, D. Muthig, A. Dagnino, C. Wallin, M. Stupperich, D. Kiefer, J. May, and Kähkönen T. Agile software development in large organizations. *IEEE Computer Society*, 37(12):26–34, October 2006.

[35] P. Manhart and K. Schneider. Breaking the ice for agile development of embedded software: An industry experience report. *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, pages 36–47, 2004.

[36] G. Martin. Uml for embedded systems specification and design: Motivation and overview. *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE'02).*, 2002.

[37] Microgênios. Manual de operação da plataforma de desenvolvimento 8051nx. *Disponível em http://www.microgenios.com.br. Última visita no dia 26 de Outubro*, 2007.

[38] K. Nguyen, Z. Sun, , and P. Thiagarajan. Model-driven soc design via executable uml to systemc. *Proceedings of the 25th IEEE International Symposium on Real-Time Systems, Page(s) 459-468*, pages 459–468, 2004.

[39] M. Jr. Oliveira, S. Neto, P. Maciel, R. Lima, A. Ribeiro, R. Barreto, E. Tavares, and F. Braga. Analyzing software performance and energy consumption of embedded systems by probabilistic modeling: An approach based on coloured petri nets. *ICATPN*, pages 261 – 281, 2006.

[40] D. O. Patrick and L. C. Cordeiro. *Ferramenta para Captura de Log do Plataforma 8051NX da Microgênios*. Universidade Federal do Amazonas., 2007.

[41] M. R. Prasad and V.V. Sastry. Rapid prototyping tool for a fuzzy logic based soft-starter. *PCC-Nagaoka, IEEE*, pages 877–880, 1997.

[42] T. Punkka. Unit test framework for embedded c systems. *Disponível em http://embunit.sourceforge.net/. Última visita no dia 26 de Outubro*, 2007.

[43] Richard. *C Optimization: How to make your C, C++ or java program run faster with little effort.* Disponível em http://www.rddvs.com/FasterC/. Última visita no dia 06 de Novembro, 2007.

[44] J. Ronkainen and P. Abrahamsson. Software development under stringent hardware constraints: Do agile methods have a chance? *eXtreme Programming Conference*, 2003.

[45] N. V. Schooenderwoert and N. Morsicato. Taming the embedded tiger - agile test techniques for embedded software. *Proceedings of the Agile Development Conference (ADC'04)*, 2004.

[46] K. Schwaber and M. Beedle. *Agile Software Development with Scrum.* First Edition, Series in Agile Software Development, Prentice Hall, 2002.

[47] Philips Semiconductors. *The I2C-bus and how to use it.* Disponível em www.mcc-us.com/i2chowto.htm. Última visita no dia 30 de Outubro, 2007.

[48] I. Sommerville. *Software Engineering 7.* Addison Wesley, 2006.

[49] SourceForge. *C and C++ Code Counter.* Disponível em sourceforge.net/projects/cccc. Última visita no dia 30 de Outubro, 2007.

[50] SourceForge. *embUnit: Unit Test Framework for Embedded C Systems.* Disponível em http://embunit.sourceforge.net/. Última visita no dia 30 de Outubro, 2007.

[51] G. Stitt, R. Lysecky, and F. Vahid. Dynamic hardware/software partitioning: A first approach. *Design and Automation Conference (DAC'03)*, pages 250–255, 2003.

[52] Inc. Sun Microsystems. *Java Platform, Standard Edition.* Disponível em java.sun.com/javase/. Última visita no dia 30 de Outubro, 2007.

[53] V. D. Toro. *Basic Electric Machines.* Prentice Hall, Inc., 1990.

[54] A. S. Vicentelli, P. L. Carloni, F. Bernardinis, and M. Sgroi. Benefits and challenges for platform-based design. *Proceedings of the Design Automation Conference*, (41):409–414, 2004.

[55] A. S. Vicentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design and Test of Computers*, 18(6):23–33, 2001.

[56] K. Villela. Definição e construção de ambientes de desenvolvimento de software orientados a organização. *Tese de PhD, programa de engenharia de sistema e computação da Universidade Federal do Rio de Janeiro.*, 2004.

Apêndice A

Abreviações

API - **A**pplication **P**rogramming **I**nterface

ASIC - **A**pplication **S**pecific **I**ntegrated **C**ircuit

ATM - **A**utomated **T**eller **M**achine

BJT - **B**ipolar **J**unction **T**ransistors

CCU - **C**enter **C**are **U**nits

DSP - **D**igital **S**ignal **P**rocessor

FPGA - **F**ield-**P**rogrammable **G**ate **A**rray

GTO - **G**ate **T**urn **O**ff

GUI - **G**raphical **U**ser **I**nterface

HP - **H**orse**p**ower

IID - **I**terative and **I**ncremental **D**evelopment

ILP - **I**nteger **L**inear **P**rogramming

LED - **L**ight **E**mitting **D**iodes

MOP - **M**ultiple-**O**bjective **O**ptimization

PBD - **P**latform-**B**ased **D**esign

PC - **P**ersonal **C**omputer

SAP - **S**ingle **A**pplication **P**rocessor

SCR - **S**ilicon **C**ontrolled **R**ectifiers

SOTR - **S**istema **O**peracional de **T**empo **R**eal

SCM - **S**oftware **C**onfiguration **M**anagement

TOC - **T**heory **o**f **C**onstraints

TDD - **T**est-**D**riven **D**evelopment

TXM - **T**he ne**X**t **M**ethodology

XP - e**X**treme **P**rogramming

Apêndice B

Requisitos dos Estudos de Caso e Infra-Estrutura

B.1 Requisitos do Oxímetro de Pulso

Esta seção descreve os requisitos funcionais e não funcionais do oxímetro de pulso.

Requisitos Funcionais da Aplicação

/RF10/ O sistema deve ser capaz de mensurar os dados de freqüência cardíaca e saturação de oxigênio no sangue do paciente usando um método não evasivo.

/RF20/ O usuário deve ser capaz de visualizar, a cada segundo, os dados de freqüência cardíaca e saturação do oxigênio no sangue do paciente através do display.

/RF30/ O sistema deve ser capaz de detectar a conexão e desconexão do sensor através da porta serial.

/RF40/ O sistema deve possibilitar o usuário de armazenar os dados de freqüência cardíaca e saturação do oxigênio na memória RAM do micro-controlador.

/RF50/ Um aplicativo que executa no PC deve ser desenvolvido para que os dados de freqüência cardíaca e saturação armazenados na memória RAM do micro-controlador sejam analisados no PC.

/RF60/ O aplicativo que executa no PC deve ser capaz de copiar o log da memória RAM do micro-controlador e transferi-los para o sistema de arquivos do PC.

/RF70/ O aplicativo que executa no PC deve disponibilizar os dados da memória do

micro-controlador no formato csv (dados separados por vírgulas) para que o mesmo possa ser aberto por ferramentas do microsoft office ou openoffice.

/RF80/ A aplicação deve verificar se os frames de bytes enviados pelo sensor estão corretos.

/RF90/ A aplicação deve verificar a curva pletimosgráfico do sensor com o intuito de detectar falhas no sinal de saturação de oxigênio do paciente.

Requisitos Não-Funcionais da Aplicação

/RNF100/ O número de defeitos do sistema deve ser o menor possível.

/RNF110/ O sistema deve ser alimentado com uma bateria comum de 9V.

/RNF120/ O tamanho do software a ser desenvolvido no micro-controlador deve ocupar no máximo 12KB de memória.

/RNF130/ A quantidade de bytes armazenados na memória RAM do micro-controlador deve ser no máximo 32KB.

Requisitos Funcionais de Interface com o Usuário

/RF140/ Uma interface homem-máquina (display e teclado) deve estar presente na solução final de modo que o usuário possa interagir com o sistema.

/RF150/ O usuário deve ser capaz de ajustar a freqüência de amostragem dos dados do sensor que serão mostrados no display do micro-controlador.

/RF160/ O usuário deve ser capaz de ajustar a taxa de amostragem dos dados que serão armazenados na memória RAM do micro-controlador.

/RF170/ O sistema deve indicar para o usuário através de uma mensagem no display a ausência de sinais de pulsos.

/RF180/ O alarme do sensor deve ser disparado toda vez que o sensor fornecer dados falsos para a análise.

Requisitos Funcionais para os Drivers dos Dispositivos

/RF190/ O driver do dislpay deve ser desenvolvido para que permita que o desenvolvedor escreva textos em qualquer posição do display.

/RF200/ O driver do teclado deve ser desenvolvido de tal modo que possibilite o usuário ajustar os parâmetros do oxímetro de pulso.

/RF210/ O driver da porta serial deve ser desenvolvido para que seja possível estabelecer um meio de comunicação entre o sensor e o micro-controlador.

Requisitos Funcionais para o Log do Sistema

/RF220/ O desenvolvedor deve ser capaz de habilitar/desabilitar o armazenamento de log do sistema.

/RF230/ O armazenamento do log deve ser feito de uma maneira circular na RAM do micro-controlador com o propósito de não modificar o comportamento da aplicação.

/RF240/ O log armazenado na memória RAM do micro-controlador deve ser enviado para o PC através da porta serial do micro-controlador.

/RF250/ O usuário deve ser capaz de habilitar/desabilitar o armazenamento de dados de SpO2 e HR na memória do micro-controlador.

B.2 Requisitos do Soft-Starter Digital

As próximas seções descrevem os requisitos funcionais e não funcionais do *soft-starter* digital.

Requisitos Funcionais da Aplicação

/RF10/ O sistema deve controlar automaticamente a partida do motor monofásico.

/RF20/ O sistema deve ler o sinal de tensão fornecido pelo sensor através do conversor analógico-digital.

Requisitos Não-Funcionais da Aplicação

/RNF30/ O sinal PWM gerado nos pinos do micro-controlador deve atender os requisitos temporais da aplicação.

/RNF40/ O software de controle do micro-controlador deve ser projetado de tal forma que possibilite no futuro a geração de sinais SPWM para um motor trifásico.

/RNF50/ O número de defeitos do sistema deve ser o menor possível.

/RNF60/ O sistema deve ser alimentado com uma bateria comum de 9V.

/RNF70/ O sistema deve fornecer uma boa usabilidade.

Requisitos Funcionais de Interface com o Usuário

/RF80/ Uma interface homem-máquina (display e teclado) deve estar presente na solução final de modo que o usuário possa interagir com o sistema.

/RF90/ O usuário deve ser capaz de visualizar o sinal de corrente e tensão do sensor.

/RF100/ O desenvolvedor deve ser capaz de habilitar/desabilitar o armazenamento de log do sistema.

Requisitos Funcionais para os Drivers dos Dispositivos

/RF110/ O driver do dislpay deve ser desenvolvido para que permita que o desenvolvedor escreva textos em qualquer posição do display.

/RF120/ O driver do teclado deve ser desenvolvido de tal modo que possibilite o usuário ajustar os parâmetros do dispositivo.

Requisitos Funcionais para Detecção de falhas

/RF130/ O sistema deve ser capaz de indicar falhas no sistema.

/RF140/ O sistema deve possibilitar que falhas do sistema sejam armazenadas na memória RAM do micro-controlador.

/RF150/ Desenvolver um componente de software no PC para capturar o log que será enviado pela porta serial do micro-controlador

/RF160/ O armazenamento do log deve ser feito de uma maneira circular na RAM do micro-controlador com o propósito de não modificar o comportamento da aplicação.

B.3 Requisitos Simulador do Motor de Indução Monofásico

As próximas seções descrevem os requisitos funcionais e não funcionais do simulador de motor de indução.

Requisitos Funcionais da Aplicação

/RF10/ O sistema deve simular o comportamento do motor monofásico.

/RF20/ O sistema deve reproduzir o sinal senoidal fornecido (pelo *soft-starter* digital) através das portas de E/S do micro-controlador.

/RF30/ O sistema deve calcular o valor de tensão, corrente e velocidade baseado no sinal senoidal fornecido nas portas de E/S do micro-controlador.

/RF40/ O valor de tensão deve ser fornecido pelo conversor digital-analógico de tal forma que os requisitos temporais da aplicação sejam atendidos.

Requisitos Não-Funcionais da Aplicação

/RNF50/ O software de controle do micro-controlador deve ser projetado de tal forma que possibilite no futuro a simulação de um motor trifásico ao sistema.

/RNF60/ O sistema deve fornecer uma boa usabilidade.

/RNF70/ O número de defeitos do sistema deve ser o menor possível.

/RNF80/ O sistema deve ser alimentado com uma bateria comum de 9V.

Requisitos Funcionais de Interface com o Usuário

/RF90/ Uma interface homem-máquina (display e teclado) deve estar presente na solução final de modo que o usuário possa interagir com o sistema.

/RF100/ O sistema deve mostrar no display a velocidade em RPM no eixo do motor monofásico.

/RF110/ O sistema deve permitir monitoramento via computador PC conectado pela porta serial.

/RF120/ O usuário deve ser capaz de solicitar do sistema a reprodução do sinal senoidal fornecido através das portas de E/S do micro-controlador.

Requisitos Funcionais para os Drivers dos Dispositivos

/RF130/ O driver do display deve ser desenvolvido com o propósito de permitir que o desenvolvedor escreva textos em qualquer posição do display.

/RF140/ O driver do teclado deve ser desenvolvido de tal modo que possibilite o usuário ajustar os parâmetros do dispositivo.

Requisitos Funcionais para Detecção de Falhas

/RF150/ O sistema deve ser capaz de indicar falhas no sistema.

/RF160/ O sistema deve possibilitar que falhas do sistema sejam armazenadas na memória RAM do micro-controlador.

/RF170/ Desenvolver um componente de software no PC para capturar o log que será enviado pela porta serial do micro-controlador

/RF180/ O armazenamento do log deve ser feito de uma maneira circular na RAM do micro-controlador com o propósito de não modificar o comportamento da aplicação.

B.4 Infra-Estrutura para Desenvolvimento dos Protótipos

A Figura B.1 mostra a infra-estrutura inicialmente planejada para o desenvolvimento dos protótipos. Conforme descrito nos processos para integração de tarefas (Seção 4.5.9) e gerenciamento da linha de produto (Seção 4.5.7), esta infra-estrutura suporta estes dois processos permitindo que o desenvolvedor integre novas tarefas do sistema e libere novas versões do produto no mercado.

Nós criamos um repositório usando a ferramenta CVS para controle de versão do código. Este repositório CVS está hospedado no servidor do sourceforge e pode ser acessado através do endereço [17]. O processo de desenvolvimento usando um ambiente de integração e teste contínuo funciona da seguinte maneira. Primeiramente o desenvolvedor baixa o código que está no repositório CVS para um diretório de trabalho local. Neste diretório local, o desenvolvedor é capaz de implementar novas funcionalidades, corrigir defeitos e realizar melhorias no sistema. Além disso, o desenvolvedor é capaz de gerar versões intermediárias do produto no seu diretório de trabalho local.

Após implementar e testar o código seguindo as atividades descritas nos processos de implementação de novas funcionalidades (Seção 4.5.8) e refatoração do código (Seção 4.5.10), o desenvolvedor disponibiliza o código no repositório CVS. Depois disso, a ferramenta *Cruise Control* procura por modificações no código fonte da aplicação. Caso a data/horário do arquivo tenha sido alterada, ou seja, caso o arquivo tenha sido modificado

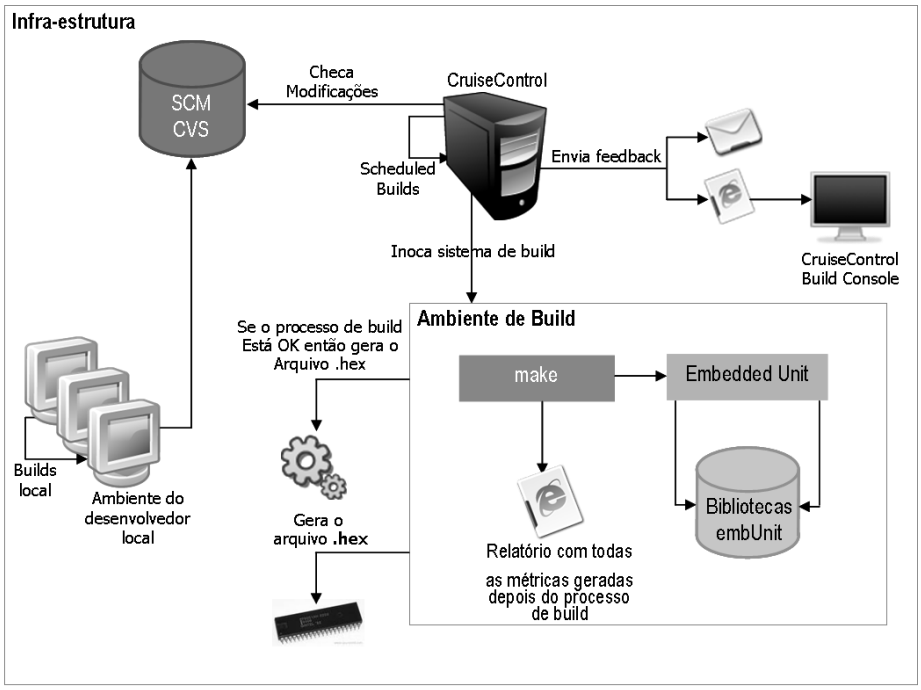

Figura B.1: Infra-Estrutura.

então o *Cruise Control* inicia o processo de build[1]. Por falta de infra-estrutura na Universidade, não foi possível instalar o aplicativo *Cruise Control* no servidor. Sendo assim, esta etapa de verificar mudanças no código não pôde ser realizada de forma automatizada.

Caso tenha ocorrido algum erro na compilação então o aplicativo *Cruise Control* envia um e-mail para o responsável do código informando que o processo de build foi interrompido. Caso contrário, o aplicativo *Cruise Control* gera o arquivo .hex e executa as ferramentas para captura de métricas e execução dos casos de teste. No final deste processo, é gerado um arquivo HTML que fornece dados da compilação, testes e métricas. A Tabela B.1 mostra as ferramentas que foram usadas para integrar e testar os componentes de software da plataforma.

[1]O processo de build significa gerar uma versão intermediária do produto

Tabela B.1: Ferramenta para integração e teste contínuo

Ferramenta	Máquina de desenvolvimento	Máquina de build
Compilador GNU C	Obrigatório	Obrigatório
CruiseControl	N/A[2]	Obrigatório
Small Device C Compiler	Obrigatório	Obrigatório
GNU make	Obrigatório	Obrigatório
Embedded Unit	Obrigatório	Obrigatório
Keil μVision3 V3.51 or +	Recomendado	N/A
Cygwin	Obrigatório[3]	Obrigatório
Cliente CVS	Obrigatório	Obrigatório
CCCC	Obrigatório	Obrigatório
Doxygen	Obrigatório	Obrigatório

Apêndice C

Descrição dos Módulos dos Protótipos

C.1 Descrição dos Módulos do Oxímetro de Pulso

As próximas subseções descrevem as funções públicas dos módulos dos drivers (componentes) e o software de aplicação desenvolvido para o projeto do oxímetro de pulso.

Sistema de Log

Este módulo fornece acesso às funcionalidades para inserir, remover e enviar curtas mensagens de textos do micro-controlador para o PC através da porta serial RS-232. Com este módulo o desenvolvedor é capaz de verificar se um dado trecho de código foi alcançado, o valor de uma variável e armazenar uma seqüência de dados da memória do micro-controlador. Além disso, este módulo usa um esquema de buffer circular para armazenar os dados na memória RAM do micro-controlador. Este esquema tende a não consumir muitos recursos do micro-controlador e manter a previsibilidade do sistema. Uma interface comum foi criada para acessar as funcionalidades deste módulo conforme mostrado na figura C.3. A seguir uma breve descrição das funções.

Funções Públicas

- void initLog(Data8)

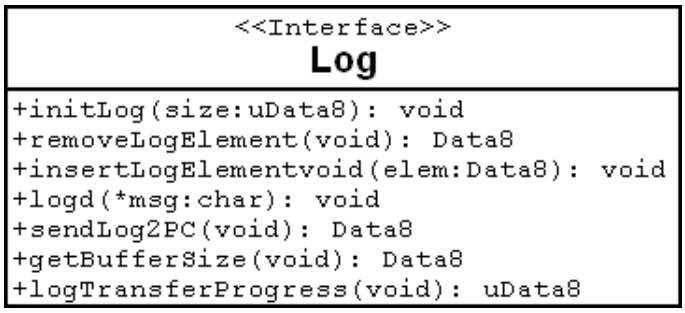

Figura C.1: Diagrama do Módulo Sistema de Log.

Esta função é usada para definir a quantidade de bytes que serão armazenados e inicializar os ponteiros de entrada e saída do buffer. O sistema de log permite que seja armazenados até 3.2 K bytes (cerca de 20% da capacidade toal de memória) de mensagens na memória do micro-controlador.

Entrada: Quantidade máxima de bytes a serem armazenados na memória RAM do micro-controlador.

Saída: Nenhuma

- Data8 removeLogElement(void)

Esta função remove o elemento apontado pelo ponteiro atual de saída do buffer. Depois de remover o elemento, o ponteiro passa para o próximo elemento do buffer.

Entrada: Nenhuma

Saída: Esta função retorna o elemento que foi removido do buffer.

- void insertLogElement(Data8)

Esta função insere o elemento apontado pelo ponteiro atual de entrada do buffer. Depois de inserir o elemento, o ponteiro passa para a próxima posição do buffer e verifica se passou do último elemento. Caso tenha passado do último elemento então ele retorna para a primeira posição do buffer.

Entrada: O elemento a ser inserido no buffer circular.

Saída: Nenhuma

- void logd(char *msg)

```
1 void printValue(void){
2     switch(getSenPos()) {
2           case HR:
3                 dlog("#HR : %d", getHR());
4                 break;
5           case SPO2:
6                 dlog("#SPO2 : %d", getHR());
7                 break;
8           case SREV:
9                 dlog("#SREV : %d", getSREV());
10                break;
11    }
12 }
```

Figura C.2: Função para Armazenar Mensagens na Memória

Esta função é usada para escrever no buffer circular. Primeiramente, a função verifica o tamanho da mensagem para que possa determinar o número de vezes que chamará a função *insertLogElement*. Depois disso, cada elemento da mensagem é obtido e colocado no buffer circular através da função *insertLogElement*.

A figura C.2 mostra um exemplo de código de aplicação.

- Data8 sendLog2PC(void)

Esta função é usada para enviar mensagens de texto para o PC via porta serial RS-232. A função primeiramente verifica o tamanho do buffer. Caso o tamanho do buffer seja igual ou menor que 0 então um erro é retornado para a aplicação. Caso contrário, a função envia cada caractere contido no buffer para o PC pela porta serial atribuindo somente o caractere ao registrador SBUF do 8051. Além disso, todas as vezes que um caractere é colocado neste registrador e enviado pela porta serial, a função *logTransferProgress* é chamada com o intuito de fornecer o progresso do envio.

Entrada: Nenhuma

Saída: Esta função retorna 0 se todos os elementos contidos no buffer foram enviados com sucesso pela porta serial. Caso contrário, o valor -1 é retornado.

- Data8 getBufferSize(void)

Esta função fornece o tamanho do buffer que contém as mensagens de texto.

Entrada: Nenhuma

Saída: Esta função retorna o tamanho do buffer em bytes.

- uData8 logTransferProgress(void)

Esta função é usada com o propósito de fornecer o status da tranferência de mensagens de texto para o PC. Toda vez que a função é chamada, um contador é incrementado e o progresso global da operação é obtido simplesmente multiplicando o valor da unidade de progresso (obtido com a função *calculateUnitProgress*) pelo valor do contador.

Entrada: Nenhuma

Saída: Esta função fornece o progresso global da operação de envio de mensagens do micro-controlador para o PC.

Módulo Sensor

Este módulo fornece acesso às funcionalidades do sensor OEM III da Nonin. Este módulo permite que o desenvolver capture os sinais de saturação de oxigênio (SpO2) e batimento cardíaco (HR), verifique a validade de um pacote enviado pelo sensor e seu status atual (p.e., conectado, desconectado, ausência de sinal) e analise a curva pletimosgráfica do sinal. Uma interface comum foi criada para acessar as funcionalidades do driver conforme mostrado na figura C.3. A seguir uma breve descrição das funções.

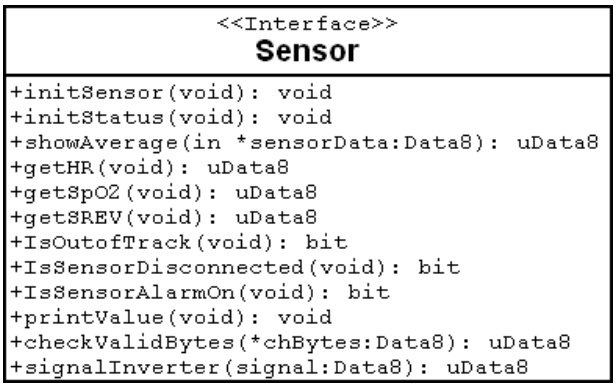

Figura C.3: Interface do Módulo Sensor.

Funções Públicas

- void initSensor(void)

Esta função inicializa a estrutura que contém os dados de saturação de oxigênio, batimento cardíaco e status do sensor.
Entrada: Nenhuma
Saída: Nenhuma

- void initStatus(void)

Esta função inicializa o byte de status do sensor que contém informações sobre conexão/desconexão do sensor, ausência de sinais de pulso e qualidade dos sinais.
Entrada: Nenhuma
Saída: Nenhuma

- uData8 showAverage(Data8 *sensorData)

Esta função calcula a média dos dados coletados do sensor. Por exemplo, o sensor fornece os dados da saturação do oxigênio três vezes em um segundo, então esta função soma os valores de SpO2 e os divide pela quantidade lida.
Entrada: Um *array* que contém os dados de SpO2 ou HR.
Saída: Retorna a média aritmética dos dados de SpO2 ou HR.

- uData8 getHR(void)

Esta função fornece o valor do batimento cardíaco no modo padrão[1].
Entrada: Nenhuma
Saída: Nenhuma

- uData8 getSpO2(void)

Esta função fornece o valor de saturação do oxigênio do sangue do paciente no modo padrão.
Entrada: Nenhuma
Saída: Nenhuma

- uData8 getSREV(void)

[1]O modo padrão atualiza o valor de SpO2 ou HR a cada batimento do pulso

Esta função fornece a versão do firmware do módulo OEM III.
Entrada: Nenhuma
Saída: Nenhuma

- bit IsOutofTrack(void)

Esta função verifica se existem sinais de saturação de oxigênio ou batimento cardícado disponíveis no sensor.
Entrada: Nenhuma
Saída: Caso não exista sinal de SpO2 ou HR então esta função retorna verdadeiro (1). Caso exista sinal então a função retorna falso (0).

- bit IsSensorDisconnected(void)

Esta função verifica se o sensor está conectado ao módulo OEM III ou se o sensor está em condições de funcionamento.
Entrada: Nenhuma
Saída: Caso o sensor não esteja conectado ao módulo ou não esteja em condições de funcionamento, então esta função retorna verdadeiro (1). Caso o sensor esteja conectado ao módulo ou em condições de funcionamento então a função retorna falso (0).
A figura C.4 mostra um exemplo de código de aplicação das funções *IsOutofTrack* e *IsSensorDisconnected*.

- bit IsSensorAlarmOn(void)

Esta função verifica se o sensor está enviando dados confiáveis para o módulo OEM III.
Entrada: Nenhuma
Saída: Caso o sensor não esteja enviando dados confiáveis para o módulo então esta função retorna verdadeiro (1). Caso o sensor esteja enviando dados confiáveis para o módulo, então a função retorna falso (0).

- void printValue(void)

Esta função envia um sinal para a camada de aplicação informando que os dados de saturação de oxigênio e batimento cardíaco sejam lidos.

```
1    Data8 checkError(void){
2    Data8 err = 0;
3    if(IsSensorDisconnected()){
4         lcd_printf("Sensordiconnected", 1, 1);
5         err = −1;
6    }
7    else if(IsOutofTrack()){
8         lcd_printf("OutOfTrack", 1, 1);
9         err = −1;
10   }
11   return err;
12 }
```

Figura C.4: Função para Checar Erros de Aquisição de Dados

Entrada: Nenhuma

Saída: Nenhuma

- uData8 signalInverter(Data8 signal)

Esta função checa o sinal do byte enviado pelo módulo OEM III. Caso o byte tem um valor negativo então o sinal do byte é invertido.

Entrada: Deve ser passado o valor do byte enviado pelo módulo OEM III.

Saída: Caso o sinal do byte seja negativo então é retornado o módulo do byte. Caso contrário, a função retorna apenas o valor passado na sua chamada.

- uData8 checkValidBytes(Data8 *chBytes)

Esta função tem o propósito de verificar se um conjunto de bytes enviado pelo módulo OEM III é válido. Esta verificação é realizada através da soma dos quatro primeiros bytes que compõe o *frame* (o módulo OEM III envia 75 frames a cada segundo para a plataforma de desenvolvimento e cada frame contém 5 bytes). Caso o byte enviado pelo módulo tenha um valor negativo então a função *signalInverter* inverte o sinal do byte para que o mesmo seja somado e armazenado na variável *chksum*. Após ter somado os quatro primeiro bytes do frame, verifica-se se o resto da divisão do valor contido na variável *chksum* pelo valor 256 é igual ao valor do quinto byte. Caso seja então o conjunto de bytes enviado pelo módulo é válido.

Entrada: Deve-se passar na chamada desta função o ponteiro para o *frame* enviado pelo módulo OEM III.

Saída: Caso o frame seja válido então 0 é retornado. Caso contrário, -1 é retornado.

Módulo Teclado

Este módulo permite que o desenvolvedor identifique qual tecla foi pressionada durante o uso do oxímetro de pulso.Uma interface comum foi criada para acessar a funcionalidade do driver conforme mostrado na figura C.5. A seguir uma breve descrição da função.

Figura C.5: Interface do Módulo Teclado.

Funções Públicas

- Data8 checkPressedButton(uData8)

Esta função verifica se uma dada tecla foi pressionada pelo usuário do oxímetro.

Entrada: O valor da porta de E/S onde está conectado o teclado deve ser passado como parâmetro para esta função.

Saída: Esta função retorna o valor da tecla que foi pressionada pelo usuário. Caso nenhuma tecla tenha sido pressionada, então a função retorna -1.

A figura C.6 mostra um exemplo de código de aplicação.

Módulo Display

Este módulo permite que o desenvolvedor manipule mensagens no LCD 16x2. Uma interface comum foi criada para acessar a funcionalidade do driver conforme mostrado na figura C.7. A seguir uma breve descrição da função.

Funções Públicas

- void lcd_printf(const char *sPtr, int line, int column)

```
1 uData8 keys = 0x00; /*nokeypressed*/
2 keys = P1;
3 pressedkey = checkPressedButton(keys);
4 if(pressedkey>0) {
5     switch(pressedkey) {
6           ⋮
7     }
8 }
```

Figura C.6: Função para Detectar Tecla Pressionada

Figura C.7: Interface do Módulo Display.

Esta função escreve algum texto em um display 16x2 de acordo com a linha e coluna passada como parâmetro na função.

Entrada: O desenvolver deve passar o texto a ser escrito no display e a posição (linha e coluna).

Saída: Nenhuma

- void lcd_clean(void)

Esta função limpa mensagens escritas no LCD 16x2 durante a execução de uma aplicação.

Entrada: Nenhuma

Saída: Nenhuma

Módulo Serial

Este módulo fornece acesso a comunicação serial RS-232 disponível na plataforma de desenvolvimento. O periférico comunicador serial permite uma comunicação bidirecional entre dois dispositivos e transmite/recebe um byte, bit por bit, em seqüência pré-estabelecida e pré-programada. Sendo assim, este módulo permite configurar a taxa de comunicação

e usar as rotinas de interrupção da serial para envio e recebimento de dados. Uma interface comum foi criada para acessar as funcionalidades do driver conforme mostrado na figura C.8. A seguir uma breve descrição das funções.

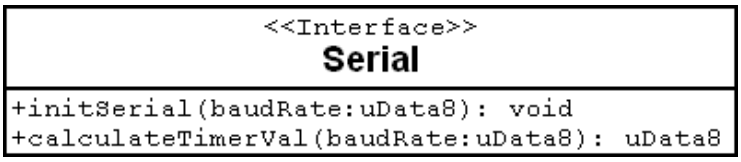

Figura C.8: Interface do Módulo Serial.

Funções Públicas

- void initSerial(uData8 baudRate)

Esta função tem o propósito de configurar a porta serial para a taxa de transmissão passada na chamada da função.

Entrada: A taxa de transmissão (1200, 2400, 9600, 19200) em bps deve ser passada na chamada da função.

Saída: Nenhuma

- uData8 calculateTimerVal(uData8 baudRate)

Esta função é usada para calcular o valor do registrador TH0 do 8051.

Entrada: A taxa de transmissão (1200, 2400, 9600, 19200) em bps deve ser passada na chamada da função.

Saída: Esta função retorna o valor do registrador. Caso ocorra algum erro no cálculo então o valor -1 é retornado.

Módulo Temporizador

Este módulo fornece acesso às funções para configurar o temporizador da plataforma de desenvolvimento. O temporizador, também conhecido como timer/counter, é um periférico interno da plataforma e nada mais é do que um grupo de flip-flops em arranjo de "divisor por dois" que é acionado pelo clock do micro-controlador (o clock da plataforma de desenvolvimento é de 11.059MHZ). Uma interface comum foi criada para acessar as

funcionalidades do driver conforme mostrado na figura C.9. A seguir uma breve descrição das funções.

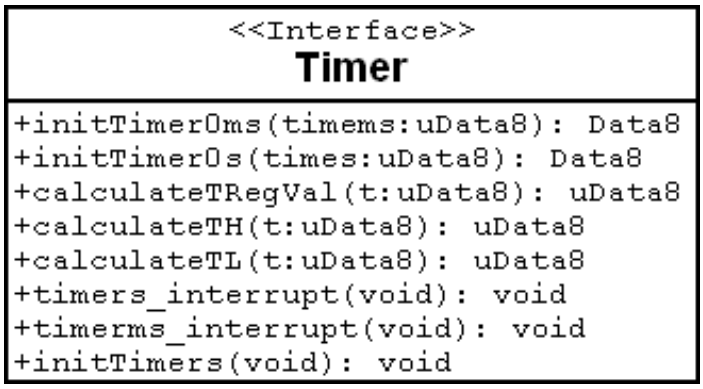

Figura C.9: Interface do Módulo Temporizador.

Funções Públicas

- void initTimers(void)

Esta função no início do programa e serve para inicializar as variáveis do módulo temporizador.

Entrada: Nenhuma

Saída: Nenhuma

- Data8 initTimer0ms(uData8 timerms)

Função usada para configurar o temporizador 0 do 8051. O processo de configuração consiste essencialmente em *(i)* desabilitar todas as interrupções, *(ii)* parar o temporizador 0, *(iii)* configurar o valor do registrador TMOD, *(iv)* carregar o valor dos registradores TH0 e TL0 com o conteúdo para realizar a contagem, *(v)* habilitar o contador e interrupção do temporizador 0 e *(vi)* finalmente habilitar todas as interrupções.

Entrada: O valor do disparo do temporizador 0 em milisegundos.

Saída: Esta função retorna o valor em milisegundos que foi usado para ajustar os registradores do temporizador 0. Caso tenha ocorrido algum erro então -1 é retornado.

- Data8 initTimer0s(uData8 timers)

Esta função tem como objetivo configurar o temporizador para ser disparado de acordo com o valor passado na chamada da função.

Entrada: O valor do tempo em segundos deve ser passado na chamada da função.

Saída: Esta função retorna o valor em segundos que foi usado para ajustar o temporizador 0. Caso tenha ocorrido algum erro então -1 é retornado.

- uData8 calculateTRegVal(uData8 t)

Função usada para calcular o valor a ser carregado no registrador do temporizador. Este valor é calculado subtraindo o valor máximo que o contador suporta (neste caso é $2^{16} = 65535$) pelo tempo desejado que o temporizador dispare. Esta função é chamada pelas funções *calculateTH* e *calculateTL*.

Entrada: O tempo para disparar o temporizador deve ser passado em milisegundos na chamada da função.

Saída: Nenhuma

- uData8 calculateTH(uData8 t)

Esta função é usada para calcular o byte mais significativo do registrador do temporizador. Este valor é calculado realizando uma máscara $0xFF00$ com o retorno da função *calculateTRegVal* e em seguida deslocando 8 bits para direita.

Entrada: O tempo para disparar o temporizador deve ser passado em milisegundos na chamada da função.

Saída: Nenhuma

- uData8 calculateTL(uData8 t)

Esta função é usada para calcular o byte menos significativo do registrador do temporizador. Este valor é calculado realizando uma máscara $0x00FF$ com o retorno da função *calculateTRegVal*.

Entrada: O tempo para disparar o temporizador deve ser passado em milisegundos na chamada da função.

Saída: Nenhuma

- void timersinterrupt(void)

Esta função envia um sinal para a camada de aplicação informando o estouro do contador do temporizador. Este sinal é enviado de acordo com o valor passado na chamada da função *initTimer0s*.

Entrada: Nenhuma

Saída: Nenhuma

- void timermsinterrupt(void)

Esta função envia um sinal para a camada de aplicação informando o estouro do contador do temporizador. Este sinal é enviado de acordo com o valor passado na chamada da função *initTimer0ms*.

Entrada: Nenhuma

Saída: Nenhuma

Módulo Lista de Comandos

Este módulo fornece uma interface com o usuário para acessar as funcionalidades do oxímetro de pulso. Este módulo fornece basicamente 5 opções de comandos que incluem: *(i)* ajustar o tempo de amostragem dos dados do sensor, *(ii)* habilitar/desabilitar captura dos dados na memória RAM do micro-controlador e *(iii)* a visualização dos dados de SpO2 e HR na tela do oxímetro de pulso. Além disso, este módulo possibilita o usuário de iniciar uma comunicação com PC com o intuito de transferir os dados armazenados na memória. Uma interface comum foi criada para acessar as funcionalidades do driver conforme mostrado na figura C.10. A seguir uma breve descrição das funções.

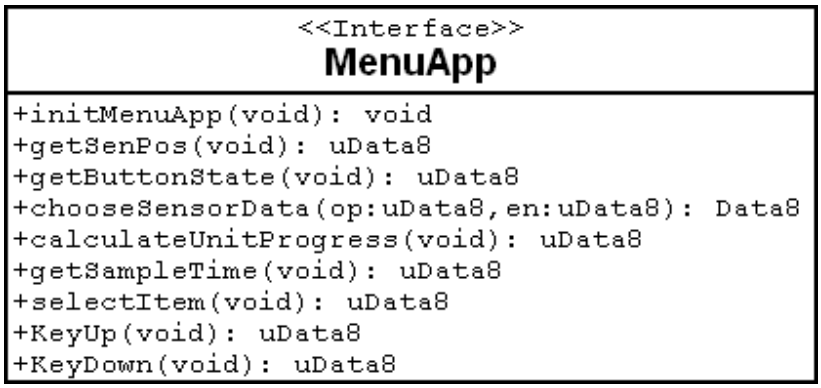

Figura C.10: Diagrama do Módulo Lista de Comandos.

Funções Públicas

- void initMenuApp(void)

Esta função é usada para incializar as variáveis internas do módulo Lista de Comandos. Estas variáveis incluem, por exemplo, o estado da lista, a estrutura que contém os dados de SpO2 e HR e algumas variáveis de controle.

Entrada: Nenhuma

Saída: Nenhuma

- uData8 getSenPos(void)

Esta função é usada para obter os dados do sensor selecionados pelo usuário para serem mostrados no LCD 16x2. Esta função consiste essencialmente de uma estrutura de seleção *switch* que verifica em um dado momento qual valor de variável deve ser mostrada na tela.

Entrada: Nenhuma

Saída: Esta função retorna o valor da variável a ser exibida na tela do oxímetro. Caso ocorra um erro então -1 é retornado.

- uData8 getButtonState(void)

Esta função é usada para saber o estado atual do oxímetro de pulso.

Entrada: Nenhuma

Saída: Esta função retorna o estado atual do sistema.

- Data8 chooseSensorData(uData8 op, uData8 en)

Esta função é usada para armazenar os dados do sensor a serem exibidos no display 16x2. Esta função verifica primeiramente se o tipo de dado a ser armazenado pertence realmente a classe de dados do sensor. Caso pertença então o dado é armazenado em uma estrutura de dados para ser posteriormente obtido pela função *getSenPos*.

Entrada: A identificação do dado do sensor e conteúdo da variável sob observação.

Saída: O dado armazenado na estrutura de dados. Caso ocorra um erro então -1 é retornado.

- uData8 calculateUnitProgress(void)

Esta função é usada para calcular o valor da unidade de progresso. Este cálculo é baseado na quantidade de elementos do buffer de dados coletados do sensor. O valor é calculado através da divisão de 100% pelo tamanho do buffer.

Entrada: Nenhuma

Saída: Esta função retorna o valor da unidade de progresso. Caso ocorra um erro no cálculo então -1 é retornado.

- uData8 selectItem(void)

Esta função tem o objetivo de selecionar o item (p.e., HR, SpO2, Log) a ser mostrado para o usuário. Esta função consiste essencialmente em uma estrutura de seleção *switch* que verifica o estado atual do sistema e determina o que deve ser feito para cada estado.

Entrada: Nenhuma

Saída: Esta função retorna o próximo estado do sistema.

- uData8 KeyUp(void)

Esta função é usada para incrementar as variáveis da lista de comandos do oxímetro de pulso. Esta função consiste essencialmente em uma estrutura de seleção *switch* que verifica o estado atual do sistema e incrementa as variáveis dos itens em observação.

Entrada: Nenhuma

Saída: Esta função retorna o valor da variável que foi incrementada. Caso ocorra um erro então -1 é retornado.

- uData8 KeyDown(void)

Esta função é usada para decrementar a lista de comandos do oxímetro de pulso. Esta função consiste essencialmente em uma estrutura de seleção *switch* que verifica o estado atual do sistema e decrementa as variáveis dos itens em observação.

Entrada: Nenhuma

Saída: Esta função retorna o valor da variável que foi decrementada. Caso ocorra um erro então -1 é retornado.

C.2 Descrição dos Módulos do *Soft-Starter* Digital

As próximas subseções descrevem as funções públicas dos módulos dos drivers (componentes) e o software de aplicação desenvolvido para o projeto do *soft-starter* digital.

Gerador PWM

Este módulo possibilita gerar os sinais PWM que são aplicados nas chaves do circuito de controle do motor. Estes sinais são produzidos em quatro diferentes pinos do microcontrolador, convencionados como Q_1, Q_2, Q_3 e Q_4. Este módulo produz o mesmo sinal para os pinos Q_1 e Q_2, porém invertidos. Quando um pino está em nível lógico alto o outro está em nível lógico baixo. Para gerar os sinais nos pinos Q_3 e Q_4, este módulo gera o sinal no pino Q_3 depois de um ângulo α com relação ao pino Q_2. O mesmo acontece com o pino Q_1 e Q_4. É importante salientar que os valores usados para ajustar o temporizador do 8051 são calculados em tempo de compilação com o intuito de otimizar o tamanho do código (conhecimento a *priori* do ambiente). Isto significa que de posse do valor de tensão do motor de indução monofásico, nós geramos os valores para serem carregados no temporizador para uma faixa de tensão que varia de 50 a 127 volts. Uma interface comum foi criada para acessar as funcionalidades deste módulo conforme mostrado na figura C.11. A seguir uma breve descrição das funções.

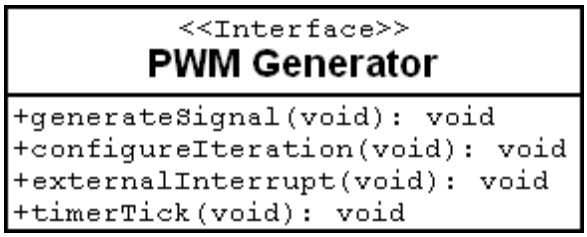

Figura C.11: Diagrama do Módulo Gerador PWM.

Funções Públicas

- void externalInterrupt(void)

Esta função trata da interrupção externa 1 que é responsável pela comunicação com o simulador do motor de indução. Sendo assim, esta função tem como objetivo mudar a flag para nível lógico alto e ler o valor de tensão a partir do conversor A/D. Esta flag

(ou sinal) é usada para a comunicação entre o *soft-starter digital* e simulador do motor de indução. A implementação deste mecanismo evita que o simulador do motor fique checando periodicamente o valor dos pinos Q_1, Q_2, Q_3 e Q_4. Sendo assim, toda vez que o simulador do motor receber o sinal PWM e calcular o valor de tensão RMS, o mesmo gera um sinal para o soft-starter digital informando do recebimento do sinal. Deste modo, o soft-starter digital verifica o valor da tensão nos terminais do motor e calcula o novo sinal PWM a ser enviado para o simulador do motor.

- void timerTick(void)

Esta função tem como objetivo configurar o temporizador para ser disparado a cada 100μs com o intuito de atender as restrições temporais da aplicação. Para cada disparo do temporizador, os contadores para os pinos Q_1 e Q_3 são incrementados com o intuito de controlar a geração dos sinais PWM.

- void generateSignal(void)

Esta função usa os valores do contador para comparar com os respectivos períodos e gerar os sinais dos pinos de controle. Cada sinal enviado para o pino de controle do motor fica em nível lógico alto por um tempo T/2 e nível lógico baixo por um tempo T/2. Q_4 é deslocado de Q_1 pelo ângulo α assim como Q_2 em relação a Q_3. Os valores de α estão tabelados no array *tonTable*. Além disso, esta tabela é gerada a partir das funções disponíveis no arquivo *table_gen.c* em tempo de compilação . A figura C.12 mostra o código fonte responsável pela geração dos sinais PWM.

Conforme mostrado na figura C.12, a linha 2 verifica se o contador do pino Q_1 excedeu o tempo permitido para o sinal ficar em nível lógico alto. Caso tenha excedido, então a variável que habilita Q_3 recebe o valor da constante simbólica que representa o tempo de 100μs para o disparo do temporizador. A variável MAX_T_2 representa a largura do pulso do sinal PWM. Com isso, a linha 3 verifica se o valor do contador de Q_1 já excedeu o tempo necessário para ficar em nível lógico alto. Caso tenha excedido, o contador de Q_1 é zerado e o sinal dos pinos de controle Q_1 e Q_2 são invertidos. O mesmo procedimento acontece para as linhas 11 e 14. A função *INTERRUPT* é usada nas linhas 9 e 15 para informar o simulador do motor que os sinais de controle mudaram.

```
1 void generateSignal(void) {
2     if (counterQ1 >= tonTable[(ton * 2) + 1])
3         q3Enable = TICK;
4     if (counterQ1 >= MAX_T_2) {
5         counterQ1 = 0;
6         Q1 = !Q1;
7         Q2 = !Q2;
8         if (increasing)
9             INTERRUPT(3);
10    }
11    if (counterQ3 >= MAX_T_2) {
12        counterQ3 = 0;
13        Q3 = !Q3;
14        Q4 = !Q4;
15        INTERRUPT(3);
16    } 17 }
```

Figura C.12: Código para gerar os sinais PWM

- void configureIteration(void)

Esta função tem como objetivo configurar a geração de cada sinal aplicado nos pinos Q_1, Q_2, Q_3 e Q_4 do circuito de controle do motor. Para cada iteração, o tempo T_{on} em que sinal fica no nível lógico alto é reconfigurado. No entanto, o período do sinal é sempre o mesmo, ou seja, 16.666 ms (ou 60 Hz). O valor de T_{on} é obtido a partir da tabela *tonTable* que foi gerada em tempo de compilaçao. O valor atual de T_{on} é indexada na variável *ton*.

Conversor A/D

Este módulo tem como objetivo ler o valor de tensão nos terminais do motor de indução monofásico. Este valor de tensão é lido pelo conversor A/D e é convertido para um valor que pode chegar até 127 volts. A camada de aplicação realiza uma leitura já convertida do conversor A/D através da função *readFromAD*. Este driver acessa o hardware do conversor do A/D (PCF8591) através do barramento I^2C conforme descrito na seção de arquitetura do sistema (6.2.2). Sendo assim, uma interface comum foi criada para acessar as funcionalidades do chip PCF8591 conforme mostrado na figura C.13. A seguir uma breve descrição das funções.

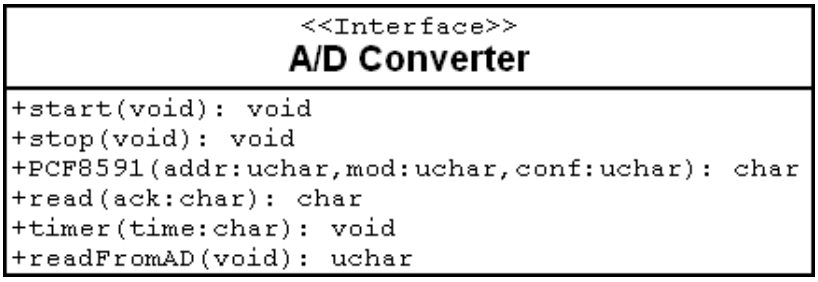

Figura C.13: Diagrama do Módulo Conversor A/D.

- void start(void)

Para que seja iniciada uma comunicação entre o micro-controlador (mestre) e conversor A/D (escravo), deve ser enviado uma seqüência de *start* com o intuito de receber os valores lidos pelo conversor. Sendo assim, esta função envia um nível lógico alto para os pinos de dados e clock (SDA e SCL respectivamente) e logo em seguida muda o pino SDA para nível lógico baixo. Depois disso, espera-se por 5 μs até mudar o nível lógico do pino SCL para baixo.

- void stop(void)

Para que seja encerrada a comunicação entre o micro-controlador e conversor A/D via barramento I^2C, deve ser enviado um comando *stop*. Sendo assim, esta função envia um nível lógico alto para o pino SCL e um nível lógico baixo para o pino SDA. Depois disso, espera-se por 5 μs até mudar o nível lógico do pino SDA para alto.

- char PCF8591(unsigned char addr, unsigned char mod, unsigned char conf)

Esta função tem como objetivo configurar o chip do PCF8591 para realizar leituras no canal do conversor A/D.

Entrada: Esta função deve ser chamada passando como parâmetro o endereço (*addr*) do chip PCF8591 no barramento I^2C que é 0x02H, o modo (*mod*) que pode ser de escrita(1) ou leitura(0) e o número do registro (*conf*) que pode ser 0x04H (para o D/A) ou 0x00H (para o A/D).

Saída: Retorna o valor do status para o controle da função.

- char read(char ack)

Esta função tem como propósito realizar a leitura do valor digitalizado da tensão nos terminais do motor. Esta leitura é realizada através do barramento I^2C da plataforma de desenvolvimento. Com isso, deve-se configurar o barramento para leitura através da operação "(addr << 1)+1" o qual indica que o endereço do barramento deve ser deslocado de 1 posição para a esquerda e adicionado 1 no bit menos significativo.

Entrada: O valor 1 deve ser passado como parâmetro de entrada para esta função com o intuito de habilitar a leitura de dados no barramento I^2C.

Saída: Retorna o valor lido do conversor A/D como um dado de 8 bits.

- void timer(char times)

Esta função é responsável por configurar o temporizador para disparar a cada 60 *m*s. Além disso, esta função tem uma estrutura de repetição *while* que define o número de vezes que a função deve aguardar em cada disparo do temporizador. Isto significa que a função levará (número de vezes×60 ms) de tempo de processamento antes de liberar o processamento para outras funções do sistema.

Entrada: O parâmetro *times* determina o número de vezes que a função deve aguardar antes de ser liberada do processador.

Saída: Nenhuma.

- unsigned char readFromAD(void)

Esta função tem como objetivo realizar a leitura do valor de tensão do conversor D/A através da função *read*. O valor lido do conversor D/A é então convertido para o valor real da tensão que pode chegar até 127 volts. Este valor calculado pela função *readFromAD* é usado pelo módulo do gerador PWM com o intuito de calcular os sinais de controle nos pinos Q_1, Q_2, Q_3 e Q_4.

Entrada: Nenhuma

Saída: O valor lido do conversor A/D é convertido para à faixa de tensão do motor de indução monofásico.

O arquivo *generateTable.c* fornece funções para gerar a tabela *lookup* com informações sobre os valores de T_{on} para cada valor de tensão do motor. Esta tabela foi desenvolvida com o propósito de melhorar o desempenho do sistema e precisão dos valores calculados.

```
1 double calculateTonMicro(double E, double Vrms, double T) {
2     double d;
3     double t_2;
4     double t_on;
5     d = Vrms / E;
6     d = d × d;
7     t_2 = T / 2.0;
8     t_on = d × t_2;
9     t_on = t_on × 1000000.0;
10    return t_on;
11 }
```

Figura C.14: Código para gerar os valores de T_{on}

A figura C.14 apresenta a função responsável por calcular os valores de T_{on} para diferentes níveis de tensão do sinal PWM.

Como pode ser observado neste código, usamos a seguinte equação para calcular o valor de T_{on} que foi deduzida na referência [2]:

$$Ton = (Vrms/E)^2 \times T/2 \tag{C.1}$$

A equação C.1 está representada nas linhas 5 a 7 da figura C.14. Além disso, a linha 8 multiplica o valor de T_{on} pelo ciclo de trabalho representado pela variável *d*. Logo em seguida, na linha 9, o valor de T_{on} é multiplicado por 1×10^6 para transformar a unidade em μs. Como parâmetro de entrada para a função *calculateTonMicro*, deve ser passado a tensão nominal do motor (E), o valor da tensão RMS que representará a largura do pulso do sinal (V_{rms}) e o período do sinal (T). A função *calculateTonMicro* retorna o valor de T_{on} em μs. A próxima seção descreve as características e arquitetura do simulador de motor de indução monofásico.

C.3 Descrição dos Módulos do Motor de Indução

As próximas subseções descrevem as funções públicas dos módulos dos drivers (componentes) e o software de aplicação desenvolvido para o projeto do motor de indução monofásico.

Tratador do PWM

Este módulo tem como propósito capturar o valor de tensão RMS representado pelo sinal PWM que está sendo aplicado ao circuito de controle do motor. Este módulo basicamente monitora os pinos Q_1, Q_2, Q_3 e Q_4 do micro-controlador. O valor da tensão RMS é calculada por este módulo de acordo com os períodos T_{on} (tempo em que o sinal passa em nível lógico alto) e T (tempo para repetição do sinal). Este valor é posteriormente usado para ser aplicado nos terminais do motor de indução monofásico que é representado por um modelo matemático no micro-controlador. Uma interface comum foi criada para acessar as funcionalidades deste módulo conforme mostrado na figura C.15. A seguir uma breve descrição das funções.

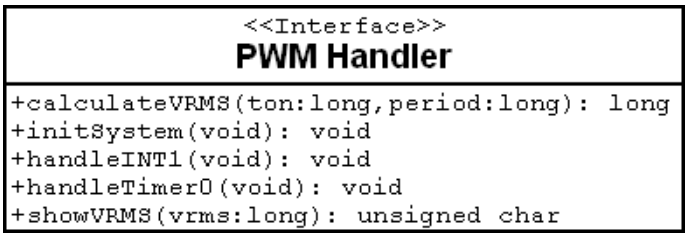

Figura C.15: Diagrama do Módulo Tratador PWM.

- void handleTimer0(void)

Esta função basicamente configura o temporizador 0 para disparar a cada 65 ms. O valor do temporizador configurado por esta função é usado para mensurar os períodos T_{on} e T do sinal PWM.

- void handleINT1(void)

Esta função calcula o valor de T_{on} e T do sinal PWM aplicado nos pinos Q_1, Q_2, Q_3 e Q_4. Esta função verifica primeiramente se T_{on} é igual a T/2. Isto ocorre na condição de contorno do sinal PWM, ou seja é o valor máximo que T_{on} pode assumir em um semi-ciclo positivo ou negativo. Caso o valor de T_{on} não esteja na condição de contorno então esta função verifica o sinal aplicado nos pinos Q_1 e Q_4. Sendo assim, a função monitora o ângulo de disparo de Q_4 em relação a Q_1 e armazena este valor em um *array* de inteiros. Depois disso, os valores contidos neste *array* serão usados para o cálculo da tensão RMS pela função *calculateVRMS*.

- void initSystem(void)

Esta função é responsável por inicializar o *array* que contém informações dos períodos T_{on} e T assim como a inicilização dos registradores do temporizador. Além disso, esta função inicializa com nível lógico baixo o pino do micro-controlador responsável por estabelecer a comunicação com o *soft-starter* digital.

- long calculateVRMS(long ton, long period)

Entrada: O valor em que o pulso fica em nível lógico alto e o período do sinal PWM.
Saída: O valor de tensão RMS representado pelo sinal PWM.

Esta função verifica se o *array* que contém informações dos períodos T_{on} e T está preenchido. Caso esteja, então esta função calcula o valor da tensão RMS e depois inicializa o *array* de inteiros para uma nova captura de dados. Antes de calcular o valor de tensão RMS, esta função calcula o ciclo de trabalho do sinal PWM da seguinte maneira como descrito em [2]:

$$d = T_{on}/T \tag{C.2}$$

Depois de encontrar o ciclo de trabalho do sinal PWM, finalmente esta função calcula o valor da tensão RMS usando a seguinte equação como descrito em [2]:

$$V_{rms} = E \times \sqrt{2 \times d} \tag{C.3}$$

Onde E é o máximo valor de tensão representado pelo sinal PWM (este valor representa a tensão nominal do motor de indução monofásico) e d representa o ciclo de trabalho. A Figura C.16 apresenta o código da função *calculateVRMS* responsável por calcular o valor de tensão RMS. Como mostrado nesta figura, as linhas de 2 e 3 representam as equações C.2 e C.3.

- unsigned char showVRMS(long vrms)

Esta função imprime o valor da tensão RMS no display, escreve este valor no conversor D/A e finalmente envia um sinal para o soft-starter digital com o propósito de iniciar um novo ciclo do sinal PWM.

```
1 calculateVRMS(long ton, long period) {
2     cycle = ((ton * 10000)/period);
3     factorcycle = (int) RAIZQUAD(2.0 * cycle);
4     return ((long) (127 * factorcycle)/100);
5 }
```

Figura C.16: Função para calcular o valor V_{rms}

```
1 unsigned char showVRMS(long vrms) {
2     if ((vrms > 0) && (vrms <= 127)) {
3     write("RMS: %d", vrms);
4     PRINTAD((int) vrms);
5     for(i=0; i < 5000; i++);
6     P20 = 1;
7     for(i=0; i < 10; i++);
8     P20 = 0;
9     return(0); 10
11 return(-1);
12 }
```

Figura C.17: Função para visualizar o valor V_{rms}

Entrada: Valor de tensão RMS que foi calculado pela função *calculateVRMS*.

Saída: Esta função retorna 0 se o valor de tensão RMS é escrito no display com sucesso. Caso contrário, o valor -1 é retornado.

A figura C.17 mostra o código da função *showVRMS*. A linha 3 escreve o valor da tensão RMS no display do simulador do motor. A linha 4 é responsável por escrever o valor de tensão RMS no conversor D/A. Este valor será lido pelo *soft-starter* digital e será usado como base para a geração do novo ciclo de pulsos PWM. As linhas 6 e 8 enviam uma sinal para o *soft-starter* digital informando que o sinal atual nos pinos do micro-controlador foi capturado, o valor de tensão RMS já foi calculado e escrito no conversor D/A.

Conversor D/A

Este módulo tem como objetivo fornecer o valor de tensão nos terminais do motor de indução monofásico para o *soft-starter* digital. Este valor de tensão que varia de 50 a 127 volts é convertido para um valor de tensão que é representado de 0 a 5 volts no canal do conversor D/A. A camada de aplicação realiza a escrita do valor de tensão no canal do conversor D/A através da função *write2DA*. Este driver acessa o hardware do conversor do D/A (PCF8591) através do barramento I^2C conforme descrito na seção de arquitetura do sistema (6.2.2). Sendo assim, uma interface comum foi criada para acessar as funcionalidades do chip PCF8591 conforme mostrado na figura C.18. A seguir uma breve descrição das funções.

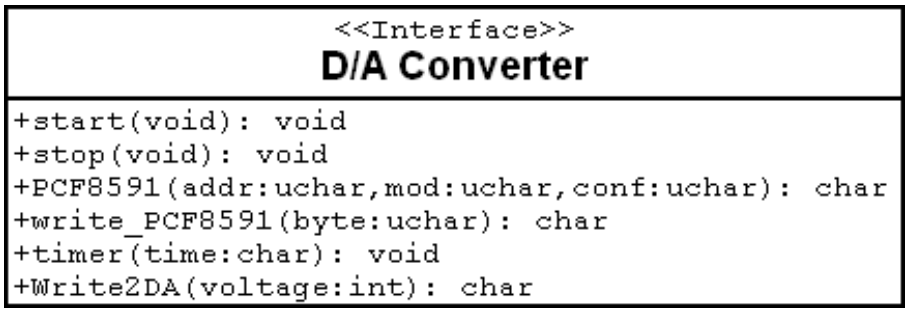

Figura C.18: Diagrama do Módulo Conversor D/A.

Os métodos *start, stop, PCF8591* e *timer* são os mesmos descritos na seção C.2 referente ao componente do conversor A/D.

- char write_PCF8591(unsigned char byte)

Esta função é responsável por escrever o valor de tensão entre 0 e 5 volts no canal do conversor D/A. Isto é realizado enviando uma seqüência de 0's e 1's no pino de dados SDA. No final desta operação, o pino de dados SDA recebe um nível lógico alto e o pino de clock SDL recebe um nível lógico baixo. O estado atual do pino SDA é atribuído a uma variável de controle e o valor desta variável é retornada para a função que chamou *write_PCF8591*.

Entrada: Valor que deve ser escrito no canal do conversor D/A. Este valor pode variar de 0 a 255, o valor máximo representa a tensão de 5 volts no canal do conversor.

Saída: Envia um byte de controle para reconhecimento do conversor D/A.

- char write2DA(int voltageConverted)

Esta função é chamada pelo software de aplicação com o intuito de escrever o valor de tensão que varia de 50 a 127 volts no canal do conversor D/A. Sendo assim, esta função converte o valor que está no intervalo de 50 à 127 volts para o intervalo de 0 à 5 volts. Além disso, a função *PCF8591* é chamada com o propósito de configurar o chip do conversor para funcionar no modo escrita. Depois disso, a função *write_PCF8591* pode escrever realmente o valor de tensão convertido no intervalo de 0 à 5 volts no canal do conversor D/A.

Entrada: O valor de tensão no intervalo de 50 a 127 volts.

Saída: Esta função retorna 0 se o valor de tensão foi escrito com sucesso no canal do conversor D/A. Caso contrário, o valor -1 é retornado.

Apêndice D

Técnicas de Otimização de Código

Este apêndice tem como objetivo apresentar apenas um pequeno conjunto das técnicas de otimização que foram mais usadas nos experimentos desta dissertação de mestrado[1]. Como mencionado no processo para otimização do sistema 4.5.11, a primeira e a mais importante parte na otimização de um programa é descobrir onde otimizar. Geralmente são pequenas partes do código que consomem muita memória ou processamento.

Com o propósito de reduzir partes do código que consomem memória/tempo de execução, nós utilizamos: *(i)* as técnicas de execuções condicionais (if, else, switch), *(ii)* técnicas de desvio de fluxo (loops, chamadas de funções), *(iii)* técnicas de otimização aritmética, e *(iv)* técnicas de manipulação de variáveis.

A técnica de execuções condicionais conhecida como *Binary Breakdown* aconselha quebrar cadeias de *if-else* em estilo binário, agilizando assim a sua pesquisa [23]. A figura D.1 mostra um exemplo de aplicação da técnica *Binary Breakdown*. Este trecho de código pertence a função *collectData* do protótipo do oxímetro de pulso. Como pode ser observado na linha 6, o *else* fica localizado logo após o fechamento do segundo *if* na linha 2.

Uma outra técnica de execuções condicionais conhecida como *Lazy evaluation exploitation* aconselha que em um if do tipo (algo && algumacoisa), é bom que a primeira parte do AND seja uma expressão que dê uma resposta falsa ou que seja mais fácil de resolver [23]. Assim a segunda parte, mais trabalhosa, será executada somente quando for necessária e menos vezes possível. A linha 6 da figura D.1 apresenta o uso desta técnica. Neste caso, a

[1]Para uma lista completa das técnicas de otimização de código, refira-se a [23]

```
1 if (sensorByte == 1 || flag2 == TRUE) {
2     if (flag2 == FALSE) {
3           fillArrays(sensorByte, contPos);
4           contPos++;
5           flag2=TRUE;
6     } else if (flag1 == TRUE || (SYNC&sensorByte) == TRUE) {
7           fillArrays(sensorByte, contPos);
8           contPos++;
9           flag1=TRUE;
10    }
11 }
```

Figura D.1: Técnica *Binary Breakdown*

comparação entre dois valores exige menos instruções do processador quando comparado com segunda condicional que é representada pela operação & e a comparação.

A técnica de execuções condicionais para *switch* aconselha que ao invés de escrever cascatas de *if-else* é melhor utilizar um *switch*. Além disso, é interessante colocar *cases* mais utilizados primeiro. A figura D.2 mostra um trecho de código da função *fillArrays* do oxímetro de pulso que utiliza esta técnica. Todo pacote de dados que é transmitido pelo sensor através da porta serial possui um byte de status que fornece informação da qualidade do sinal, representação da amplitude e assim por diante. Sendo assim, o case *posStatus* é o mais executado dentro *switch* da função *fillArrays*. Depois do case *posStatus*, os demais possuem a mesma freqüência de ocorrência, ou seja, eles aparecem em apenas um pacote de dados de 75 que são enviados pelo sensor.

A técnica de desvio de fluxo usada para condição de parada de loops consiste em implementar contadores de loops decrementando [23]. Para verificar a parada é feito somente uma comparação com zero, se contador igual a zero, então pára, senão, continua o loop. Em loops que incrementam, é feito uma subtração e só assim a comparação com zero. A figura D.3 mostra a aplicação desta técnica no trecho de código da função *showAverage* do oxímetro de pulso. O *loop* for da linha 1 mostra que a variável de controle i é inicializada com o número de elementos do *array* e é decrementada toda vez que o trecho de código nas linhas 2 a 4 são executados. Este loop é mais eficiente, pois exige

```
1 switch(cont) {
2     case posStatus:
3           checkStatus(rawData);
4           break;
5     case posSpO2:
6           if (rawData != 127) {
7                 df2_ptr->spo2[itr]= rawData;
8           } else {
9                 df2_ptr->spo2[itr]=0;
10          }
11          break;
12    ⋮
13    case posREV:
14          srev = rawData;
15          break;
16 }
```

Figura D.2: Técnica para o *Switch*

```
1 for(i=ELEMEN; i--; ) {
2     if (sensorData[i] != 0) {
3     sensorValue = sensorValue + sensorData[i];
4     numElements++;
5     }
6 }
```

Figura D.3: Condição de parada de loops

menos instruções para processar o decremento i−− como condição de teste que verifica somente se i é diferente de 0.

Outra técnica de otimização é a elminação do excesso de chamadas de funções. No código dos experimentos nós procuramos limitar o número de argumento em quatro. Os argumentos que não estão nesse limite são passado para a função por meio da pilha, acarretando assim acessos direto à memória. Quando tínhamos uma função que possuía mais de quatro parâmetros, nós verificávamos se elas eram realmente importantes. Um outro ponto importante é evitar passar argumentos longos como *double* e *long*.

Para melhorar o tempo de processamento para as operações aritméticas, nós evitamos

usar operações com pontos flutuantes. Por exemplo, para o protótipo do *soft-starter* digital, nós desenvolvemos uma função para calcular os valores do temporizador para diversos valores de tensão do motor. Sendo assim, nós aproximamos a função responsável pelo cálculo do valor do temporizador usando uma tabela *lookup*. Trabalhar com inteiros melhora de forma significativa o tempo de processamento da aplicação [43]. Além disso, nós definimos o tipo de dado *unsigned int* para ser usado sempre que possível dentro do corpo das funções dos protótipos. Nós também evitamos ao máximo operações que envolviam divisão pelo fato delas consumirem um maior tempo de processamento [23].

Apêndice E

Linguagem de Modelagem de Processos

Este apêndice apresenta a linguagem de modelagem de processos descrita por [56]. Esta linguagem foi usada para descrever os processos que foram propostos nesta dissertação de mestrado. Esta linguagem é composta de três principais elementos gráficos como segue: área, objeto e conexão. A conexão estabelece um relacionamento entre dois diferentes objetos. A área agrupa diferentes objetos através da definição de um contexto. O objeto pode ser visto como uma pessoa ou coisa que está sob estudo ou atenção. Os objetos (descrição e notação) que foram usados nesta dissertação de mestrado são apresentados abaixo.

E.1 Descrição e Notação dos Objetos

E.1.1 Processo

Este objeto representa um conjunto de ações na qual uma ou mais entradas são usadas para produzir uma ou mais saídas.

Figura E.1: Notação do Processo.

Evento

Este objeto representa um artefato que acontece com o propósito de iniciar ou terminar um processo.

Figura E.2: Notação do Evento.

Ator

Este objeto representa uma pessoa, agente ou unidade organizacional.

Figura E.3: Notação do Ator.

Atividade

Este objeto representa um artefato que deve ser feito com o intuito de alcançar um objetivo particular. Este objeto solicita recursos e pode usar ou produzir artefatos.

Figura E.4: Notação de Atividade.

Múltiplas Atividades

Este objeto representa várias atividades que devem ser realizadas.

Figura E.5: Notação de Múltiplas Atividades.

Estado Inicial

Este objeto é originado do diagrama de estado e representa o ponto de início de um processo ou atividade.

Figura E.6: Notação de Estado Inicial.

Estado Final

Este objeto é originado do diagrama de estado e representa o ponto final de um processo ou atividade.

Figura E.7: Notação de Estado Final.

Conhecimento Explícito

Este objeto representa um conhecimento que pode ser convertido para palavras ou números. Este conhecimento pode facilmente ser transmitido e compartilhado.

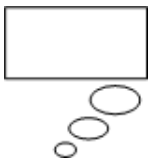

Figura E.8: Notação do Conhecimento Explícito.

Conhecimento Tácito

Este objeto representa um conhecimento que pertence a um indivíduo e é dificilmente formalizado. Este objeto também torna difícil de compartilhar o conhecimento com outros indivíduos.

Figura E.9: Notação do Conhecimento Tácito.

Artefato

Este objeto representa produtos de software que são criados ou usados durante as atividades.

Figura E.10: Notação do Artefato.

Componentes de Hardware

Este objeto representa um elemento físico que implementa funcionalidade a nível de sistema.

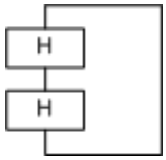

Figura E.11: Notação dos Componentes de Hardware.

Componentes de Software

Este objeto representa um pedaço reusável de software na forma binária que implementa uma ou várias responsabilidades relacionadas e tem interfaces claramente definidas.

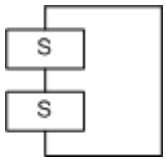

Figura E.12: Notação dos Componentes de Software.

Nota de Explicação

Este objeto permite que nota de explicação seja incluída no modelo.

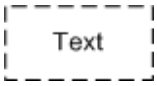

Figura E.13: Notação da Nota de Explicação.

E.1.2 Descrição e Notação das Áreas

As próximas seções apresentam a descrição e notação das áreas que foram usadas nesta dissertação de mestrado.

Grupo de Processos

Esta área agrupa os processos relacionados.

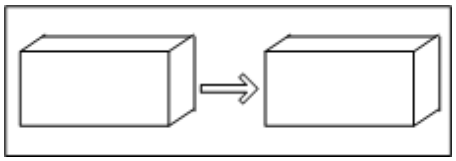

Figura E.14: Notação de Grupo de Processos.

Área do Ator

Esta área agrupa as atividades que são realizadas por um ator ou um grupo de atores.

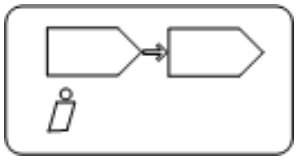

Figura E.15: Notação de Área do Ator.

E.1.3 Descrição e Notação das Conexões

As próximas seções apresentam a descrição e notação das conexões que foram usadas nesta dissertação de mestrado.

Fluxo de Entrada/Saída

Esta conexão estabelece um fluxo de entrada/saída de uma atividade.

Figura E.16: Notação do Fluxo Entrada/Saída.

Conexão não Direcionada

Esta conexão conecta eventos entre processos com o propósito de iniciá-los ou terminá-los.

Figura E.17: Notação da Conexão não Direcionada.

Conexão da Nota de Explicação

Esta conexão estabelece uma nota de explicação para um elemento.

Figura E.18: Notação da Conexão da Nota de Explicação.

Apêndice F

Publicações

F.1 Referentes à pesquisa

F.1.1 TXM: An Agile HW/SW Development Methodology for Building Medical Devices.

Cordeiro, L. C.; Barreto, R. S.; Barcelos, R. F.; Oliveira, M.; Lucena Jr., V. F.; Maciel, P. (2007). ***TXM: An Agile HW/SW Development Methodology for Building Medical Devices***. **In ACM SIGSOFT Software Enginnering Notes. ISSN:0163-5948.**

F.1.2 Agile Development Methodology for Embedded Systems: A Platform-Based Design Approach.

Cordeiro, L. C.; Barreto, R. S.; Barcelos, R. F.; Oliveira, M.; Lucena Jr., V. F.; Maciel, P. (2007). ***Agile Development Methodology for Embedded Systems: A Platform-Based Design Approach***. **In 14th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2007, Tucson, Arizona, USA. ECBS, 2007. p. 195-202.**

F.1.3 Applying Scrum and Organizational Patterns to Multi Site Software Development.

Cordeiro, L.C.; Becker, C. O.; Barreto, R. S. (2007). ***Applying Scrum and Organizational Patterns to Multi Site Software Development***. **In 6th Latin American Conference on Pattern Languages of Programming, 2007, Porto de Galinhas, Brazil. SugarLoafPlop'07, 2007.**

F.2 Contribuições em outras pesquisas

F.2.1 ezRealtime: A Domain-Specific Modeling Tool for Embedded Hard Real-Time Software Synthesis.

Cruz, F. T.; Barreto, R. S.; **Cordeiro, L. C.**; Maciel, P. (2008). ***ezRealtime: A Domain-Specific Modeling Tool for Embedded Hard Real-Time Software Synthesis***. In 11th Design, Automation and Test in Europe Conference (DATE'08). March 10-14, 2008, Munich, Germany.

F.2.2 Towards a Model-Driven Engineering Approach for Developing Embedded Hard Real-Time Software.

Cruz, F. T.; Barreto, R. S.; **Cordeiro, L. C.**; Maciel, P. (2008). ***Towards a Model-Driven Engineering Approach for Developing Embedded Hard Real-Time Software***. In 23rd ACM Symposium on Applied Computing, Track on Real-Time Systems. March 16-20, 2008, Fortaleza, Brazil.

F.2.3 Mandos: A New User Interaction Method in Embedded Applications for Mobile Telephony.

Teófilo, M.; **Cordeiro, L. C.**; Barreto, R. S.; Pereira, J. R. (2008). ***Mandos: A New User Interaction Method in Embedded Applications for Mobile Telephony***. In First IEEE International Conference on Advances in Computer-Human Interaction. Sainte Luce, Martinique, ACHI 2008.

F.2.4 Projeto e Implementação de um Plug-in Baseado no Framework do OSGi para Particionamento de Hardware/Software.

Kimura, P.; **Cordeiro, L. C.**; Barreto, R. S. ***Projeto e Implementação de um Plug-in Baseado no Framework do OSGi para Particionamento de Hardware/Software***. Início: 2007. Trabalho de Iniciação Científica. Universidade Federal do Amazonas. Conselho Nacional de Desenvolvimento Científico e Tecnológico. (Colaborador).