ALEXANDRE JOSÉ DA SILVA

ASPECTOS DA MODELAGEM EM SYSML LIGADOS À SELEÇÃO DE PROCESSADOR PARA SISTEMA EMBUTIDO

FLORIANÓPOLIS
2006

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

ASPECTOS DA MODELAGEM EM SYSML LIGADOS À SELEÇÃO DE PROCESSADOR PARA SISTEMA EMBUTIDO

Dissertação submetida à
Universidade Federal de Santa Catarina
Como parte dos requisitos para a
Obtenção do grau de Mestre em Engenharia Elétrica

ALEXANDRE JOSÉ DA SILVA

Florianópolis, Maio de 2006

ASPECTOS DA MODELAGEM EM SYSML LIGADOS À SELEÇÃO DE PROCESSADOR PARA SISTEMA EMBUTIDO

Alexandre José da Silva

‘Esta Dissertação foi julgada para obtenção do Título de Mestre
em Engenharia Elétrica, Área de Concentração em Automação e Sistemas,
e aprovada em sua forma final pelo Programa de Pós-Graduação em
Engenharia Elétrica da Universidade Federal de Santa Catarina.’

_____
Rômulo Silva de Oliveira, Doutor
Orientador

_____
Nelson Sadowski, Doutor
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

_____
Rômulo Silva de Oliveira, Doutor
Presidente

_____
Victor Juliano De Negri, Doutor

_____
Jean-Marie Farines, Doutor

_____
Nestor Roqueiro, Doutor

Dedicatória

Dedico este trabalho a minha esposa e amiga Patricia, pois sem seu apoio, amor e incentivo, não conseguiria tornar esse sonho uma realidade.

Agradecimentos

A Deus, que sempre está ao meu lado, guiando e orientando minha vida, agradeço por mais essa conquista.

A minha esposa e amiga Patricia, pela compreensão, apoio e incentivo nessa jornada.

Aos meus pais Odair e Matilde pela compreensão e incentivo de sempre e ao meu sogro Renato e sogra Lourdes, pelo carinho e orações.

A empresa Weg Industria, divisão Acionamentos, pela oportunidade de realização deste mestrado.

Ao gerente do departamento Técnico, Sr. Reinaldo Stuart Junior, pelo apoio, atenção e confiança em meu trabalho.

Ao professor e orientador Rômulo Silva de Oliveira, pela confiança depositada em meu trabalho, bem como a dedicação, otimismo e amizade com que sempre me recebeu e orientou durante a realização deste trabalho.

Aos meus colegas e amigos de mestrado, em especial o Sr. Itamar Fernandes Soares e o Sr. Luiz Henrique Schreiner Maran, no qual estivemos juntos nesta caminhada, pela companhia e amizade em todos os momentos.

Ao professor Nestor Roqueiro e Sr. Rafael Sangoi Padilha do Laboratório de Controle de Processos e Instrumentação (LCPI) pela paciência e cooperação.

A todas as pessoas que de alguma forma contribuíram para a realização deste trabalho.

Resumo de Dissertação apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

ASPECTOS DA MODELAGEM EM SYSML LIGADOS À SELEÇÃO DE PROCESSADOR PARA SISTEMA EMBUTIDO

Alexandre José da Silva

Maio/2006

Orientador: Rômulo Silva de Oliveira, Doutor.
Área de Concentração: Automação e Sistemas.
Palavra-chave: sistemas embutidos, hardware/software co-design, processadores, metodologias de seleção, modelagem de sistemas, linguagem de modelagem SysML.
Número de Páginas: 134

RESUMO: Existe atualmente uma enorme variedade de equipamentos específicos cujo controlador microprocessado está embutido nos mesmos. Tais sistemas computacionais são embutidos como componentes dentro de um sistema maior. Na perspectiva da computação, tais sistemas são conhecidos como sistemas computacionais embutidos (*embedded computer system)*, ou simplesmente sistemas embutidos (*embedded systems*). A grande maioria dos sistemas embutidos são programados e incluem componentes de hardware e de *software*. Para suportar o projeto de tais sistemas uma nova metodologia de projeto vem sendo desenvolvida denominada *hardware/software co-design.* Além do *hardware/software co-design*, é muito importante uma metodologia que leve a seleção do elemento de processamento ideal para a realização da tarefa específica do sistema embutido. O aumento de complexidade e variedade dos equipamentos com processador embutido gera a necessidade de uma abordagem interdisciplinar no processo de desenvolvimento desses equipamentos, envolvendo as áreas de engenharia de software, mecânica, elétrica e eletrônica. Neste sentido, está sendo especificada pela OMG uma linguagem de modelagem, denominada SysML (*System Modelling Language*), que pretende incluir em uma única especificação uma visão integrada do sistema, incluindo *hardware, software* e partes eletro-mecânicas. A dissertação faz um levantamento dos métodos e critérios empregados na seleção do processador a ser utilizado em um sistema embutido. As sugestões e métodos presentes na literatura são descritos, classificados e analisados. São modelados dois estudos de casos utilizando a linguagem de modelagem SysML. A partir da modelagem é realizada uma avaliação prática da atual proposta da linguagem SysML, no sentido de identificar suas capacidades e suas limitações na modelagem de sistemas embutidos, são analisados as possibilidades de extração das informações relevantes à seleção do processador embutido.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

ASPECTOS DA MODELAGEM EM SYSML LIGADOS À SELEÇÃO DE PROCESSADOR PARA SISTEMA EMBUTIDO

Alexandre José da Silva

Maio/2006

Advisor: Rômulo Silva de Oliveira, Doctor.
Area of Concentration: Systems and Automation.
Keywords: embedded systems, hardware/software co-design, processors, selection methodologies, systems modeling, modeling language SysML.
Number of Pages: 134

ABSTRACT: Nowadays, there is a huge variety of specific equipments which microprocessor controller is embedded in them. Such computer systems are embedded as components inside of a larger system. Such computer systems are known as embedded computer systems or just embedded systems. Most embedded systems are programmed and they include both hardware and software components. Methodology of hardware/software co-design have been developed to support embedded systems design. It is very important to have a methodology for the selection of the ideal processing element for the embedded system, in the context of hardware/software co-design. The processor selection methodology should select the processing element for a specific task so that it optimizes cost, performance and it reduces the product time-to-market. The complexity increase and variety of equipments with embedded processor generate the need of an multidisciplinary approach for the process of development of those equipments, involving the areas of software, mechanics, electric and electronics engineering. In this sense, it is being specified by OMG a modeling language, denominated SysML (System Modeling Language). It intends to include in a single specification an integrated view of the system, one that includes hardware, software and eletro-mechanics parts. The dissertation makes the survey methods and guidelines proposed for the selection of an embedded system processor. The methods and guidelines described in the literature are organized and analyzed. Two use case are modeling using the modeling language SysML. From the modeling is to do an empirical evaluation of the current proposal of SysML language, in the sense of identifying its capacities and its limitations for modeling of embedded systems and analyzing the possibilities of relevant information extract for embedded processor selection.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

A/D	analógico para digital
ASIC	Application Specific Integrated Circuits
ARM	Advanced RISC Machine
CI	circuito integrado
CISC	Complex Instruction Set Computer
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Device
D/A	digital para analógico
DMA	Direct Memory Access
DSC	Digital Signal Controler
DSP	Digital Signal Processor
EPLD	Erasable PLD
EEMBC	EDN Embedded Microprocessor Benchmark Consortium
EFFBD	Enhanced Functional Flow Block Diagrams
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GAL	Generic Gate Array
IDE	Integrated Development Environment
IOB	Input Output Block
LSI	Large Scale Integration
LUT	Look-up Table
MAC	Multiply-accumulator
MCU	Microcontrolador

MFLOPS	Million Float Operator Per Second
MIPS	Million Instruction Per Second
MSI	Medium Scale Integration
NoC	Network-on-Chip
OMG	Object Management Group
PAL	Programmable Array Logic
PLD	Programmable Logic Device
RAM	Randon Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read only memory
RTOS	Real Time Operated System
SoC ou SOC	System-on-Chip
SPEC	System Performance Evaluation Cooperative
SSI	Small Scale Integration
SysML	System Modeling Language
UML	Unified Modeling Language
UCP	Unidade Central de Processamento
ULA	Unidade Lógico Aritmética
VLSI	Very Large Scale Integration
VHDL	VHSIC Hardware Description Language

Capítulo 1

Introdução

Atualmente, existe uma enorme variedade de equipamentos específicos cujo controlador microprocessado está embutido nos mesmos. Tais sistemas computacionais são embutidos como componentes dentro de um sistema maior. Para o usuário deste sistema maior, o computador não é visível como tal, onde o usuário apenas percebe as conseqüências de sua operação através do comportamento externo do sistema maior. Na perspectiva da computação tais sistemas computacionais são conhecidos como sistemas computacionais embutidos, ou simplesmente sistemas embutidos (*embedded systems*).

Os sistemas embutidos estão presentes em praticamente todas as atividades humanas e, com os baixos custos tecnológicos atuais, tendem a aumentar sua presença no cotidiano das pessoas. Exemplos de tais sistemas são os telefones celulares, máquinas fotográficas digitais, sistemas de controle de carros, equipamentos industriais em geral, fornos microondas e inúmeros outros equipamentos eletrônicos.

O projeto deste tipo de sistema computacional é complexo, pois envolve questões até agora pouco analisadas pela computação de propósito geral. Existem questões sobre portabilidade e limite de consumo de energia sem perda de desempenho, a baixa disponibilidade de memória, restrições temporais da aplicação e o curto tempo de projeto.

A grande maioria dos sistemas embutidos são programados e incluem componentes de *hardware* e de *software*. A qualidade de tais sistemas é, em geral, medida por critérios específicos à aplicação incluindo desempenho, custo de desenvolvimento e de produção, flexibilidade, tolerância a falhas, etc, os quais serão atingidos dependendo de como o

hardware e o *software* são projetados. Para suportar o projeto de tais sistemas de forma efetiva uma nova metodologia de projeto vem sendo desenvolvida, denominada *hardware/software co-design* [01].

Conforme as etapas de desenvolvimento proposto pelo *hardware/software co-design*, antes de iniciar a etapa de modelagem do sistema é necessário a execução de uma etapa muito importante e fundamental, a etapa de análise de requisitos. Nesta etapa são definidas as características do sistema com base nas especificações do usuário ou do cliente. As principais características a serem capturadas incluem requisitos de tempo real, tecnologia a ser utilizada, programabilidade, consumo de energia, tamanho do produto, custo, ambiente de utilização, confiabilidade, manutenção e normas técnicas. A análise de requisitos é uma etapa informal, onde as características do sistema são descritas em linguagem natural em documento de texto de forma subjetiva, sem preocupação com os detalhes e regras de uma linguagem de modelagem.

Muitas vezes, na prática, as etapas do *hardware/software co-design* não são aplicadas na sua íntegra dentro do processo de desenvolvimento de sistemas embutidos. A etapa de especificação é a mais comprometida, pois não existe uma linguagem padronizada para modelagem das partes de hardware e software do sistema embutido. Esta lacuna gera problemas de comunicação entre as equipes interdisciplinares envolvidas no desenvolvimento do sistema embutido e de documentação do mesmo.

Diante desta lacuna, a OMG (*Object Management Group* ou Grupo de Gerenciamento de Objetos) propôs a criação de uma linguagem de modelagem específica para atender os requisitos da engenharia de sistemas. A proposta desta nova linguagem de modelagem, denominada SysML (*System Modelling Language* ou Linguagem de Modelagem de Sistema), é uma simplificação da UML (*Unified Modeling Language* ou Linguagem de Modelagem Unificada) 2.0 que possibilita em uma única especificação a abstração completa do sistema, incluindo *hardware* e *software*.

SysML é uma linguagem de modelagem de propósito geral para aplicações de engenharia de sistemas, fornecendo um padrão de descrição para uma grande variedade de sistemas complexos. Esses sistemas podem incluir *hardware*, *software*, dados, métodos, pessoal e instrumentos. Além de fornecer um padrão de modelagem, a SysML busca o aperfeiçoamento da qualidade do sistema, assegurando a troca de informações e

diminuindo a distância semântica entre a engenharia de sistemas, *software*, e outras áreas da engenharia.

Além da metodologia de *hardware/software co-design* e de uma linguagem de modelagem padronizada, é muito importante uma metodologia para a seleção do processador ideal à realização da tarefa específica do sistema embutido, otimizando custo, desempenho e reduzindo o *time-to-market* do produto.

Esta metodologia para seleção do processador é extremamente importante pois atualmente existem inúmeros fabricantes de MCU (microcontrolador), DSP (Processador Digital de Sinais) e DSC (Controlador Digital de Sinais), com inúmeras famílias, cada qual com inúmeros dispositivos. Selecionar o componente de processamento com melhor custo/benefício/desempenho para determinada aplicação é uma tarefa complexa, apresentando muitas variáveis que podem tendenciar a seleção, como a necessidade de um periférico específico, resolução e precisão exigida, consumo de energia, custo, etc.

A seleção do processador ideal para determinado sistema embutido envolve aspectos técnicos, econômicos, estratégicos e específicos à cada sistema.

O objetivo desta dissertação é avaliar através de estudos de caso o emprego da linguagem SysML para modelagem de sistemas embutidos. Em especial, avaliar a utilidade desta modelagem à seleção do processador. Este estudo limita-se a considerar as características técnicas, englobando requisitos elétricos, físicos, periféricos, memória e arquitetura. Os aspectos econômicos e estratégicos específicos a cada sistema não serão abordados na avaliação.

A dissertação está dividida em 8 capítulos, sendo o capítulo 1 a introdução, o capitulo 2 define sistemas embutidos e *hardware/software co-design*, o capítulo 3 descreve as soluções de *hardware* existentes que podem ser utilizadas como processador de um sistema embutido, o capítulo 4 faz um levantamento dos métodos existentes para seleção do processador na literatura, o capítulo 5 descreve a linguagem SysML exemplificando seus diagramas, o capítulo 6 descreve dois estudos de caso modelados com SysML, o capítulo 7 descreve os aspectos que devem ser observados na modelagem do sistema embutido e que serão utilizados para definição das características técnicas do processador a ser selecionado. O capítulo 8 traz as conclusões e perspectivas para trabalhos futuros.

Capítulo 2

Sistemas Embutidos

A maioria das funções dos sistemas eletrônicos atuais, em geral, envolvem algum tipo de computação e controle que são realizados por componentes digitais. Existem três classes básicas de sistemas digitais segundo [01]:

- **Sistemas de emulação e de prototipação:** são baseados em tecnologias de *hardware* reprogramáveis. Tais sistemas requerem usuários especialistas e são utilizados para a validação de sistemas digitais.
- **Sistemas de computação de propósito geral:** incluem computadores tradicionais abrangendo desde *laptops* até supercomputadores. Tais sistemas são caracterizados pelo fato de que os usuários finais podem programar o sistema. Diferentes aplicações são suportadas dependendo do tipo do *software* utilizado pelo usuário.
- **Sistema de computação e controle dedicados:** referenciam uma aplicação específica, na qual o usuário pode ter acesso limitado à programação do sistema. Tais sistemas, também são denominados sistemas embutidos, podem ser dedicados à aplicações de controle (controle embutido), comunicação, processamento de dados e aplicações híbridas incluindo controle e processamento de dados.

Os sistemas computacionais embutidos, também chamados de *embedded systems*, estão presentes em praticamente todas as atividades humanas e devido ao baixo custo tecnológico atual, o mercado de sistemas embutidos tem crescido numa taxa extremamente

alta não só em volume de produção, mas também em diversidade de aplicações. Essas aplicações podem ser desde de um simples eletrodoméstico como uma cafeteira até sistemas eletrônicos automotivos, aéreos, espaciais e industriais. Esta demanda crescente de mercado cria a necessidade de novas ferramentas e métodos para um suporte efetivo no projeto de tais sistemas, pois os produtos deste mercado possuem um tempo de vida relativamente curto em relação a outras aplicações, exigindo que o *time-to-market* seja o menor possível para que o produto possa ser competitivo no mercado.

O projeto de um sistema embutido é complexo, pois envolve conceitos até agora pouco analisados pela computação de propósitos gerais, e caro para uma empresa, envolvendo equipes multidisciplinares nas áreas de *hardware* digital, *hardware* analógico, *software* e testes, além de ferramentas computacionais de elevado custo.

Segundo [05], os sistemas embutidos podem ser classificados em:

- sistemas embutidos para aplicações leves;
- sistemas embutidos para aplicações médias;
- sistema embutidos de alto desempenho.

Sistemas embutidos para aplicações leves, tipicamente exigem desempenho e bom preço, resultando em código de programa reduzido e o mínimo de *hardware* extra. O consumo de energia é geralmente importante, mesmo em equipamentos que não sejam portáteis. As UCPs (Unidade Central de Processamento) geralmente utilizadas são de 4 bits ou 8 bits. Grande parte do *software* é implementada em linguagem *assembly*, devido a preocupação dos projetistas de *software* quanto ao tamanho do código do programa, pois existem restrições de memórias e de desempenho. Algumas dessas aplicações necessitam de conversor A/D (analógico para digital) e D/A (digital para analógico) para processamento de sinais [05].

Uma grande gama de aplicações exige desempenho médio, consequentemente, uma UCP embutida de custo médio. Sistemas de desempenho médio utilizam UCPs de 16 bits ou 32 bits que admitem significativa quantidade de código de programa que geralmente é armazenada em uma memória ROM. A linguagem de programação de alto nível é a mais utilizada para aplicações de desempenho médio [05].

Sistemas de alto desempenho também são fabricados com UCPs embutidas, geralmente são sistemas multiprocessadores. Aplicações de alto desempenho exigem alta velocidade de processamento devido as restrições temporais. A característica de processamento de dados em tempo real pode ser obtida através de *hardware* modificado ou por um ou mais UCPs executando um código feito sob medida. Uma outra característica complicada do projeto de sistemas de alto desempenho é o grande tempo de vida. O sistema é projetado para ter pequenos incrementos no período de muitos anos, causando problemas de compatibilidade [05].

O projeto de sistemas eletrônicos embutidos enfrenta diversos desafios, pois o espaço de projeto arquitetural a ser explorado é vasto. A arquitetura de *hardware* de um SoC (*System-on-Chip* – sistema em um circuito integrado) embutido pode conter um ou mais processadores, memórias, interfaces para periféricos e blocos dedicados. Os componentes são interligados por uma estrutura de comunicação que pode variar de um barramento a uma rede complexa (NoC – *Network-on-Chip*). Os processadores podem ser de diversos tipos (RISC, CISC, VLIW, DSP e etc), conforme a aplicação. No caso de sistemas contendo componentes programáveis, o software de aplicação pode ser composto por múltiplos processos, distribuídos entre diferentes processadores, comunicando-se através de mecanismos variados. Também pode ser utilizado um sistema operacional de tempo real (RTOS) o qual oferece serviços como comunicação e escalonamento de processos. Além do grande tempo que pode ser gasto com uma exploração sistemática deste espaço de projeto, deve-se considerar ainda o tempo necessário para o projeto e validação individual de todos os componentes dedicados do sistema, como processadores, blocos de *hardware*, rotinas de *software* e RTOS, assim como o tempo de validação do sistema completo [02].

A maioria dos sistemas embutidos são programados e incluem componentes de *hardware* e de *software*. A qualidade de tais sistemas é, em geral, medida por critérios específicos à aplicação incluindo desempenho, custo de desenvolvimento e de produção, flexibilidade, tolerância a falhas, etc, os quais serão atingidos dependendo de como *hardware* e *software* são projetados. Para suportar o projeto de tais sistemas de forma efetiva uma nova metodologia de projeto vem sendo desenvolvida denominada *hardware/ software co-design* [01].

2.1 *Hardware/Software Co-design*

Tradicionalmente, o *hardware* era considerado uma tecnologia fixa, de alto desempenho e alto custo de desenvolvimento e produção, enquanto *software* era considerado uma tecnologia com alto grau de adaptabilidade, de baixo desempenho e baixo custo de desenvolvimento e produção [04].

Os subsistemas de *hardware* eram implementados em placas de circuitos impressos com circuitos de grande (LSI - *Large Scale Integration*) ou muito grande (VLSI - *Very Large Scale Integration*) integração, ou dependendo da escala de produção com ASICs (*Application Specific Integrated Cicuits*). Devido a isto, os subsistemas de hardware possuíam baixa tolerância a mudanças. Porém, por serem confeccionados para executarem uma só tarefa, possuíam alto desempenho [04].

Subsistemas de *software*, por outro lado, eram projetados para serem executados em uma unidade de processamento, como um microprocessador ou microcontrolador. Seu desempenho era naturalmente baixo, visto que um processador, para garantir a generalidade, necessitava de instruções de ULA (Unidade lógico aritmética) para controle de fluxo, manipulação de memória e entrada e saída, execução e escrita dos resultados. Entretanto, para ser modificado, bastava-se trocar o código do processador, o que facilitava bastante a atualização deste componente [04].

Projetos de sistemas contendo componentes de *hardware* e *software* têm sido feitos na indústria e nas universidades desde os primórdios da computação (os primeiros computadores eram máquinas com partes em *hardware* e *software*). Entretanto, esses projetos eram feitos de maneira direta e improvisada *(ad hoc)*, e a escolha dos componentes nem sempre levava a um sistema de custo/beneficio/desempenho ótimo.

O objetivo principal *do Hardware/software co-design* consiste em projetar sistemas digitais que satisfaçam às restrições de projeto através da utilização de componentes de prateleira e componentes de aplicação especifica. Esta heterogeneidade dos componentes implica que em tais sistemas *hardware* e *software* devem ser desenvolvidos de forma integrada e eficiente [01].

Conforme [03], o termo *hardware/software co-design* se refere às metodologias, ferramentas e práticas que sustentam a integração de componentes de *hardware* e *software* durante o projeto e desenvolvimento do sistema .

O projeto integrado do sistema pode ser considerado como o desenvolvimento concorrente dos componentes de *hardware* e *software*, onde as tarefas de análise e especificação, particionamento, síntese e validação são realizadas concorrentemente para *hardware* e *software* [04].

Hardware/software co-design vem desafiando engenheiros desde a existência dos primeiros *hardwares* e plataformas com um *software* executando, as pesquisas com maior empenho neste campo foram iniciadas no início dos anos 90, onde iniciou-se uma nova geração de ferramentas de síntese de *hardware* e *software* [03]. O crescente interesse em *hardware/software co-design* pode ser justificado pelo avanço tecnológico e pela crescente complexidade das aplicações. A disponibilidade de ambientes de projeto suportando desde a especificação até a prototipação de sistemas digitais complexos tem permitido o projeto de uma variada gama de aplicações cada vez mais complexas.

Segundo [01], os modelos de arquitetura de hardware adotados em *co-design* podem ser classificados em duas categorias:

- **Arquitetura Monoprocessador:** um processador principal e um ou mais componentes de hardware (ASIC, FPGA e etc.). Embora os componentes de hardware possam também ser considerados processadores (com funções específicas) em muitos casos o processador principal age como o controlador dos demais componentes e da comunicação.

- **Arquitetura Multiprocessador:** vários processadores trabalhando em paralelo, com um controle distribuído de processamento. Estes processadores compreendem desde processadores de software como ASICs, FPGAs, etc.

O modelo monoprocessador é empregado pela maioria das abordagens de *co-design* devido sua simplicidade na implementação do controle, geração de interfaces e produção de estimativas de qualidade do projeto.

A metodologia de *hardware/software co-design* procura fazer com que o projeto seja tratado no nível de sistema, abstraindo detalhes de implementação, até que seja adquirido o máximo de informações que auxiliem nas decisões relativas à implementação. O projeto de sistemas embutidos segundo a metodologia de *hardware/software co-design*, envolve as seguintes etapas:

- análise de requisitos;
- especificação do sistema;
- particionamento do mesmo em hardware e software;
- síntese de interfaces entre hardware e software;
- síntese de software;
- técnicas do co-simulação;
- técnicas e métodos para geração rápida de protótipos.

Na etapa de análise de requisitos são definidas as características do sistema com base nas especificações do usuário ou cliente. As principais características a serem capturadas incluem requisitos de tempo real, linguagem de programação utilizada, arquitetura alvo, consumo de energia, tamanho do produto, custo, manutenção e evolução do projeto.

A especificação do sistema é o resultado da etapa de análise de requisito representado por um modelo. A modelagem de sistemas embutidos não é uma tarefa simples face a heterogeneidade de seus componentes e diversidade de aplicações [01].

Na etapa de particionamento do sistema, são identificadas as partes que serão implementadas em hardware e em software. A decisão é tomada com base em resultados de desempenho obtidos para cada opção de implementação através de estimadores. Os estimadores são ferramentas que analisam uma proposta de implementação, estimando o resultado que será obtido para esta proposta caso seja realmente implementada .

Após a etapa de particionamento, realiza-se a síntese do hardware, software e da interface entre ambos, chamada de co-síntese. Durante a síntese dos módulos de hardware,

uma arquitetura que implementa tais módulos é gerada. A síntese de hardware envolve o mapeamento tecnológico da descrição do hardware nas unidades físicas funcionais da arquitetura [01].

A síntese de software consiste em converter uma especificação executável num programa tradicional que possa ser compilado e executado por um processador. A problemática da síntese de software nos sistemas embutidos apresenta além do escalonamento das tarefas, a dificuldade da escolha do processador encarregado de executar as tarefas.

Os sistemas embutidos apresentam três componentes distintos, porém altamente relacionados, que precisam ser desenvolvidos: o hardware, o processador e o software para o processador. A etapa de co-simulação é indispensável no desenvolvimento conjunto dos componentes dos sistemas embutidos, reduzindo o tempo de desenvolvimento e permitindo o uso concorrente de vários simuladores. O aspecto mais importante é o de melhorar as condições de validação do software embutido comunicando com o resto do sistema, antecipando a validação funcional em ambiente real.

A prototipação é um exemplo de mecanismo para a validação final do projeto, que pode ser feita através de implementação física, criando um protótipo físico, ou de forma abstrata (simulação), através de sua especificação em alto nível.

2.2 Conclusão

A idéia básica da metodologia de *hardware/software co-design* consiste em implementar sistemas embutidos de forma sistemática, reduzindo drasticamente o tempo e custo de projeto.

O fluxo de desenvolvimento de sistemas embutidos segundo o *co-design* é subdivido em algumas etapas. Em cada uma destas etapas ferramentas de verificação e validação permitem a constatação do bom funcionamento do sistema antes de avançar para a próxima etapa.

A maioria dos sistemas de *hardware/software co-design* não implementam todas as etapas de forma automática, devido à complexidade das mesmas. Um sistema que possui co-simulação automática implementa particionamento semi-automático ou manual,

apresentando uma arquitetura restrita a uma biblioteca de componentes pré-definidos e um número e topologia dos componentes de *software*, microprocessadores, microcontroladores e DSPs fixo.

Após a especificação do sistema, deve-se utilizar uma metodologia para seleção de processadores, a fim de limitar o universo de processadores que podem ser utilizados na implementação do sistema. Na etapa de particionamento, onde define-se o que vai ser implementado em hardware ou em software, utiliza-se uma ferramenta para estimar o desempenho e tamanho do programa para diferentes arquiteturas de processadores, os quais são resultado da primeira etapa de seleção após a especificação do sistema. Entretanto, para selecionar o processador que apresenta o melhor desempenho na execução do programa, utiliza-se a técnica estática ou dinâmica. Na forma estática é compilado o código para cada processador alvo e através de simulador/emulador é verificado os valores de desempenho. Na forma dinâmica o código é compilado para um conjunto genérico de instruções que não é associado a nenhum processador. Após essa compilação são utilizadas informações sobre o tempo de execução das operações para os processadores pré-selecionados, obtendo a estimativa do tempo desejado.

Inicialmente, antes de utilizar uma metodologia para seleção de processadores para sistemas embutidos, é preciso entender os componentes que podem ser utilizados como elemento de processamento, possibilitando a identificação da melhor solução custo/beneficio/desempenho. Entre eles podemos destacar os microprocessadores, microcontroladores, DSP, DSC e PLD. Eles serão descritos no próximo capítulo.

Capítulo 3

Soluções de Hardware para Sistemas Embutidos

Inicialmente, os projetistas de sistemas embutidos limitavam-se a seleção de microprocessadores versus microcontroladores. Os recentes avanços da tecnologia de semicondutores tem gerado novas possibilidades, incluindo outras soluções de processamento como PLDs (sub-divididos em CPLDs e FPGAs), DSPs e a implementação de um *system-on-chip* (SOC ou SoC).

Com os avanços das pesquisas nas aéreas da engenharia, computação e o aumento do nível de integração, foram introduzidas novas técnicas no projeto de processadores para aplicações específicas e arquiteturas reconfiguráveis, os FPGAs (*Field Programmable Gate Array*), possibilitando o encapsulamento de um sistema de computação inteiro.

A tecnologia SoC, consiste no desenvolvimento de sistemas completos em um único CI, no qual são embutidos processador, memória, temporizadores, interfaces, etc. Dessa forma, todos os circuitos necessários para o funcionamento de um sistema são implementados em um único dispositivo.

Em muitas aplicações é adequada a integração do sistema em uma única pastilha (SoC). Em situação onde requisitos de área, potência e desempenho sejam críticos, o projeto do SoC na forma de um ASIC (circuito integrado para aplicação específica) pode ser obrigatório, elevando os custos de projeto e fabricação. Em muitas outras situações, no

entanto, é mais indicada a implementação do sistema em um FPGA, alternativa de customização mais econômica para baixos volumes, ou através de sistemas baseados em famílias de microprocessadores ou microcontroladores, componentes que são fabricados em grandes volumes e integram milhões de transistores. Essa solução tem sido amplamente utilizada pela maioria dos fabricantes da área de *hardware.*

Na atual situação tecnológica apresentada pela indústria de componentes eletrônicos, podemos destacar os seguintes elementos de processamento para o projeto de sistemas embutidos:

- microprocessador;
- microcontrolador;
- processador digital de sinal (DSP);
- dispositivo com conjunto de portas programáveis (FPGA);
- dispositivo complexo de lógica programável (CPLD);
- circuito integrado para aplicação específica (ASIC);

3.1 Microprocessadores

O microprocessador é uma UCP em um único circuito integrado, o qual pode ser muito complexo, apresentando alguns milhões de transistores. Os transistores formam os mais diversos circuitos lógicos, como registradores, decodificadores, contadores, só para citar os mais simples e comuns. Estes inúmeros circuitos lógicos arranjados em estruturas complexas dão ao microprocessador a capacidade de executar operações lógicas, aritméticas e de controle. O microprocessador é constituído de diversas partes, como ilustra a figura 1.

Os microprocessadores conseguem executar com rapidez várias funções lógicas e aritméticas, sob controle de um programa externo, que dita a seqüência de execução das operações. Estas operações são executadas sob o comando de um sinal síncrono de relógio, chamado de *clock***,** que geralmente é baseado em um cristal oscilador de quartzo.

A freqüência de *clock* é dividida dentro do microprocessador (dependendo do microprocessador, utiliza-se um fator de 4 a 12), gerando uma freqüência interna menor. Um período desta freqüência interna é as vezes chamado de um ciclo de máquina ou um ciclo de instrução [06].

A unidade de controle é a parte do processador responsável pela busca das instruções na memória principal e pela determinação do tipo de cada instrução. A unidade lógico aritmética (ULA) realiza um conjunto de operações necessárias à execução de instruções [07].

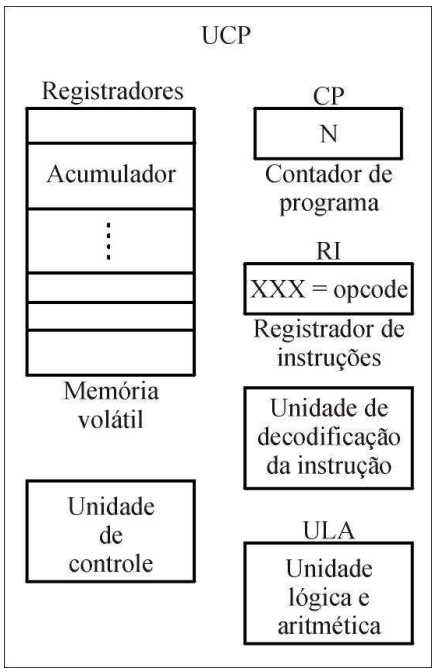


Figura 1: Estrutura interna de um microprocessador

Segundo [06], o acumulador faz parte da ULA e é responsável por conter o operando que está sendo manipulado. O tamanho do acumulador é expresso em bits e determina o tamanho do operando que pode ser processado pelo microprocessador. Isso tem reflexo direto em todo o conjunto do sistema microprocessado, por exemplo o tamanho do barramento de dados e na alocação de memória.

A unidade de controle é uma memória pequena e de alta velocidade, usada para armazenar resultados temporários e certas informações de controle. Essa memória é formada por um conjunto de registradores, cada qual com um determinado tamanho e uma determinada função. Em geral, todos os registradores de um processador têm o mesmo

tamanho. Em função de estarem situados dentro do processador, os registradores podem ser lidos e escritos a uma velocidade bastante alta [7].

De todos os registradores de um processador, o mais importante é o contador de programa (CP), cujo valor aponta para a próxima instrução a ser buscada da memória para ser executada no processador. A maioria dos processadores tem vários outros registradores, alguns de propósito geral e outros de propósito específico [07].

A ULA em conjunto com a unidade de controle é denominada Unidade Central de Processamento (UCP) [06].

Um microprocessador por si só não forma um computador, necessitando componentes externos. A figura 2, mostra a arquitetura de um sistema microprocessado simples, onde as principais características são:

- microprocessador;
- uma seção para armazenar o programa - ROM;
- uma seção para armazenar os dados temporários - RAM;
- interface com o mundo externo – entradas / saídas;
- recursos para conexão dos elementos – barramento de dados e endereço, linha de controle.

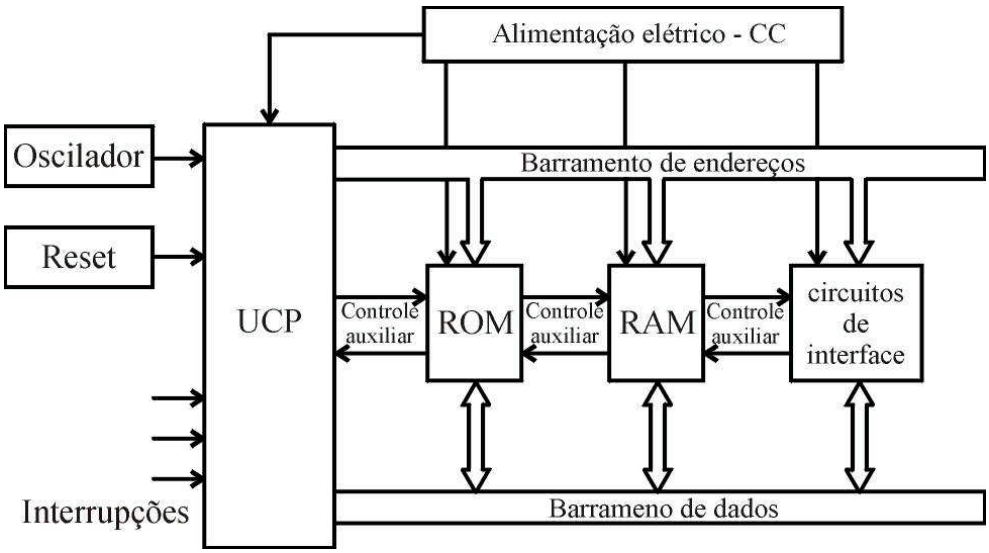


Figura 2: Componentes básicos de um sistema de hardware microprocessado

O processador em execução normal trabalha seguindo a lista de instruções indicada pelo programa. As instruções são carregadas da memória de programa uma de cada vez, decodificadas no circuito decodificador de instruções e então executadas. As instruções muitas vezes são acompanhadas por mais segmentos de código, também armazenados na memória de programa, e são tratados como dados operandos ou endereços onde o dado operando pode ser encontrado [06].

A memória de programa é geralmente uma memória do tipo ROM – *Read Only Memory* e a memória de dados do tipo RAM – *Random Access Memory*. A memória ROM possui a característica de reter os dados nela contidos mesmo com a interrupção do sistema de alimentação. Já a memória RAM é volátil, perdendo todo seu conteúdo com a interrupção do sistema de alimentação [06].

A possibilidade de programação do componente torna-o flexível para implementação de vários tipos de aplicações, podendo ser utilizado desde uma simples máquina de café até em um sistema avançado de radar.

3.1.1 Arquiteturas Típicas

Os microprocessadores apresentam basicamente arquitetura de máquina interna do tipo Von Neumann ou Hardvard.

Na arquitetura Von Neumann, memória de programa e memória de dados compartilham o mesmo barramento de dados e endereço, como mostra a figura 3, e ambos estão dentro do mesmo mapa de memória. Esta arquitetura é simples, robusta e prática, sendo amplamente utilizada com sucesso.

Se inicialmente a memória de dados é acessada, a memória de programa encontra-se desativada e vice e versa. Após definido o espaço de memória, o usuário define qual área é alocada para dados e para o programa.

A grade limitação é a velocidade no cálculo dos dados e do endereçamento feito pela ULA, devido ao compartilhamento do barramento pelos dados e instruções. Outra limitação é a de requerer o mesmo número de bits de dados e de instruções [07].

A arquitetura apresenta um tempo de compartilhamento do barramento entre os recursos de instrução e dados, limitando a velocidade máxima de execução de um programa, formando o “gargalo “ de Von Neumann.

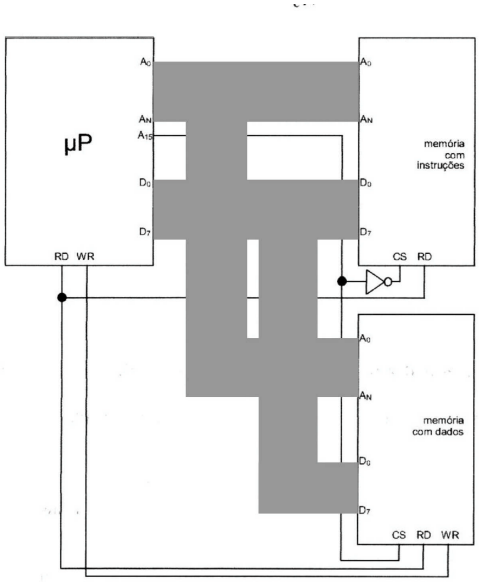


Figura 3: Arquitetura Von Neumann

Conforme [08], utilizando a arquitetura de Von Neumann para realizar uma soma de dois números, é necessário micro código, que nada mais é que uma seqüência de instruções (programa) que fará com que a ULA faça a operação de soma.

Entretanto é possível ter mais que um barramento de endereço e dados, colocando a memória de dados e memória de programa em mapas de memória diferentes. Essa abordagem é obtida com a arquitetura Harvard, como mostra a figura 4. As instruções podem ser carregadas independentemente e se necessário simultaneamente com a execução de algumas instruções, eliminado o “gargalo” de Von Neumann.

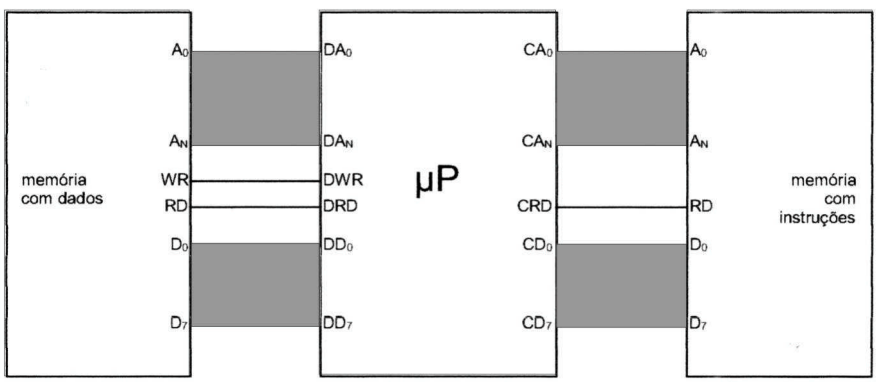


Figura 4: Arquitetura Harvard

Os dois barramentos de dados podem ter tamanhos diferentes, assim como os de endereço, permitindo a otimização de cada um, trazendo melhoramentos importantes na estrutura de alguns processadores [06].

Nesta arquitetura, para executar uma operação de soma de dois números, é necessário apenas um ciclo de máquina, um vez que não são utilizados os micro códigos, pois as operações são feitas diretamente em cada um dos blocos e através dos barramentos de dados e de instruções. Isso fornece uma grande vantagem, enquanto uma instrução é processada, outra pode executar seu ciclo de busca, carregando a próxima instrução [08].

A arquitetura Harvard permite o processamento, ao mesmo tempo, de um ciclo de busca e um ciclo de execução, assim a cada ciclo de máquina existe uma instrução pronta para ser executada. Este sistema de busca / execução é conhecido como estrutura *pipeline* [08].

3.1.2 Conjunto de Instruções

Segundo [07], os primeiros microprocessadores eram baseados em interpretadores. A metodologia de projeto baseada em interpretadores era a capacidade de projetar um processador muito simples, utilizando arquitetura Von Neumann e deixando a complexidade confinada no interpretador. Portanto, um projeto complexo de hardware poderia ser transformado em um projeto complexo de software.

O interpretador é um software que divide as instruções de sua máquina alvo em um conjunto de pequenos passos (microinstruções), ficando armazenado na memória de controle, do tipo ROM, na unidade de controle da UCP. Como consequência, o processador no qual o interpretador roda pode ser muito mais simples e muito mais barato que o processador construído em hardware para a execução das instruções da máquina alvo. Essa economia é especialmente importante se a máquina possuir um grande número de instruções e se as suas instruções forem complexas, com muitas opções de execução.

Conforme [08], as instruções estão relacionadas ao tipo de arquitetura utilizada no microprocessador. Para a arquitetura do tipo Von Neumann tem-se o CISC – *Complex Instruction Set Computer*, apresentando instruções complexas baseadas em interpretador.

Para a arquitetura Harvard tem-se o RISC – *Reduced Instruction Set Computer*, apresentando instruções reduzidas baseadas em hardware.

Nos últimos anos da década de 1970, a técnica dominante para projeto de processadores privilegiava as instruções muito complexas, cuja implementação era muito simples com o emprego do interpretador. Os projetistas estavam tentando minimizar o chamado "*gap* semântico", a distância entre aquilo que a máquina podia realizar e aquilo que as linguagens de programação de alto nível exigiam.

Porém, alguns grupos de pesquisa (IBM e Universidade de Stanford) resolveram seguir o caminho oposto, criando processadores como o RISC I, RISC II e MIPS, que não utilizavam interpretação, designando-os como padrão RISC.

Esses processadores eram muito diferentes dos processadores comercializados na época, possuindo novos conjuntos de instruções que maximizavam a performance do sistema. Essas instruções eram simples, sendo executadas rapidamente e podendo ser iniciadas prontamente, favorecendo a técnica de *pipeline*. Uma característica que chamava atenção era o seu pequeno número de instruções, em torno de 50, bem menor que as 200 a 300 instruções dos processadores CISC.

Um processador RISC é capaz de executar as instruções muito mais rapidamente. A idéia principal, é que apesar de um processador CISC ser capaz de executar centenas de instruções diferentes, apenas algumas são usadas freqüentemente. Devido a isso, pode-se criar um processador otimizado para executar apenas estas instruções simples que são usadas mais freqüentemente. Em conjunto com um software adequado, este processador seria capaz de desempenhar quase todas as funções de um processador CISC, acabando por compensar suas limitações com uma maior velocidade de processamento. É indiscutível, porém, que em instruções complexas os processadores CISC saem-se melhor. Por isso, ao invés da vitória de uma das duas tecnologias, atualmente existem processadores híbridos, que são essencialmente processadores CISC, mas incorporam muitos recursos encontrados nos processadores RISC ou vice-versa.

Examinando de um ponto de vista mais prático, a vantagem de uma arquitetura CISC é que já possui muitas instruções no próprio processador, facilitando o trabalho dos programadores que utilizam linguagem de programação *assembly*, pois já dispõe de

praticamente todas as instruções que serão usadas em seus programas. Já para um CI estritamente RISC, o programador utilizando a linguagem *assembly* tem mais trabalho, pois como dispõe apenas de instruções simples, necessita combinar várias instruções sempre que precisar executar alguma tarefa mais complexa.

Como é típico a um sistema CISC de arquitetura Von Neumann, a quantidade de instruções disponíveis é grande suficiente para produzir programas eficientes e de pequeno tamanho, comparando com programas de um sistema RISC de arquitetura Harvard [08].

3.2 Microcontroladores

Um microcontrolador ou MCU, é um tipo particular de microprocessador, otimizado para atuar em controles de baixo custo e de dimensões reduzidas [06].

Inicialmente, os microcontroladores eram chamados de microcomputadores e foram projetados focando a integração de periféricos necessários para suportar as técnicas de controle em um ambiente embutido.

Um microcontrolador é um circuito integrado que no mínimo possui os elementos necessários para um sistema computacional completo: UCP, memória, circuito de *clock* (oscilador) e circuitos de entrada e saída [09].

Os microcontroladores tipicamente, oferecem módulos de periféricos adicionais como temporizadores**,** contadores, conversor analógico para digital (A/D) e/ou conversor digital para analógico (D/A), porta de dados serial e paralelo, integrados ao CI. O desempenho desses periféricos geralmente é menor que componentes especialmente dedicados a essas funções periféricas [10]. A figura 5 ilustra a arquitetura interna de um microcontrolador.

A principal diferença entre microprocessadores e microcontroladores é que microprocessadores são projetados para uso em computadores de uso geral, onde são requeridos alta velocidade e grande tamanho de palavra. Microcontroladores são desenvolvidos para uma maior integração de hardware com inúmeros periféricos incorporados [11].

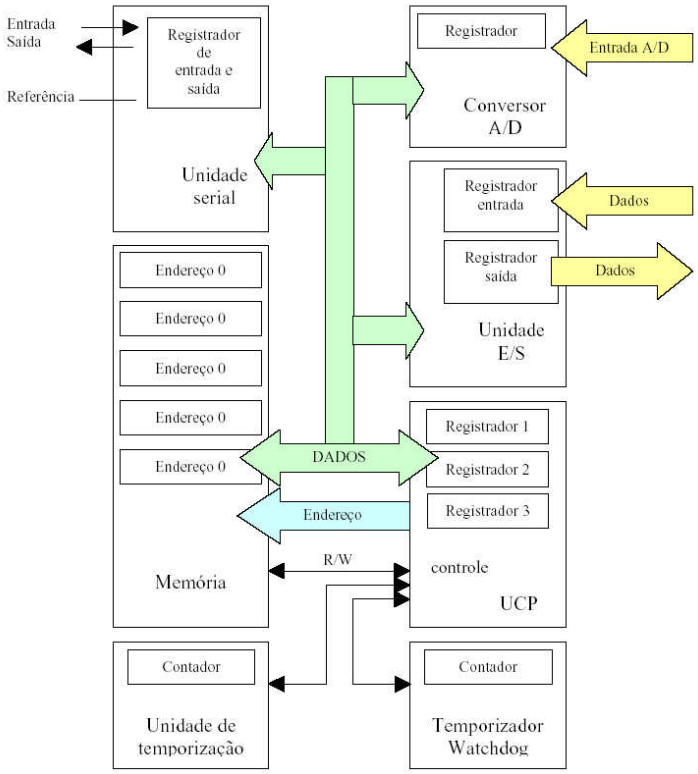


Figura 5: Estrutura interna de um microcontrolador de arquitetura Von Neumann

Outra diferença entre microprocessadores e microcontroladores consiste no conjunto de instruções. O microprocessador é rico em instruções relacionadas com o movimento de dados para dentro e fora da memória. O microcontrolador possui poucas instruções de movimento de dados, mas muitas instruções para tratamento de bits. Isso se deve ao fato dos microcontroladores apresentarem pequena memória RAM, que é somente utilizada como bloco temporário [11].

Existem diferenças na arquitetura básica da UCP, refletindo diretamente na aplicação. Equipamentos baseados em microprocessadores geralmente apresentam arquitetura Von Neumann com uma única memória para o programa e para os dados, permitindo máxima flexibilidade na alocação da memória. Microcontroladores por outro lado, freqüentemente incorporam a arquitetura Harvard, que apresenta memória de programa separada da memória de dados [11].

Segundo [12], [06] e [10], no ambiente dos controladores embutidos, os microcontroladores surgem com as seguintes vantagens:

- Grande número de entradas e saídas: capacidade de interfaceamento com inúmeros atuadores e sensores;
- Alto nível de integração: inúmeros periféricos;
- Conjunto de instruções otimizado para o ambiente embutido: código compacto, aritmética limitada, capacidade de endereçamento e forte manipulação de bits;
- Baixo custo: um componente substitui vários outros;
- Maior confiabilidade: menos encapsulamentos, menos conexões;
- Melhor desempenho: componentes do sistema são otimizados;
- Baixo consumo de energia;
- Capacidade de operação em ambientes com muita variação de temperatura e alta radiação eletromagnética;
- Baixo RF: sinais rápidos não são radiados da placa do sistema, emitindo pouco ruído eletromagnético.

Conforme [10], os microcontroladores apresentam as seguintes desvantagens:

- Flexibilidade reduzida, pois não pode-se alterar as funções projetadas dentro do CI;
- Expansão limitada ou impossível de E/S ou memória;
- Taxa de transferência de dados limitada devido ao tamanho útil e limite de velocidade para um único CI;
- E/S com baixo desempenho devido ao projeto compatível com todos os periféricos do CI.

Em um extremo, microcontroladores simples são utilizados para substituir simples sistemas digitais, entretanto, avançados microcontroladores de 32 bits desempenham sofisticadas atividades como o processamento digital de sinais [06].

Existe uma grande faixa de aplicações para microcontroladores, como em automóveis, celulares, eletrodomésticos e brinquedos. Para essas aplicações com alto volume, freqüentemente são desenvolvidos controladores dedicados. Já para aplicações de baixo volume como aparelhos científicos ou hospitalares, são utilizados microcontroladores de propósito geral [06].

3.3 Processador Digital de Sinal – DSP

O processamento aplicado em um sinal que é executado por um computador digital é chamado de processamento digital de sinal. Em sistemas modernos de processamento digital de sinal geralmente utiliza-se um único CI de microcontrolador especialmente projetado. Esse microcontrolador especial é conhecido como um processador digital de sinal, ou simplesmente DSP [13].

Um DSP é, basicamente, um tipo de microcontrolador otimizado para efetuar processamento matemático enquanto os demais processadores são otimizados para manipulação e gerenciamento de dados. Esta característica o faz apto a trabalhar com aplicações que não toleram atrasos de transporte significativos entre aquisição de um dado digital, efetuação dos cálculos necessários e devolução da resposta digital.

O DSP caracteriza-se por apresentar blocos dedicados que permitem ao processador um funcionamento eficiente na execução de algoritmos de processamento de sinais. O DSP efetua operações repetitivas num conjunto de dados, como é o caso de filtros digitais (FIR – *finite impulse response*), FFT (transformada rápida de Fourier), processamento de imagens ou multimídia. Para efetuar estas operações, o DSP possui unidades aritméticas de alta velocidade tais como MACs (*multiply-accumulator*), unidades de cálculo de endereços, deslocadores circulares e memórias múltiplas, ligadas em paralelo e isoladas por *pipelines* para aumentar a velocidade de cálculo [01].

Conforme [14] sugere, processadores que implementam processamento digital de sinal devem apresentar as seguintes características:

- Operação em tempo real;
- Determinismo;

- Fluxo de dado dirigido;

- Pequenos programas com alto grau de recursividade.

Todos os processadores DSP utilizam a arquitetura Harvard, separando memória de dados da memória de programa. Isso permite que uma instrução do programa e um dado possam ser acessados ao mesmo tempo. O processamento digital de sinal em sistemas de tempo real apresenta um crítico fluxo de dados entrando e saindo do processador. Utilizando a arquitetura Harvard, o fluxo de dados não precisa ser interrompido para a leitura de instruções [13].

DSPs tipicamente incorporam a essência dos sistemas em tempo real, onde realizam tratamento contínuo de pacotes de dados. Essas operações não podem perder um *deadline* sem que sejam introduzidos sérios erros nos dados processados [13].

No processamento em tempo real, o sinal de saída é produzido ao mesmo tempo que o sinal de entrada é adquirido. Por exemplo, isto é necessário na comunicação, em aparelhos auditivos, celulares e radares. Estas aplicações devem ter a informação rapidamente disponível.

O fato do DSP estar otimizado para processamento matemático não o impede de efetuar tarefas de gerenciamento. O mesmo vale para os demais processadores que podem efetuar processamento matemático, porém o desempenho será menor quando comparado com um processador otimizado àquela aplicação.

O formato numérico de um processador está ligado a forma como são armazenados e manipulados os dados em sua arquitetura interna. Os dados podem ser trabalhados tanto no formato de ponto fixo quanto no formato de ponto flutuante. Deste modo, os DSPs são divididos em duas grandes categorias, definidas pelo modo de como é tratado o formato numérico [13].

A representação de ponto fixo é caracterizada pelo ponto *radix* e pelo comprimento da palavra (16 bits, 32 bits...). O ponto *radix* indica o modo de como a palavra binária é escalonada. Os números de ponto fixo podem, ainda ser subdividos em:

- Representação com sinal e representação sem sinal;

- Representação por números inteiros e representação fracionária.

Os dados do tipo ponto flutuante, são caracterizados por um campo para a mantissa (fração), um campo para o expoente e um bit de sinal. A principal norma que define estes formatos é a IEEE 754 [13].

Os DSPs de ponto flutuante tipicamente utilizam um mínimo de 32 bits para armazenar cada valor. Isto resulta em um número maior de bits para representação do que no ponto fixo. Um ponto importante a ser ressaltado sobre a notação de ponto flutuante é que os números representados não são uniformemente espaçados.

O fato do processador trabalhar com um tipo específico de dado não o impede de trabalhar com o outro tipo de dado. Os DSPs de ponto fixo podem manusear dados de ponto flutuante, somente que, isto deve ser feito por software e não por hardware, o que acaba onerando a programação e o tempo de execução.

Já nos DSPs de ponto flutuante o hardware está adaptado para trabalhar com os dois formatos, porém, necessitam de um conjunto de instruções maior do que nos DSPs de ponto fixo. Isto se faz necessário devido a necessidade de usar contadores, laços e os próprios sinais provenientes dos conversores A/D que trabalham com números no formato de ponto fixo, tornando necessária instruções que também manipulem este tipo de dado.

A principal vantagem do formato de ponto fixo está no custo. Pelo hardware ser mais simples, seu custo é geralmente bem menor do que um DSP de ponto flutuante.

A diferença chave entre os DSPs de ponto fixo e de ponto flutuante está ligado ao desempenho de ambos em relação a característica de sinal-ruído. A representação por palavras binárias só é possível para um determinado conjunto de valores. Se um número não coincidir com um dos valores este será arredondado, inserindo um ruído denominado de ruído de quantização. No formato ponto flutuante, o espaço entre dois números adjacentes é muito menor do que no formato de ponto fixo, significando que DSPs de ponto fixo apresentam um ruído de quantização muito maior [13].

3.4 Controlador Digital de Sinal - DSC

É uma nova categoria de controladores eletrônicos, que combina a maioria das características dos microcontroladores e dos DSPs. Esse controlador híbrido é chamado de controlador digital de sinal. O DSC combina o poder computacional dos DSPs com facilidade de uso dos microcontroladores.

Os microcontroladores são otimizados para processar conjuntos de operações lógica, aritmética e de diagnóstico com os dados de entradas provenientes de diversas fontes. Os DSPs são muito eficiente para tarefas repetitivas de cálculos. Com base nas características especificas que definem um DSP ou um MCU, fica claro que o mesmo hardware está envolvido em ambos os componentes. O DSC é uma evolução lógica e progressiva desta categoria de produtos. Tudo indica que o DSC substituirá os MCUs e DSPs, pois é um componente que atende as aplicações sensíveis a custo e que necessitam os dois componentes. Controle de motores é uma aplicação que pode ser beneficiada com o advento dos DCSs [15].

Segundo [15], as vantagens da integração das funções de um controlador com as funções de um processador em um único CI, são:

- redução do custo de componentes;
- redução do espaço da placa eletrônica;
- aumento da confiabilidade.

O grande beneficio destas características em uma única arquitetura, que é equivalente a uma aproximação com um sistema que utiliza coprocessador, por exemplo, um sistema que utiliza um microcontrolador e um coprocessador DSP integrado, é a necessidade de desenvolvimento de apenas um único software que controla a parte de controle e a parte de processamento de sinal [15].

3.5 Lógica Programável

Um desenvolvimento recente em eletrônica digital foi a introdução dos dispositivos de lógica programável (PLD – *Programmable logic device*). Antes do desenvolvimento

dos PLDs, circuitos digitais eram construídos com várias escalas de integração de circuitos lógicos, como componentes com pequena integração (SSI – *Small scale integration*) e média integração (MSI – *Medium scale integratiom*). Esses componentes continham portas lógicas e outros circuitos digitais. As funções eram determinadas pelo fabricante e não podiam ser alteradas. Se o projetista necessitasse de um componente com uma função particular a qual não era fornecida pelo fabricante, ele era forçado a implementar um circuito usando vários componentes lógicos, necessitando um grande espaço na placa eletrônica e um grande tempo de desenvolvimento [16].

A lógica programável trouxe a solução para esses problemas. Um PLD é fornecido sem qualquer função lógica programada. Todas as funções requeridas pelo projeto são programadas pelo projetista através de linguagem VHDL, respeitando o desempenho do PLD. Visto que várias funções geralmente podem ser combinadas no projeto e programadas em apenas um PLD, o número de componentes e o tamanho da placa eletrônica podem ser reduzidos. Outro aspecto relevante é a alteração do projeto, pois um PLD pode ser programado com a nova solução sem a remoção de componentes do circuito [16].

Com o PLD, projetistas utilizam ferramentas de software para o desenvolvimento rápido, simulação e testes. O circuito projetado pode ser rapidamente programado no componente e imediatamente testado em um sistema [17].

PLD é um termo genérico. Existe uma grande variedade de tipos de PLD, incluindo PAL (*Programmble Array Logic*), GAL (*Generic Gate Array*), EPLD (*Erasable PLD*), CPLD (*Complex PLD*) e FPGA (*Field Programmable Gate Array*). Os dois principais tipos de lógica programável são os CPLDs e os FPGAs.

Os CPLDs apresentam menor densidade lógica, mas possuem uma boa característica para estimar tempos de processamento, sendo assim ideais para aplicações de controle crítico. CPLDs são componentes que apresentam baixo consumo e custo, tornando-se uma boa alternativa para aplicações sensíveis ao custo, alimentadas por bateria e portáteis como telefonia móvel e assistentes digitais de mão [17].

Os FPGAs apresentam alta densidade lógica, mais características/módulos e alto desempenho. Esses avançados componentes oferecem características como um processador

hardwired (como o IBM PowerPC), substancial quantidade de memória, sistema de gerenciamento de *clock* e suporte para muitos outros componentes. São utilizados em uma grande variedade de aplicações, desde de processamento e armazenagem de dados até o processamento digital de sinais [17].

Cada tipo de PLD apresenta vantagens que os tornam mais adequados para algumas aplicações do que outros. Atualmente, um projetista depara-se com a difícil tarefa de pesquisar os diferentes tipos, entender qual sua melhor utilização, escolher um fabricante, aprender a utilizar as ferramentas de desenvolvimento, para só então começar a projetar o hardware.

3.5.1 *Complex Programmable Logic Device* – CPLD

O primeiro dispositivo desenvolvido para a implementação de circuitos lógicos foram os PLA (*Programmable logic array*) [18].

Originalmente, um PLA consistia em um arranjo de portas lógicas ***AND*** e ***OR*** programáveis, conectadas em uma matriz de fusíveis de ligação. Um fusível intacto permite a realização de uma conexão e um fusível queimado rompe a conexão [16].

A figura 6 ilustra uma simples matriz de fusíveis conectada a uma porta lógica AND de 4 entradas.

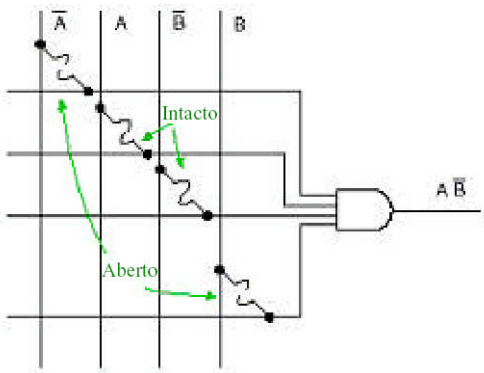


Figura 6: Matriz de fusíveis

Os PLAs foram introduzidos pela Philips no início dos anos 70, estes dispositivos consistem de dois níveis de portas lógicas: um plano de portas *wired-AND* seguido por um plano de portas *wired-OR*, ambos programáveis. Um PLA é estruturado de tal forma que cada saída do plano AND pode corresponder a qualquer temo do produto das entradas. Da

mesma forma, cada saída do plano OR pode ser configurada para produzir a soma lógica de qualquer saída do plano AND [18]. A figura 7 mostra o esquema simplificado de um PLA.

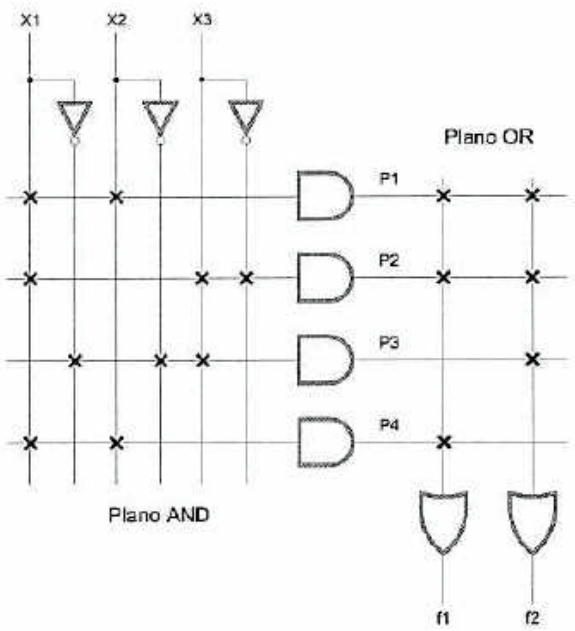


Figura 7: Esquema simplificado de um PLA

Devido a esta estrutura, os PLAs são adequados para implementação de funções lógicas na forma de soma de produtos, e eles se apresentam muito versáteis, pois tanto os termos AND como os termos OR podem ter muitas entradas. Porém, essa tecnologia também apresenta alguns problemas como alto custo de fabricação e baixo desempenho em termos de velocidade. Essas desvantagens existem devido aos dois níveis de lógica configurável. Os planos lógicos programáveis são difíceis de serem fabricados e introduzem atrasos significativos de propagação dos sinais elétricos [18].

A tecnologia PAL (*Programmable array logic*) foi desenvolvida, superando as deficiências da arquitetura PLA. O PAL apresenta um único nível de programação, um custo mais baixo e um melhor desempenho, pois apenas o plano AND é configurável. A figura 8 mostra a estrutura do PAL.

Os dispositivos PAL foram importantes, pois, quando foram introduzidos, tiveram um profundo efeito no projeto de hardware digital, e também foram a base para algumas das novas e mais sofisticadas arquiteturas. Variantes da arquitetura básica do PAL são encontrados em outros produtos conhecidos por diferentes siglas. Todos os pequenos

PLDs, como PLAs, PALs e outros dispositivos similares são agrupados em uma única categoria chamada SPLDs (*simple* PLDs), cujas características mais importantes são o baixo custo e alto desempenho [18].

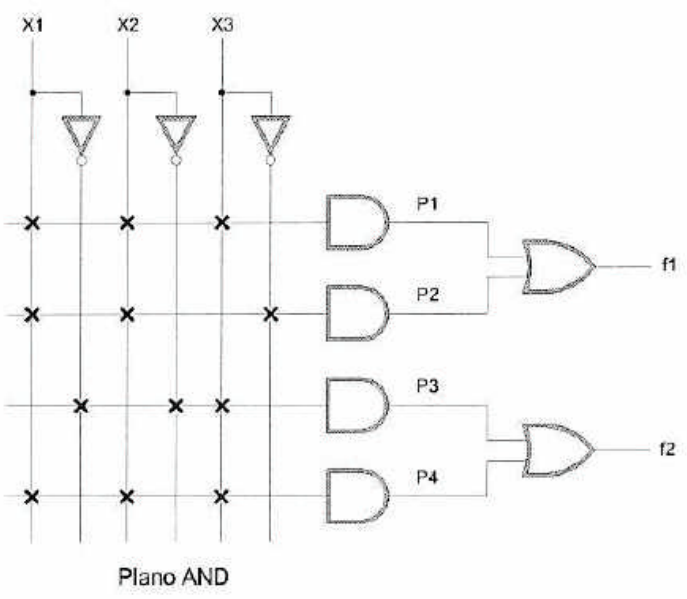


Figura 8: Esquema simplificado de um PAL

Com o avanço da tecnologia tornou-se possível a produção de dispositivos com maior capacidade que os SPLDs. A dificuldade de aumentar a capacidade da arquitetura de SPLD, é que a estrutura dos planos lógicos programáveis aumenta muito rapidamente com o aumento do número de entradas. O único modo viável de produzir dispositivos com maior capacidade baseados nas arquiteturas SPLDs foi integrar múltiplos SPLDs em um único CI, e prover interconexões programáveis para conectar os blocos SPLDs. Muitos produtos são encontrados no mercado atualmente com essa estrutura básica, e são coletivamente chamados de CPLDs (*Complex* PLDs) [18]. A figura 9 apresenta a estrutura de um CPLD.

Os CLPDs foram introduzidos pela Altera Corporation, inicialmente com sua família de CIs chamada Classic EPLDs (*Erasable* PLDs), e em seguida com a séries MAX. Devido ao rápido crescimento do mercado para grandes PLDs, outros fabricantes desenvolveram dispositivos na categoria CPLD e há atualmente diversas opções disponíveis. Os CPLDs provêem capacidade lógica de até 50 dispositivos SPLDs típicos, mas é difícil estender essa arquitetura para densidades maiores. Para construir PLDs com capacidade lógica muito alta, uma abordagem diferente é necessária [18].

A principal tecnologia de chaveamento utilizada nos CPLDs, são os transistores de porta flutuante, como aqueles encontrados em memórias do tipo EPROM (*Erasable Programmable Read-only*) e EEPROM (*Electrical Erasable PROM*).

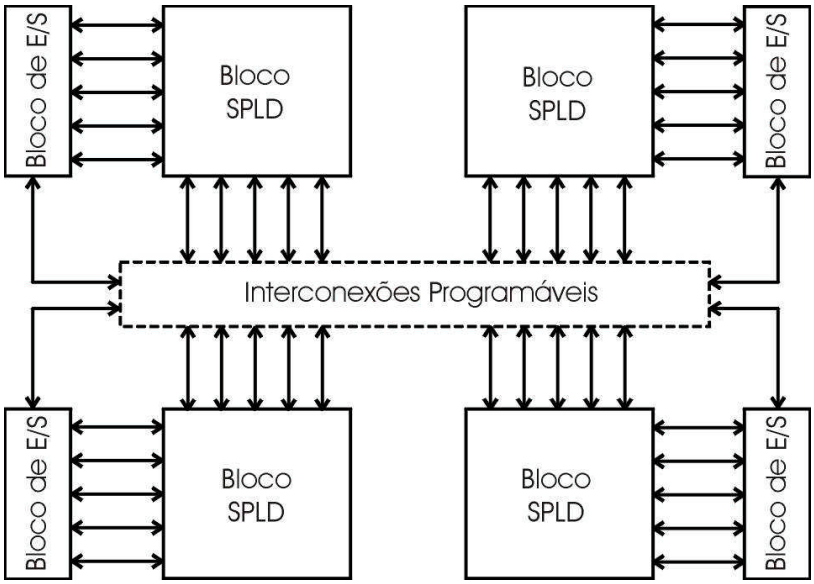


Figura 9: Estrutura interna de um CPLD [18]

Os CPLDs são utilizados em muitas aplicações, como implementação de lógicas diversas e pequenos conjuntos de portas, devido a sua alta velocidade e ampla faixa de capacidade. Uma importante razão do crescimento do mercado de CPLDs, é a conversão de projetos que consistem de múltiplos SPLDs para um pequeno número de CPLDs.

Uma vantagem dos CPLDs é que eles permitem a reprogramação de simples alterações de projeto. Há CPLDs programáveis *in-system* que tornam possível a reconfiguração de hardware sem o desligamento de energia. A grande vantagem da arquitetura CPLD é a possibilidade de prever a implementação dos circuitos e com isso prever também seu desempenho [18].

3.5.2 *Field-Programmable Gate Arrays* - FPGA

O *Field programmable gate array* ou FPGA é um circuito integrado que oferece grande capacidade e velocidade. São constituídos de um arranjo (*array)* de blocos lógicos configuráveis que implementam funções lógicas, podendo ser modificado praticamente em qualquer momento durante o uso.

Um FPGA típico possui uma arquitetura interna, como ilustra a figura 10, composta de uma matriz de blocos lógicos configuráveis (CLB - *Configurable logic block*) cercados por uma rede de interconexão programável, formada de blocos de interconexão [18].

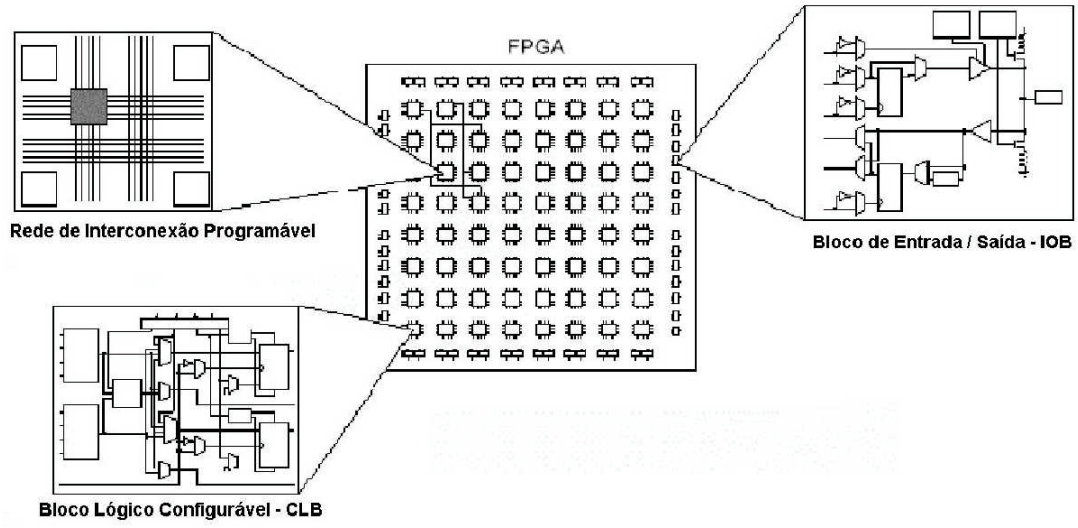


Figura 10: Estrutura básica de um FPGA [18]

Circundando todo o circuito, existem os blocos de entrada e saída (IOB - *Input output block*), que também são programáveis, e que servem como interface entre o mundo exterior e a lógica interna do dispositivo.

A arquitetura de um CLB varia de família para família e de fabricante para fabricante, mas basicamente são compostos de pontos de entrada, que se conectam a blocos que implementam funções puramente combinacionais (LUT – *Look-up table*), multiplexadores que direcionam o fluxo dos sinais internamente ao CLB e de registradores (tipicamente *flip-flops*) que estão ligados às saídas e também podem realimentar as entradas dos geradores de funções combinacionais.

Todos os elementos são configuráveis, oferecendo grande flexibilidade para implementação de funções. A rede de interconexão programável é composta por diferentes tipos de segmentos de conexão, capazes de interligar a maioria das entradas e saídas dos CLBs entre si e aos IOBs. Isso tudo permite que circuitos complexos, máquinas de estado, e algoritmos sejam implementados nos FPGAs [18].

A forma mais utilizada de se construir o bloco lógico é por meio de uma tabela de pesquisa (*look-up table)* (LUT), que contém células de armazenamento usadas para implementar uma pequena junção lógica.

Cada célula é capaz de reter um único valor lógico 0 ou 1. LUTs de vários tamanhos podem ser criados. A figura 11 ilustra um LUT com duas entradas, x1 e x2, e uma saída f. As variáveis de entrada x1 e x2 são usadas como chaves seletoras para os três multiplexadores 2x1 que em conjunto, selecionam uma das quatro células de armazenamento como saída do LUT. Nos FPGAs disponíveis comercialmente, os LUTs possuem geralmente quatro ou cinco entradas, o que exige 16 e 32 células de armazenamento respectivamente [19].

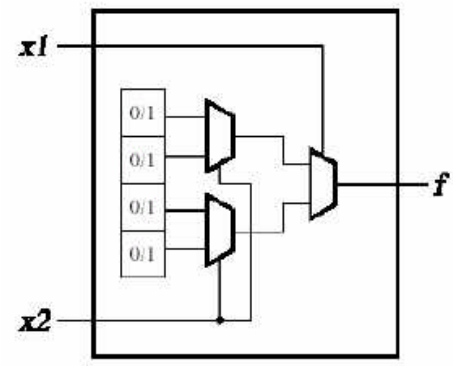


Figura 11: Circuito para um LUT com 2 entradas

Quando um circuito lógico é implementado em um FPGA, os blocos lógicos são programados para realizar as funções necessárias, e os canais de roteamento são estruturados de forma a realizar as interconexões necessárias entre os blocos lógicos.

As células de armazenamento dos LUTs de um FPGA são voláteis, o que implica na perda do conteúdo armazenado no caso de sobre tensão de alimentação. Desta forma, o FPGA deve ser programado toda vez que for energizado. Geralmente um pequeno CI de memória PROM é incluído nas placas de circuito impresso que contem FPGAs. As células de armazenamento são automaticamente carregadas a partir das PROMs toda vez que uma tensão elétrica é aplicada a estes CIs.

Dada a descrição de um circuito e sabendo alguns dos objetivos do projeto, o problema é implementar o circuito usando blocos básicos da arquitetura FPGA. Conforme [19], geralmente utiliza-se técnicas de síntese para resolver problemas como minimizar custos de projeto, satisfazer as restrições de performance, minimizar o consumo de energia, ou fazer a implementação ficar mais apta a testes.

Como síntese é um processo difícil, geralmente é dividido em duas fases: otimização independente de tecnologia e mapeada à tecnologia [19].

Segundo [19], as principais restrições na síntese de circuitos em FPGA são:

1. Um número limitado de blocos lógicos em um CI;

2. A funcionalidade de bloco, ou seja, que funções podem ser colocadas em um bloco;

3. Recursos limitados de conexão através de recursos elétricos;

4. Atrasos (*delays*) de interconexão são significantes e não podem ser ignorados durante a síntese.

Atualmente existem FPGAs da ordem de 10 milhões de portas lógicas, permitindo a implementação de sistemas extremamente complexos em um único CI (SoPC – *System on a programmable chip*). As implementações consistem desde simples circuitos, como somadores, subtratores e acumuladores, até circuitos mais complexos, como filtros e DSPs [20].

3.6 Circuito Integrado de Aplicação Especifica – ASIC

Os circuitos integrados desenvolvidos em uma pastilha de silício podem ser divididos em dois grupos de integração: circuitos integrados padrões e circuitos integrados de aplicação específica - ASIC.

Os circuitos integrados padrões são compostos por circuitos integrados de funções fixas, que correspondem aos componentes digitais básicos (funções *AND, OR, NOR, NAD, OR* Exclusivo, *Flip-Flops*, etc.), por microprocessadores e por circuitos integrados reconfiguráveis, ou programáveis (PLDs) [21].

Os circuitos integrados de aplicação específica (ASIC) são subdivididos em circuitos integrados denominados *Full Custom* e circuitos integrados denominados *Semicustom*.

Os componentes *Semicustom* se dividem nos arranjos de portas (*Gate Arrays*) e nos arranjos de células (*Cell Based*). A figura 12 mostra a classificação dos circuitos integrados.

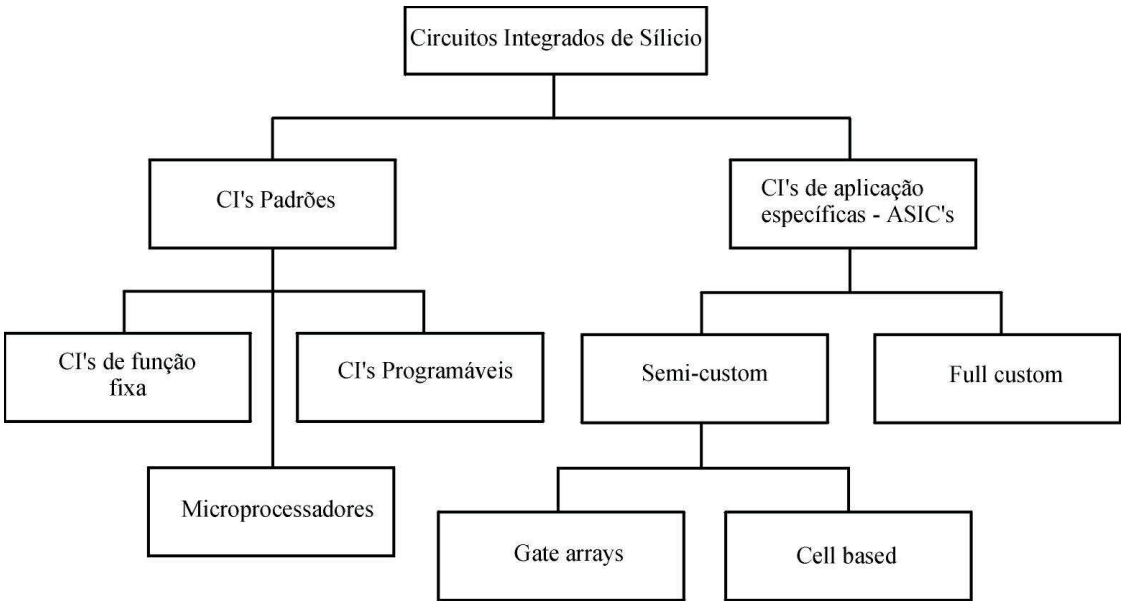


Figura 12: Classificação dos circuitos integrados [21]

Os circuitos integrados padrões são disponíveis no mercado em grande quantidade de itens. São classificados em SSI (integração de pequena escala) e MSI (integração de média escala), normalmente construídos em lógica TTL, ou CMOS, possuindo funções analógicas e digitais de propósito geral, baixo custo, devido a sua produção em massa, com blocos funcionais bastante simples e, portanto, necessitando de vários componentes para a realização de uma função específica. Ainda são disponíveis os LSI (integração de grande escala) e os VLSI (integração de muito grande escala) que possuem funções específicas mais complexas, tais como, periféricos de comunicação de dados, temporizadores, controladores de redes, interfaces de barramentos, blocos para processamento digital de sinais, etc .

Os componentes ASIC são programados através de interconexões metálicas usadas na fabricação de circuito integrado, sendo também denominados de MPLDs (*Mask-programmable logic device*), ou seja dispositivos de lógica programável por máscaras, e portanto, necessitam ter suas funções definitivas de operação implementadas na própria construção do componente [21].

Os componentes ASICs *Full Custom* tem o seu projeto desenvolvido a nível de transistores, para se obter um desempenho ótimo. Possuem longo tempo de desenvolvimento, alto custo de projeto, alto nível de integração e somente poderá ter um custo beneficio bom se produzido em grande volume [21].

Os componentes ASICs *Semicustom* são fabricados em quantidades de milhares de peças. O projeto do usuário estará baseado em blocos pré-desenvolvidos, ou em células contendo blocos lógicos, *flip-flops*, ou funções digitais maiores. O usuário não executa o seu projeto a nível de transistores, ou de desenvolvimento de células. Estes componentes geralmente são classificados em arranjos de blocos (*gate arrays*), em arranjos de células (*cell based*) e em arranjos gerais de células (*macrocells*, ou *general cells*). Em todos os casos, a fabricação final do componente é realizada pelo fabricante de circuitos integrados [21].

Macro-células são construídas com blocos e células compondo uma biblioteca de funções, com nenhuma restrição no posicionamento destes itens em uma fila ou em uma matriz [21].

É possível incluir áreas de memória RAM e ROM, conjuntos lógicos programáveis (PLA) e grandes blocos funcionais, equivalentes aos VLSIs, por exemplo, microprocessadores, etc.

Gate arrays são arranjos regulares de blocos que possuem células periféricas para possibilitar interligações de entrada e saída. Podem ser arranjados em blocos, em filas, ou em matriz de blocos (*sea-of-gates*). Usualmente, somente as camadas de metais que interligam esses blocos é que serão desenvolvidas [21].

Standard Cells são compostos de uma biblioteca de células padrões arranjadas em arquitetura de filas e com canais de roteamento desenvolvidos de acordo com a necessidade do usuário. Algumas vezes somente se desenvolvem camadas de metais, outras vezes há a necessidade de se desenvolver todas as camadas no componente [21].

A figura 13 apresenta a estrutura de conjuntos de blocos organizados por blocos, filas e matriz, com também a estrutura de um ASIC composto de células padrões e blocos funcionais.

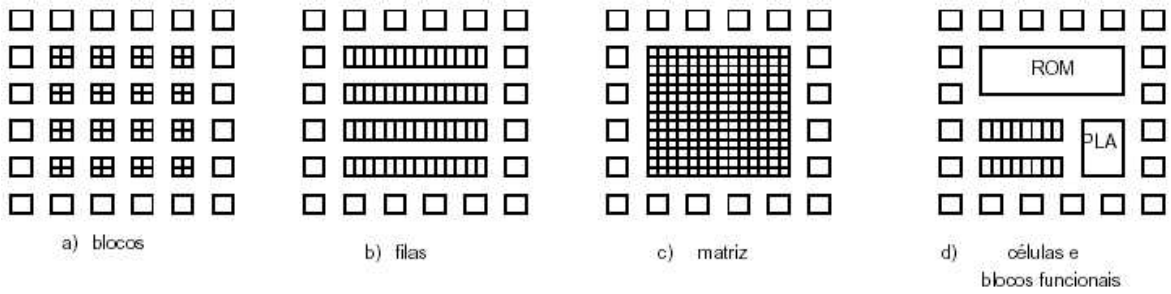


Figura 13: Estrutura de *gate arrays* organizados por: a) blocos b)filas c)matriz d) *standard cells* contendo células e blocos funcionais

3.7 Conclusão

Inicialmente os sistemas embutidos eram implementados com base em microprocessadores genéricos, apresentando razoável desempenho em aplicações de controle e processamento de sinais, além da necessidade de uma grande área de placa eletrônica, abrigando todos os componentes necessários ao funcionamento do sistema microprocessado, interfaces e componentes dedicados ao controle.

Com o advento do microcontrolador, o qual é otimizado para a implementação de controle, e do DSP, otimizado para o processamento de sinais, os sistemas embutidos iniciaram sua utilização como alternativa de melhor custo/benefício/desempenho do mercado. Contudo os principais fabricantes oferecem MCUs e DSPs com inúmeras opções de encapsulamento, variantes no números de pinos de entrada e saída, configuração de memória e características dos periféricos.

Os PLDs são utilizados com restrições devido seu alto custo. Geralmente são utilizados em conjunto com um microcontrolador ou DSP, implementando interfaces e circuitos lógicos, como complemento do sistema. Os PLDs, em especial os FPGAs, são muito utilizados em plataformas que possibilitam reconfiguração de hardware.

Os ASICs eram muito utilizados para compactar a área do circuito eletrônico, porém apresentam baixa flexibilidade e alto custo de desenvolvimento. Com o advento dos PLDs, os ASICs perderam mercado, entretanto, são uma boa alternativa para a compactação e implementação de circuitos analógicos, os quais não podem ser implementadas em um PLD.

Este capítulo descreve o universo de soluções existentes para o processamento de dados em sistemas embutidos.

Para cada tipo de aplicação existe uma solução de processamento que apresenta a melhor relação custo/benefício/desempenho, porém as soluções mais utilizadas no universo dos sistemas embutidos são os microcontroladores, DSPs e recentemente os DSCs, pois apresentam flexibilidade, baixo custo (na maioria das vezes são componentes de "prateleira") e ferramentas poderosas de desenvolvimento muitas vezes gratuitas. Devido a isso é possível o desenvolvimento de sistemas embutidos cada vez mais poderosos e com um custo menor.

Atualmente existem inúmeros fabricantes de microcontroladores, DSPs e DSCs, com inúmeras famílias, cada qual com inúmeros integrantes. Selecionar o componente de processamento com melhor custo/benefício/desempenho para determinada aplicação é uma tarefa complexa que apresenta inúmeros aspectos que devem ser observados. Como a seleção do processador é uma etapa importante no desenvolvimento de sistemas embutidos é necessário verificar na literatura a existência de metodologias ou guias, que contemplem tais aspectos, orientando a seleção do mesmo.

Capítulo 4

Metodologias para Seleção de Processador Descritas na Literatura

Neste capítulo serão apresentadas as metodologias para a seleção de processadores encontradas na literatura e alguns aspectos relevantes que devem ser observados para uma seleção correta do processador. Em computação embutida, a identificação ou seleção da melhor arquitetura que contemple todo sistema é uma questão fundamental. Procura-se identificar na literatura os aspectos que são observados no modo de selecionar o elemento de processamento, os quais devem ser extraídos dos modelos SysML.

Conforme a aplicação, o projeto de sistemas embutidos pode utilizar como plataforma um ou mais processadores. Embora esta solução pareça extremamente conservadora do ponto de vista de inovação, ela traz vantagens do ponto de vista operacional. Primeiro, o fator de escala, pois os processadores estão presentes em milhares de projetos e seu custo é diluído entre inúmeros clientes. Uma vez que uma plataforma baseada em um processador esteja disponível dentro de uma empresa, novas versões de produtos podem ser feitas pela alteração do software da plataforma. A personalização do sistema dá-se através do software da aplicação, que toma atualmente a maior parte do tempo de projeto. Além destas vantagens competitivas, há ainda o fator de treinamento de engenheiros, já que estes geralmente se formam com conhecimento de programação de microprocessadores ou microcontroladores.

É claro que o projeto usando processadores não é vantajoso em todos os aspectos. A questão da potência, cada vez mais valorizada nos tempos atuais, é crítica. Como são projetados para executar qualquer programa, existem estruturas de hardware dentro dos processadores que consomem muitos recursos, mas que são muitas vezes subtilizadas [02].

Para uma solução específica, um projeto usando lógica programável como FPGAs ou ASICs pode ter um desempenho muito melhor que usando um processador. O problema é que sistemas reais possuem diversos comportamentos (modelos computacionais) e o atendimento simultâneo dos mesmos tende a diminuir o desempenho do hardware dedicado. Além disto, é preciso considerar que se encontram processadores nas mais diversas combinações de preço, desempenho e potência. Os processadores também contam com grupos de projetos imensos, que chegam às centenas de projetistas, e com tecnologia do estado-da-arte para sua fabricação. Tudo isso torna o uso de processadores muito interessante para o projeto de sistemas embutidos [02].

Para o projetista de sistemas embutidos, a escolha de um processador é a primeira e a mais importante ação a ser tomada. Muitas vezes a entrega do produto final dentro do cronograma, respeitando orçamento (custos) e especificações, dependem da escolha correta do processador.

Na pesquisa realizada foi encontrado apenas uma metodologia que orienta passo a passo a seleção de microcontroladores. A maioria dos livros e artigos pesquisados trazem apenas aspectos relevantes a serem observados no processo de seleção do elemento de processamento para sistemas embutidos.

4.1 Metodologia para Seleção de Microcontroladores

A metodologia para seleção de microcontroladores proposto por [22] pode ser observado no fluxograma apresentado na figura 14. Conforme [22], inicialmente deve-se definir três aspectos.

Primeiramente deve-se definir o desempenho desejado e o tamanho da palavra do microcontrolador, sendo que ela pode ser dividida em: 4, 8, 16, 32 bits. *Benchmarks* padrões são inapropriados para classificar microcontroladores, podendo ser substituídos por uma análise de cálculo crítico de tempo real e restrições físicas para o sistema.

Cálculos críticos são aqueles que, caso não sejam completados em um tempo definido, causam perda de sincronismo no sistema, criando um estado instável ou gerando resultados errados. Restrições físicas impostas pelo sistema, como precisão e faixa dinâmica, determinam a classe da UCP (8,16, 32bits) [22].

A segunda definição é a identificação da quantidade, freqüência e o tipo de todos os sinais de entrada e saída, bem como outros requisitos especiais. Essa identificação é uma característica ligada à experiência do projetista, que em muitos casos não é compartilhada. A chave para essa parte da seleção é um diagrama completo do sistema, não especificando funções dos periféricos do microcontrolador [22].

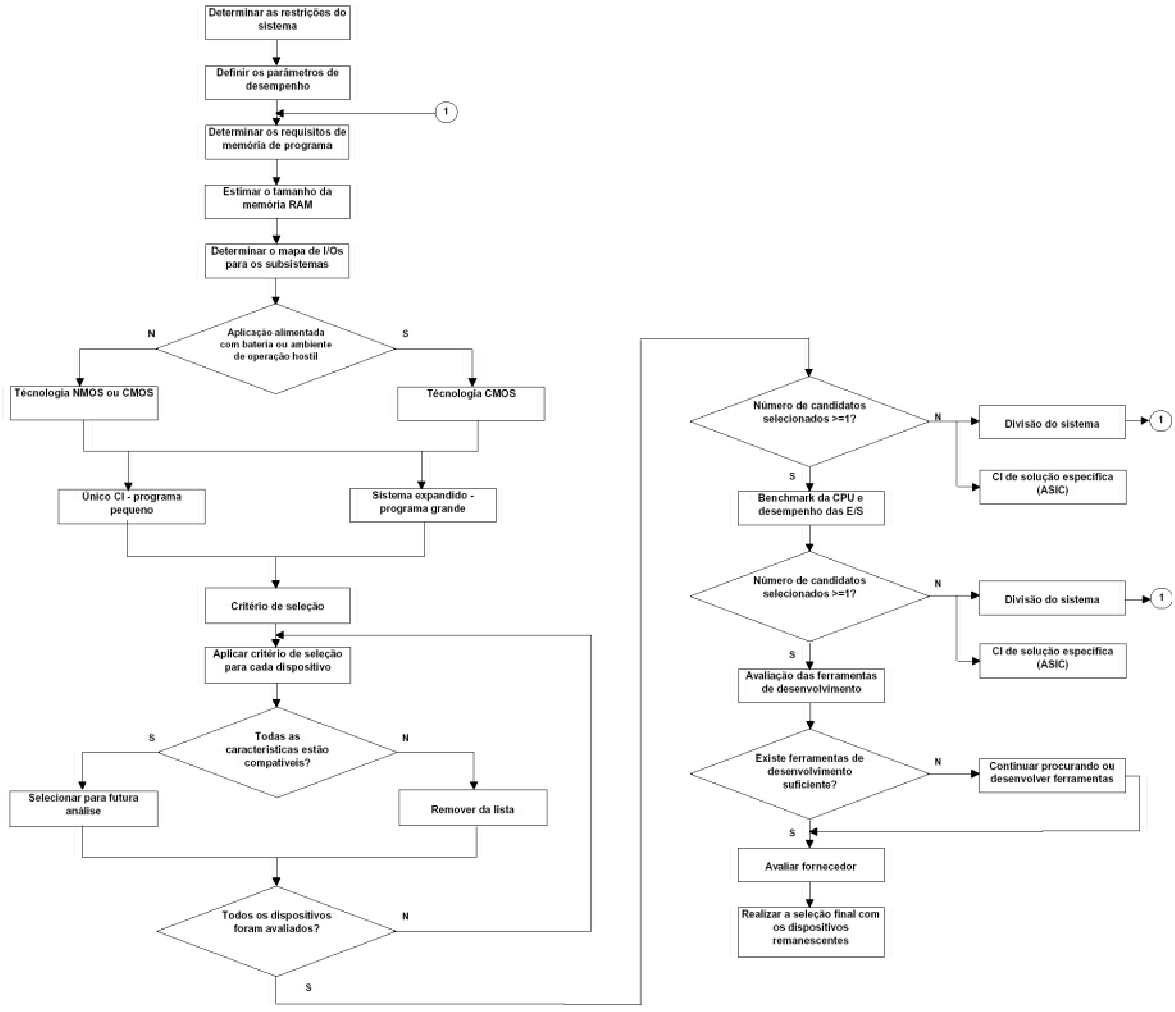


Figura 14: Fluxograma para a seleção de microcontrolador [22]

A terceira definição, são os requisitos de memória para a aplicação, que deve ser dividida em memória de programa e memória de dados. A estimativa da quantidade de memória que será alocada para armazenar o programa é uma parte difícil do processo de

seleção. Não existe uma regra mágica, mas a experiência, unida ao fluxograma e códigos exemplos disponíveis em ferramentas, auxiliam nesse processo. Existem aplicações simples que necessitam apenas 2K de memória, enquanto aplicações complexas exigem 64K de memória ou mais.

A quantidade de memória RAM para ser alocada pelo sistema para armazenar as variáveis, pilhas do sistema, resultados intermediários de um cálculo e conjuntos de dados, é diretamente afetada pela linguagem de programação utilizada, a seleção dos tipos de variáveis e as configurações de hardware. A estimativa de tamanho da memória RAM é obtida por um percentual da ROM (memória de programa) estimada. A regra proposta por [22] é uma razão entre ROM e RAM na faixa de 12 a 20:1.

Durante o processo de seleção, todos os requisitos devem ser mapeados em um diagrama de blocos, o qual não restringirá a seleção em um dispositivo ou fabricante particular, mas identificará características suficientes para prosseguir o processo de seleção.

Após deve-se comparar cada microcontrolador, observando os periféricos e outros aspectos relevantes. Para reduzir a pesquisa a poucos microcontroladores, deve-se pegar um código atual e utiliza-lo para *benchmark* do desempenho da UCP.

O final do processo de seleção não tem aspecto técnico, onde são observados os seguintes pontos do fornecedor:

- amplitude da linha de produtos;
- excelência do fabricante;
- condição financeira;
- prazos de entrega;
- estratégia para segundo fornecedor.

Segundo [22], a seleção de microcontroladores é uma tarefa difícil, envolvendo inúmeras decisões. Não existe um "*check-list*" universal que o projetista possa seguir, apenas recomendações.

4.2 Recomendações Encontradas na Literatura

Foram pesquisados diversos livros e artigos, os quais indicam alguns cuidados a serem tomados e aspectos que devem ser observados no processo de seleção de processadores. As recomendações podem ser divididas conforme a solução de processamento, considerando UCPs em geral, microcontroladores e DSPs.

4.2.1 Recomendações Segundo [12] para Seleção de UCPs em Geral

Sistemas embutidos representam plataformas que geralmente são especificadas para uma tarefa. Isso significa que o projeto do sistema pode ser altamente otimizado, pois o desempenho do equipamento deve ser o suficiente para a tarefa ser executada [12].

Devido a funcionalidade do equipamento ser estreitamente definida, deve-se selecionar a melhor opção do elemento de processamento (UCP) para o projeto. Dadas as inúmeras centenas de opções disponíveis e muitas variações destas opções, selecionar a UCP correta para o projeto pode ser uma tarefa assustadora [12].

Embora selecionar um processador seja uma tarefa complexa que descarta uma simples otimização, podem ser considerados quatro pontos críticos para a escolha final:

1. O processador é eficaz para uma implementação adequada?
2. O processador apresenta desempenho suficiente para a aplicação?
3. Existe sistema operacional disponível para o processador?
4. Existe um conjunto de ferramentas adequadas ao processador?

O processador é eficaz para uma adequada implementação: projetos sensíveis a custo, provavelmente exigem itens de "prateleira" altamente integrados. Aplicações de alto desempenho geralmente exigem tempos de *gate-to-gate* que somente são praticados quando o projeto é fabricado como um único CI, como ASICs ou PLDs [12].

Geralmente o fabricante de um equipamento de controle industrial deve fornecer suporte durante 20 anos. Devido a isso, deve-se selecionar uma processador onde o fornecedor possa garantir a disponibilidade do produto para esse intervalo de tempo.

O processador apresenta desempenho suficiente para a aplicação: no mundo dos projetos embutidos, nem sempre é necessário ter alto desempenho, somente o desempenho suficiente para a aplicação ser executada corretamente. Sistemas embutidos tornam-se mais complexos conforme a tarefa torna-se mais difícil. O gerenciamento de tarefas pelo processador é muito amplo, onde o gargalo muitas vezes não é o poder computacional e sim a combinação da arquitetura com a demanda de tarefas do dispositivo. Devido a isso pode ser difícil uma avaliação de qual processador terá um desempenho melhor para a aplicação [12].

Deve-se tomar muito cuidado na avaliação de processadores tendo como base as comparações, ou *benchmark*, do desempenho. O ideal é uma avaliação utilizando a parte do código mais crítica da aplicação.

Existe sistema operacional disponível para o processador: a seleção do processador também pode depender do sistema operacional disponível para o mesmo. RTOS (*Real Time Operated System* - sistema operacional de tempo real) comerciais devem possuir algumas propriedades que devem ser observadas, como portabilidade, onde o núcleo do RTOS é modificado para diferentes processadores, mantendo a otimização para baixo nível e garantindo o bom desempenho [12].

Existe um conjunto de ferramentas adequadas ao processador: boas ferramentas são essenciais para o sucesso do projeto. Necessita-se no mínimo de um bom compilador e uma boa ferramenta para depuração. Em muitas situações necessita-se de outros equipamentos como emuladores *in-circuit*, simuladores, etc [12].

Segundo [12], as quatro considerações podem ser aplicadas em muitos processos de seleção, em muitos casos a combinação desses critérios não resulta na melhor escolha.

4.2.2 Recomendações Segundo [23] para Seleção de UCPs em Geral

Em [23] a arquitetura de hardware para sistemas embutidos é chamada de plataforma. A idéia é selecionar uma plataforma e desenvolver um software, ajustando a mesma para a aplicação. A plataforma fornece recursos computacionais necessários para execução da aplicação embutida, e a seleção de que hardware deverá ser utilizado na plataforma geralmente depende do tipo de software que será executado na mesma [23].

A seleção da plataforma é uma decisão crítica no projeto de sistemas embutidos. A escolha de uma arquitetura adequada para o projeto em questão traz ganhos em potência e desempenho. Hardware superdimencionado gera altos custos e desperdício de potência. Hardware subdimencionado pode fazer com que a aplicação não execute corretamente [23].

Sistemas embutidos possuem objetivos de alto desempenho, como em aplicações de tempo real, onde restrições temporais são primordiais. Caso não exista capacidade de processamento suficiente para satisfazer os *deadlines* da aplicação, o sistema pode vir a falhar [23].

É necessário considerar o consumo de energia no processo de seleção da plataforma. Aplicações de tempo real necessitam de capacidade computacional, exigindo altas taxas de processamento. Entretanto, toda essa capacidade de processamento exige um consumo de energia considerado, podendo ser crítico em algumas aplicações. Pode-se economizar uma quantidade de energia significativa, construindo uma plataforma que utilize hardware especializado em algumas operações comuns da aplicação [23].

Segundo [23], os diversos sistemas embutidos necessitam diferentes plataformas com características distintas em seu funcionamento, explicando o grande número de diferentes tipos de MCUs, DSPs e DSCs que os fabricantes de semicondutores oferecem.

4.2.3 Recomendações Segundo [13] para Seleção de DSPs

Existe a necessidade de conhecimento total dos requisitos do sistema que se pretende projetar antes de selecionar um dos DSPs disponíveis no mercado. Muitos dispositivos podem ser eliminados com algumas restrições do sistema [13]:

- baixo consumo;
- resolução insuficiente;
- custo.

Após esta primeira etapa ainda restará um número de dispositivos que chamamos de candidatos, exigindo uma nova análise. Será necessário obter todos os dados de cada dispositivo e realizar uma análise detalhada de como o algoritmo e os requisitos de

interface serão adicionados para cada dispositivo. Esse tempo deve ser considerado no desenvolvimento do sistema. O DSP escolhido deve apresentar um conjunto de ferramentas de desenvolvimento com as seguintes características:

- documentação detalhada do projeto;
- ferramenta de desenvolvimento de código em linguagem *assembly* ou linguagem de alto nível;
- ferramenta para testes dos protótipos;
- exemplos de projetos e código, bem como o suporte técnico.

O objetivo é selecionar um DSP que implemente a melhor solução custo-benefício. Para aplicação com alto volume de vendas, faz sentido selecionar o DSP mais barato que consiga realizar o trabalho para a aplicação. Para volume médio existe um *trade-off* entre o custo da ferramenta de desenvolvimento e o custo do DSP. Para aplicações com baixo volume de vendas, faz sentido selecionar um DSP de fácil utilização com uma ferramenta de desenvolvimento de baixo custo [13].

Existe uma grande chance do processo de seleção do DSP ser um processo interativo, descartando o componente selecionado em um primeiro processo de seleção. Isso significa que por motivos de desempenho, tamanho da memória de programa (tamanho de código) ou consumo de energia, o componente selecionado pode ser substituído por outro componente equivalente que atenda as necessidades.

4.2.4 Recomendações Segundo [24] para Seleção de DSPs

Os processadores embutidos são diferentes dos processadores para computadores de uso geral nos seguintes aspectos:

- custo muito restrito;
- em vez de, quanto mais rápido melhor, apenas a velocidade necessária para atender a aplicação;
- em vez de 32 bits, 8 ou 16 bits são suficientes.

Microcontroladores de 8 bits são oferecidos por inúmeros fabricantes, incluindo Motorola, NEC, Hitachi, TI, Intel, Mitsubishi, Lucent Technologies, Philips, Toshiba e Siemens. Entretanto, eles não apresentam o conjunto de instruções do padrão x86, tendo cada fabricante sua própria arquitetura, criando uma grande incompatibilidade [24].

Micros de 8 bits são utilizados em tarefas de controle simples como varredura de teclados e alimentação de displays. Em muitas aplicações de processamento digital de sinais os microcontroladores de 8 bits não apresentam desempenho suficiente, abrindo espaço para os microcontroladores de 16 bits. Caso não se obtenha o desempenho desejado, pode-se considerar os microcontroladores de 32 bits, incluindo micros como ARM, MIPs e Power PC.

Quando o número de bits começa a crescer para atender as necessidades da aplicação, deve-se tomar o cuidado para simplesmente não aumentar o número de bits, mas utiliza-los de forma mais inteligente.

Em processadores embutidos encontra-se uma estrutura de acesso direto a memória chamada DMA *engine* que melhora o desempenho na troca de dados com um dispositivo de E/S. Em sistemas embutidos, o DMA é uma peça chave. Uma estrutura de DMA implementa uma fila circular que sincroniza o processador com o dispositivo de E/S. O DMA deve ser considerado na seleção de um processador embutido, pois posteriormente pode-se encontrar problemas de desempenho com o processador.

Conforme [24], podemos relacionar alguns pontos a serem observados em relação a estrutura DMA:

- conhecer a carga de trabalho e analisar o padrão de acesso necessário para a aplicação;
- procurar um DMA que tenha padrão de acesso suportado por *hardware*;
- procurar um DMA que suporte vários processos ou microprocessador que permita vários DMAs;
- tenha certeza que o DMA pode manter o ritmo com os dispositivos de E/S e memória.

Sistemas embutidos executam aritmética de ponto flutuante ou ponto fixo e utilizam alocação estática de memória com arranjo de tamanho fixo. Esses sistemas executam protocolos e resolvem problemas como o processamento digital de sinal. Existem muitos algoritmos padronizados, sendo utilizados em diferentes aplicações. Em muitas aplicações são utilizadas operações vetoriais, sendo vital o acelerador de instrução MAC (multiplica e acumula).

Conforme [24], um DSP possui algumas diferenças distintas em relação aos microprocessadores embutidos, sendo otimizado para as operações típicas dos algoritmos DSP, ou seja, a principal tarefa é triturar números. Um caminho para seleção de um DSP é determinar quantos MAC concorrentes ele pode executar por ciclo de máquina.

No processo de comparação de processadores embutidos, deve-se ter muito cuidado com o desempenho de um DSP informado pelo fabricante. A melhor opção é a realização de avaliações práticas. Os resultados de *benchmarks* que são publicados apenas ajudam para limitar o universo de seleção, mas não levam a uma decisão final confiável.

4.2.5 Recomendações Segundo [09] para Seleção de Microcontroladores

Em [09] é destacado que existem alguns guias que ajudam no processo de seleção, tendo muitos pontos em comum. A quantidade de informações referente aos microcontroladores pode ser um tanto confusa, mas é possível fazer uma boa escolha utilizando um guia para seleção.

A primeira regra é pensar sobre quais são as principais restrições do sistema. Geralmente um microcontrolador deverá apresentar um certo nível de desempenho. Caso uma rotina deva realizar um cálculo em um determinado tempo, o desempenho do microcontrolador deve ser suficiente para suportar esta restrição.

Nunca deve-se confiar em *benchmarks*, a menos que eles sejam especificados pelo projetista e relevantes para seu sistema. Um caminho simples para determinar se um dado microcontrolador pode atender os requisitos de desempenho do sistema é desenvolver alguns programas que representam as operações criticas do sistema, e então comparar o código para diferentes microcontroladores, determinando o tempo de execução e o

tamanho do código compilado. Isso é mais evidente se o programa for escrito em linguagem de alto nível [09].

Existem vários fatores que são importantes para o desempenho do microcontrolador. Velocidade de barramento e o tempo de ciclo de instrução são muito importantes, mas podem ser insignificantes se não forem avaliados juntos. Idealmente, um barramento de alta velocidade com poucos ciclos por instrução produzem um altíssimo desempenho.

Tamanho de código é muito importante. Embora os desenvolvedores produzam compiladores muito eficientes, sendo melhorados a cada revisão, existe apenas uma parte do compilador que pode ser desenvolvida antes da arquitetura do microcontrolador, tornando-se um fator limitante. Algumas arquiteturas são mais otimizadas para a utilização de linguagem de alto nível, tornando mais amigável a compilação. Geralmente, o conjunto de instruções e os modos de endereçamento, que são fornecidos para os desenvolvedores de compiladores, influenciam diretamente na eficiência do compilador. Uma arquitetura otimizada pode resultar em uma redução significativa no tamanho do código compilado e arquiteturas eficientes resultam em baixo custo [09].

Em certos casos o desempenho não é o fator limitante para o sistema. Muitas vezes o consumo, dimensões da placa eletrônica ou o custo, são os pontos críticos. Para esses casos, o *benchmark* é focado para a principal restrição do sistema. Por exemplo, um microcontrolador pode executar com uma velocidade menor, sacrificando o desempenho, para atender a restrição de baixo consumo do sistema.

Um item muito importante é o conjunto de periféricos, onde o objetivo é selecionar um microcontrolador que tenha os periféricos mais adequados para a funcionalidade exigida pelo sistema. Por exemplo, tipo e tamanho de memória, número de canais de temporizador, conversores A/D, portas de comunicação serial , etc. Se existem periféricos adicionais, os quais não são requeridos, estes podem resultar em um acréscimo no custo do microcontrolador.

Uma questão básica a ser considerada é a determinação de um orçamento de projeto considerando qualquer microcontrolador disponível. Se o custo do microcontrolador está fora do orçamento, todos as outras considerações tornam-se irrelevantes e um

microcontrolador de mais baixo custo deve ser buscado. O ideal é a seleção de uma família de microcontroladores que permita mais adiante um acréscimo no desempenho, oferecendo versões com mais capacidade de memória, velocidade maior e periféricos mais sofisticados, porém com compatibilidade pino a pino.

Um fator muito importante a ser considerado na seleção do microcontrolador correto é a disponibilidade do mesmo, pois projetar um sistema utilizando um microcontrolador que ainda não está em produção ou tem muitos erros (*bugs*) pode ser um risco. Um aspecto muito importante é avaliação do fabricante considerando:

- tempo de atuação no mercado;
- capacidade e prazo de entrega;
- suporte técnico.

As ferramentas de desenvolvimento também tem uma parte importante na seleção do microcontrolador. Ferramentas de hardware e software são exigidas para o desenvolvimento eficiente da aplicação. Geralmente os fabricantes de microcontroladores indicam a melhor solução de ferramenta de desenvolvimento conforme a necessidade [09].

4.2.6 Seleção de Processador em um Ambiente de Projeto de Sistemas Embutidos [25]

Com o aumento dos requisitos de desempenho das aplicações, os sistemas estão ficando mais complexos, exigindo uma rigorosa metodologia de projeto com a necessidade de uma ferramenta de projeto completa e abrangente.

A ferramenta proposta por [25] tem o objetivo de desenvolver uma metodologia de projeto de sistemas embutidos para aplicações de processamento de imagem e visão. Dada a especificação de um sistema, seguindo a metodologia e com a ajuda da ferramenta, o projetista será capaz de sintetizar as limitações da especificação do sistema. A avaliação do processador é um dos componentes da ferramenta, onde através da especificação comportamental da aplicação e uma faixa dos processadores alvo, os processadores são avaliados para estas especificações, gerando resultados de estimativa de desempenho. Esta avaliação é realizada em dois estágios:

1 – eliminação dos processadores inadequados com base na combinação imprópria entre os requisitos da aplicação e as características da arquitetura do processador;

2 – avaliação dos processadores restantes através de uma rápida avaliação do desempenho da aplicação em número de ciclos.

Inicialmente é exigida uma identificação da arquitetura do processador e características da aplicação através de um conjunto de parâmetros.

A metodologia utiliza as entradas da aplicação e a especificação da arquitetura do processador. A aplicação é convertida para algumas representações intermediárias utilizando uma interface da ferramenta. Simultaneamente as características da arquitetura do processador devem ser carregadas. A etapa avaliadora utiliza as duas entradas e gera estimativas como tamanho do código e tempo de execução, que são anexadas a representação intermediária da faixa de processadores, como ilustra a figura 15.

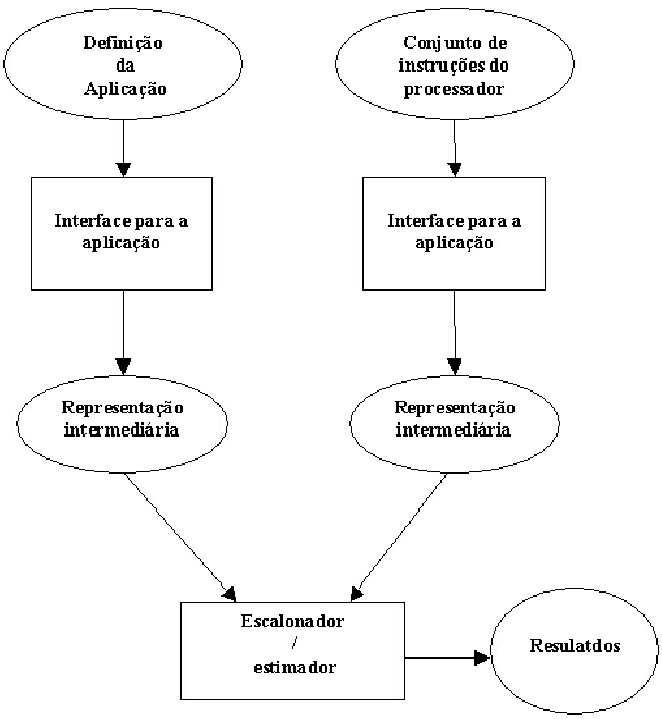


Figura 15: Metodologia da ferramenta proposta por [25]

4.2.6.1 Representação da Arquitetura

As características da arquitetura do processador são a função chave para o casamento da aplicação com o processador. Utiliza-se um formato apropriado para a

descrição do processador. Este formato é uma adaptação do formato de descrição de máquina que é utilizado pela Philips para descrever o processador Trimédia. A descrição do processador é formada por:

- tipos diferentes de unidades funcionais, com suas características;
- número de unidades funcionais de cada tipo;
- número de registradores;
- número das operações em cada instrução;
- restrições no mapeamento das operações;
- operações da carga e descarga concorrente.

Em arquiteturas com unidades funcionais múltiplas, encontra-se também as seguintes características:

- latência em uma operação depende do desempenho da unidade funcional para a mesma;
- latência depende do desempenho das operações existentes independendo de unidade funcional mapeada.

4.2.6.2 Representação da Aplicação

Para apresentar a comparação entre a aplicação e a arquitetura do processador, divide-se o problema em duas etapas. Na primeira etapa, extrai-se certos parâmetros da aplicação, independente da arquitetura do processador alvo. Estes parâmetros indicam algumas características do processador adequado. Portanto, as arquiteturas de processador que não combinam com as requisitos da aplicação são eliminadas. Na segunda etapa, seleciona-se os processadores que serão avaliados, dando uma estimativa de desempenho. Isto é feito através de um escalonador de arquiteturas que faz o planejamento da aplicação com uma arquitetura de processador capturada no arquivo de descrição [25].

Os parâmetros que influenciam o desempenho da aplicação são relacionados com as características relevantes da arquitetura do processador. Através de uma análise destes

parâmetros com as características da arquitetura, poderia ser possível uma avaliação rápida e adequada de um processador.

4.2.6.3 Escalonador de Arquitetura

É o segundo estágio do processo de avaliação. Neste estágio os aspectos da arquitetura e as características da aplicação são consideradas independentes. Mas no processo para estimar o desempenho em números de ciclos de máquina ambos são analisados juntos. O escalonador desenvolvido utiliza o arquivo que descreve a arquitetura do processador para avaliar o número de ciclos de máquina. A característica chave do escalonador é a sua capacidade de analisar a flexibilidade do conjunto de instruções para cada processador.

O escalonador faz uma passagem nos parâmetros da aplicação, gerando dependência dos dados. Depois é anexada uma prioridade em cada operação, baseada no peso da dependência da operação. A prioridade é utilizada por uma lista do escalonador para selecionar as operações disponíveis. O escalonador faz a leitura do arquivo de descrição do processador para determinar as operações para as unidades funcionais.

4.2.6.4 Ambiente e Resultados

Primeiramente é realizada uma padronização da aplicação, estimando a freqüência dos vários blocos do programa em C. Isto é seguido por dois estágios de avaliação:

- estágio independente de arquitetura;
- estágio dependente de arquitetura.

No estágio independente de arquitetura, parâmetros são extraídos da aplicação. Estes parâmetros, junto com uma faixa de processadores alvo, constituem a entrada do selecionador de processador. O selecionador de processador rejeita os processadores que não atendam as restrições descritas na aplicação. A saída é um pequeno conjunto de processadores. No estágio dependente de arquitetura, o escalonador utiliza o perfil de saída do selecionador de processador com a aplicação e o arquivo descritor de arquitetura do processador. Esta combinação estima o desempenho de cada processador, em número de ciclos, para a aplicação dada.

4.3 Avaliação de Desempenho

Segundo [26], para melhorar o desempenho de um *software* que roda em um determinado *hardware*, precisa-se conhecer quais fatores do *hardware* influenciam o desempenho global do sistema. Entre os fatores do *hardware* que tem muita influência pode-se destacar:

- utilização por parte do compilador das instruções de máquina na geração do código de um programa;
- como o hardware implementa as instruções;
- como a memória e as entradas e saídas se comportam na execução de programa.

Quando executamos um mesmo código em processadores diferentes, podemos dizer que o melhor processador é aquele que executa o código mais rapidamente. Ele apresenta o menor tempo de resposta, que pode ser definido como o tempo decorrido entre o início e o final da execução do programa. Esse tempo também é denominado tempo de execução.

Considerando que o compilador gera instruções que devem ser executadas pelo processador, o tempo de resposta depende do número de instruções do programa. Uma maneira de conceituar o tempo de execução é torná-lo igual ao número de instruções multiplicado pelo tempo médio de execução de cada instrução [26].

Além do tempo de execução, utiliza-se o termo *throughput*, que determina uma quantidade de trabalho executado em um determinado intervalo de tempo, para medição do desempenho. O *throughput* para processadores é medido em MIPS (milhões de instruções por segundo) ou em MFLOPS (milhões de operações com ponto flutuante por segundo) [27].

Com o estabelecimento de uma relação entre tais métricas, pode-se determinar o efeito que uma mudança no projeto tem sobre o desempenho. Generalizando, a diminuição no tempo de resposta quase sempre melhora o *throughput* e vice-versa.

Em função da dificuldade de medir e de comparar o desempenho de processadores com certo grau de precisão, muitos projetistas e pesquisadores tentam desenvolver métodos

para estimar o desempenho que não dependam de medidas do tempo de execução. Esses métodos são utilizados com freqüência para comparar diferentes arquiteturas de maneira a decompor os efeitos do software do sistema e obter uma conclusão sobre o desempenho. Todas as alternativas propostas para substituir o tempo como métrica do desempenho levam a conclusões erradas, resultados distorcidos ou a implementações incorretas [26].

Uma das mais conhecidas alternativas para o tempo como métrica do desempenho é o MIPS. Conforme [12], o MIPS foi um termo criado para o minicomputador VAX11/780, que foi a primeira máquina que poderia executar 1 milhão de instruções por segundo. Uma instrução é uma medida que não tem relação com a escala de trabalho de diferentes arquiteturas. Por exemplo, considerando um microcontrolador de arquitetura RISC executando 1500 instruções e um microcontrolador de arquitetura CISC executando 1000 instruções, não podemos afirmar qual tem maior desempenho. A menos que se compare VAX com VAX, a unidade MIPS não significa muito.

Com o desenvolvimento da indústria dos supercomputadores, definida como de alto desempenho, especialmente na execução de instruções de manipulação de números em ponto flutuante, os tempos médios de execução de instrução, e mesmo os MIPS, mostravam-se métricas pouco apropriadas, daí a invenção dos MFLOPS.

O MFLOPS foi desenvolvido com a finalidade de medir o desempenho com respeito as operações de ponto flutuante. Considerando que o MFLOPS é baseado em operações do programa, e não em instruções, este índice tem um apelo muito mais forte que o MIPS como comparação entre processadores diferentes. A chave para isso é que o mesmo programa, executando em processadores diferentes, pode executar um número diferente de instruções, mas irá sempre executar a mesma quantidade de operações em ponto flutuante. Mas o MFLOPS não é uma medida segura, pois o conjunto de operações em ponto flutuante não tem sua consistência assegurada para processadores diferentes, e o número real de operações em ponto flutuante pode variar [26].

Um exemplo para ilustrar o problema com MFLOPS é a comparação entre os processadores Cray-2 e o Motorola 68882. O Cray-2 não tem instrução de divisão, já o Motorola 68882, possui instrução de divisão, raiz quadrada, seno e coseno. Portanto, no Cray-2 são necessárias várias operações em ponto flutuante para a realização de uma

divisão, enquanto no Motorola 68882, a mesma operação é realizada em apenas uma operação.

Na medida que os processadores se tornam mais sofisticados, fica claro que nem MIPS nem tempo de execução podem ser calculados com base no conjunto de instruções e características da arquitetura. O mais certo à se fazer é desenvolver um conjunto de programas, *benchmark*, o mais semelhante com a aplicação real, para realizar uma comparação de desempenho.

4.3.1 *Benchmark*

O conjunto de programas utilizados pelo sistema é chamado de carga de trabalho ou *workload*. A carga de trabalho utilizada para avaliação de arquiteturas é denominada carga de trabalho de teste ou *test workload*, sendo dividida em carga de trabalho real e sintética. A carga de trabalho real é observada quando o sistema está sendo utilizado em operações normais. Porém, a carga real não é utilizada como carga de trabalho de teste pois a mesma não pode ser repetida nas mesmas condições. A carga de trabalho sintética apresenta características semelhantes a carga de trabalho real e pode ser aplicada de maneira repetitiva e controlada. É utilizada em estudos, pois apresenta uma representação ou modelo da carga de trabalho real, podendo ser facilmente modificada.

O Processo de comparação de desempenho entre processadores via medições é chamado de *benchmarking*, e a carga de trabalho utilizada nas medições é chamada de *benchmark* [27].

Um b*enchmark* é formado por um conjunto de programas que tentam retratar as características de uma aplicação completa, possibilitando a estimativa de desempenho. Segundo [12], um verdadeiro b*enchmark* requer um balanceamento cuidadoso dos requisitos do sistema e variáveis. O desempenho de um processador em uma aplicação específica pode ser diferente para outras aplicações. Deve-se considerar inúmeros aspectos quando determina-se qual processador tem desempenho melhor ou pior em um teste de b*enchmark*.

Conforme [26], os *benchmarks* mais utilizados são o Whetstone e o Dhrystone.

O *benchmark* de Whestone é formado por 11 módulos projetados para verificar a freqüência dinâmica das operações de programas em ALGOL. Avalia características do processador como endereçamento de arranjo, operações com ponto fixo e flutuante, chamada de subrotina e passagem de parâmetros. Mais tarde ele foi convertido para FORTRAN e outras linguagens. Os resultados do *benchmark* de Whestone são medidos em KWIPS (Kilo instruções de Whestone por segundo). Os módulos foram projetados para minimizar o impacto das otimizações conhecidas em compiladores, reduzindo a variação do tempo de execução da carga de trabalho no processador [27].

O *benchmark* de Dhrystone apresenta muitas chamadas de procedimentos, sendo implementado em várias linguagens, onde a versão em C é a mais utilizada. Conforme [12], o programa em C compilado gera aproximadamente 2000 linhas de código *assembly*, sendo independente dos serviços do sistema operacional. Este *benchmark* foi calibrado utilizando a máquina VAX, sendo que o VAX pode executar 1.757 laços do código de Dhrystone em 1 segundo, ou seja, 1.757 é igual a 1 DIPS (instruções de Dhrystone por segundo). O problema com o *benchmark* de Dhrystone é que o compilador pode ser otimizado para gerar um código mais eficiente, melhorando a estimativa do desempenho.

A grande desvantagem destes *benchmarks* é que não podem ser utilizados como aplicação, pois apresentam cálculos sem objetivo específico. Outra problemática é que otimizações em compiladores ou no hardware conseguem melhorar o desempenho destes *benchmarks*.

Para solucionar a necessidade de dados tangíveis para o processo de seleção, as empresas de semicondutor, compiladores e seus clientes criaram um *benchmark* mais eficiente chamado de EEMBC (EDN *Embedded Microprocessor Benchmark Consortium*). O EEMBC consiste em testes específicos para cada tipo de aplicação, sendo dividido em 46 *benchmarks* dentro de 5 tipos de aplicações [12].

Além do EEMBC, o SPEC (*System Performance Evaluation Cooperative*) é, segundo [26], o *benchmark* para processador mais conhecido atualmente. A SPEC é uma cooperativa americana formada por diversos fabricantes de computadores, com o objetivo de melhorar a medida e as informações disponíveis sobre o desempenho de processadores, através de um processo mais controlado e buscando determinar um *benchmark* mais realista. A última versão dos *benchmarks* SPEC é o conjunto SPEC95, que consiste em 8

programas que manipulam números inteiros e 10 que tratam com números em ponto flutuante.

4.4 Avaliação das Metodologias e Recomendações

As metodologias e recomendações descritas acima, apontam inúmeros aspectos relevantes para o processo de seleção do processador para sistemas embutidos. A tabela 1 procura compilar esses aspectos, classificando as metodologias e recomendações e apontando os aspectos mais relevantes. Para cada aspecto é atribuído uma nota conforme a seguinte classificação:

0 (fraco): não mencionado;

5 (médio): mencionado como consideração;

10 (forte): mencionado como requisito importante.

Aplicando as notas aos aspectos, é possível identificar a metodologia ou a recomendação que apresenta o maior número de aspectos citados e a maior pontuação, considerando este o mais completo no processo de seleção do processador. Os aspectos mais relevantes são aqueles citados mais vezes nas metodologias e recomendações pesquisadas.

As recomendações encontradas em [09] e a metodologia proposta em [22], contemplam o maior número de aspectos (13), porém a metodologia [22] cita alguns aspectos apenas como considerações, totalizando menos pontos que as recomendações de [09] que totaliza 95 pontos.

Analisando a tabela 1, é possível identificar os aspectos mais relevantes no processo de seleção do processador. Os aspectos mais citados são:

benchmark (5 citações);

custo (4 citações);

ferramenta de desenvolvimento de hardware e software (4 citações).

Aspectos relevantes / Metodologias e recomendações		Metodologia [22]	Segundo [12]	Segundo [23]	Segundo [13]	Segundo [24]	Segundo [09]	Citações
Conhecimento dos requisitos e restrições do sistema	Baixo consumo	0	0	**10**	**5**	0	**5**	**3**
	Resolução	**5**	0	0	**5**	**10**	0	**3**
	Tempo real	**10**	0	**5**	0	0	**5**	**3**
	Custo	0	0	**5**	**5**	**5**	**10**	**4**
Memória	Estendida	**5**	0	0	0	**5**	0	**2**
	On Chip	**5**	0	0	0	0	0	**1**
	De programa	**10**	0	0	0	0	0	**1**
	De dados	**10**	0	0	0	0	0	**1**
Família de processadores		0	0	0	0	0	**10**	**1**
MAC		0	0	0	0	**10**	0	**1**
DMA		0	0	0	0	**10**	0	**1**
Periféricos		**5**	0	0	0	**5**	**10**	**3**
Benchmark		**10**	**10**	0	**5**	**10**	**10**	**5**
Sistema operacional		0	**10**	0	0	0	0	**1**
Fornecedor do processador	Suporte técnico	**5**	0	0	0	0	**10**	**2**
	Ex. aplicações	**5**	0	0	0	0	**5**	**2**
	Fornecimento	**5**	**5**	0	0	0	**10**	**3**
Ferramenta de desenvolvimento	Hardware	**5**	**10**	0	**5**	0	**5**	**4**
	Software	**5**	**10**	0	**5**	0	**5**	**4**
	Suporte técnico	0	0	0	0	0	**5**	**1**
	Ex. aplicações	0	0	0	**10**	0	0	**1**
	Atualizações	0	0	0	**5**	0	**5**	**2**
Número de aspectos citados		**13**	**5**	**3**	**8**	**6**	**13**	
Total de Pontos		**85**	**45**	**20**	**45**	**45**	**95**	

Tabela 1: Metodologias e recomendações descritas na literatura

O aspecto mais citado nos métodos e sugestões pesquisados, segundo a tabela 1, é o *benchmark*, o qual revela o desempenho do processador. Os sistemas embutidos executam tarefas específicas, onde o desempenho do processador deve ser o suficiente para a execução perfeita da mesma. Dependendo do grau de complexidade da tarefa, processadores de 4 ou 8 bits apresentam-se aptos e com desempenho suficiente, mas o problema é a grande variedade de dispositivos oferecidos por inúmeros fabricantes. Neste contexto, surgem os *benchmarkings* que tentam auxiliar o projetista no processo de seleção. Porém, os índices de desempenho revelados devem ser analisados com cautela, observando o tipo de *benchmark* empregado na avaliação. Os resultados podem ser encarados como referência e não devem ser empregados como item conclusivo no processo de seleção, a não ser se o *benchmark* utilizado seja a própria aplicação ou alguma rotina critica da aplicação.

O custo surge como segundo aspecto relevante, delineando os limites dos processadores que podem ser utilizados em determinado sistema embutido. O grande objetivo é selecionar o melhor processador dentro de determinada faixa de preço, ou seja, o processador com melhor custo/beneficio. Muitas vezes, o custo do processador pode inviabilizar o desenvolvimento de um sistema embutido.

Atualmente, fabricantes oferecem processadores de 16 bits com o custo semelhante ou até mesmo igual aos processadores de 8 bits. Sem dúvida, seria melhor aplicar um processador de 16 bits com um custo de 8bits. Entretanto, o custo das ferramentas de desenvolvimento, tanto de *hardware* como de *software*, devem ser analisados.

A preocupação com as ferramentas de desenvolvimento, tanto de hardware como de software, aparece em seguida como aspecto mais relevante. Escolher a ferramenta de desenvolvimento de software para o processador não é uma decisão menos crítica que selecionar o próprio processador. Segundo [28], o desenvolvimento do software é o equivalente de 50 a 75% do custo de um projeto embutido.

4.4.1 Ferramentas de Desenvolvimento

As ferramentas de desenvolvimento irão influenciar diretamente o tempo de desenvolvimento e consequentemente no lançamento do sistema embutido. As ferramentas

de software podem ser classificadas em: compiladores, simuladores, software para gravação do programa e ambientes de desenvolvimento integrado (IDE).

O compilador faz a tradução de um conjunto de comandos, de baixo ou alto nível, para a linguagem de máquina de um determinado processador. Inicialmente os processadores eram programados diretamente em linguagem de máquina, dificultando o desenvolvimento do software. Com o desenvolvimento dos processadores, houve a necessidade de melhorar a forma de programação dos mesmos, surgindo o compilador para a linguagem de montagem do processador, mais amigável, porém considerada de baixo nível. Com o objetivo de diminuir a distância entre o usuário e a arquitetura do processador, surgiram os compiladores para as linguagens de alto nível, facilitando o desenvolvimento e criando alguma portabilidade para o software.

Os ambientes de desenvolvimento integrado, facilitam o desenvolvimento do software embutido, pois apresentam todas as ferramentas necessárias para o desenvolvimento da aplicação, como compilador, simulador e software para gravação. Existem ambientes de desenvolvimento integrado que oferecem a mesma interface para diferentes famílias de processadores de um mesmo fabricante, como o Mplab (Microchip), Code Warrior (Freescale), HEW (Hitachi) e outros. Existe um ambiente integrado desenvolvido pela IAR *Systems* que mantém a mesma interface para diferentes processadores de diferentes fabricantes.

As ferramentas de hardware podem ser divididas, conforme [28], em dois grandes grupos: ferramentas invasivas e não invasivas. As ferramentas invasivas podem afetar o funcionamento do processador, causando pequenas variações no tempo de execução e problemas no funcionamento de periféricos e das interrupções. As ferramentas não invasivas são cuidadosamente projetadas para replicar todas as funções do processador, garantindo o funcionamento do mesmo em qualquer condição de emulação do sistema.

4.5 Conclusão

Com base na pesquisa realizada, conclui-se que não foi encontrado na literatura uma metodologia completa para seleção de processadores. Porém existem inúmeras recomendações onde são abordados itens que devem ser observados nesse processo, destacando o *benchmark*, o custo do processador e as ferramentas de desenvolvimento. A

tabela 1 procura compilar todos os itens e classificar as metodologias e recomendações, identificando pontos em comum e delineando uma proposta de metodologia para seleção de processadores para sistemas embutidos.

Nenhuma metodologia pesquisada faz referência explicita à especificação do sistema embutido como ponto inicial para a seleção. Elas descrevem diretamente os aspectos que devem ser observados para determinar o processador. Entretanto, fica claro que a seleção do processador depende de uma boa especificação do sistema embutido a ser desenvolvido, pois é através desta que as características do processador são definidas, atreladas a um objetivo de custo para todo o sistema.

Uma metodologia completa é aquela que contempla o sistema embutido, utilizando uma sintaxe padrão para a modelagem tanto da sua parte de *software* como da sua parte de *hardware*, documentando todas as restrições e características do sistema. Partindo deste modelo do sistema embutido, é possível determinar as restrições e características que o processador deverá possuir. Também deve-se considerar características externas ao sistema que influenciam diretamente o custo e o tempo de desenvolvimento, como experiência do projetista com alguma família de processador, verificar se a empresa possui ferramentas de desenvolvimento de *hardware* ou de *software* para alguma família de processador e a possibilidade de reutilização de *software*.

Após a especificação do sistema embutido deve ser realizado o particionamento do *hardware* e do *software*. Considerando as etapas do projeto de sistemas embutidos, definidas no *co-design*, a seleção do processador com melhor custo/beneficio deve ser executada após a definição das partes que serão implementadas por *hardware* e por *software*, ou seja, após o particionamento. As partes que serão implementadas por software utilizam um processador, o qual deve oferecer todos requisitos exigidos pela aplicação e desempenho suficiente para executa-la.

Uma proposta de seleção de processadores pode ser composta das seguintes etapas:

- especificação e modelagem do sistema embutido;
- identificação das restrições e características do processador;
- identificação da características externas ao desenvolvimento do sistema;

- pesquisa e seleção dos processadores com base nas restrições;

- realização de *benchmark* utilizando o modelo de software do sistema.

Com o aumento dos requisitos de desempenho das aplicações, os sistemas embutidos estão ficando mais complexos, exigindo uma rigorosa metodologia de projeto com a necessidade de uma ferramenta de projeto completa e abrangente.

Esta ferramenta além de permitir a modelagem das partes de *hardware* e *software* deve possuir mecanismos para modelagem dos processadores selecionados na pesquisa. Também deve executar o *benchmark* dos processadores selecionados utilizando o modelo do software do sistema como carga de trabalho e classificando-os conforme desempenho medido.

Após encontrado o processador com melhor custo/beneficio/desempenho, deve-se analisar as ferramentas de desenvolvimento oferecidas para o mesmo, verificando a existência de material de referência como manuais e exemplos de aplicação e a existência de suporte técnico. Através desta verificação evita-se selecionar algum processador, cujo custo/benefício é o melhor, mas as ferramentas de desenvolvimento além de caras não apresentam referências de ajuda ou suporte técnico no desenvolvimento.

Também deve ser observado se o processador selecionado possui uma família (grupos de elementos com características semelhantes) que possibilite sua substituição por outro processador compatível pino a pino mas com capacidade maior de memória ou algum periférico melhorado. Isso possibilita melhoramentos no *software* do sistema embutido sem alterações no *hardware* do mesmo.

Capítulo 5

Usando SysML na Modelagem de Sistemas Embutidos

Em geral, a primeira etapa do desenvolvimento de um sistema embutido é especificar seus requisitos funcionais e não funcionais. Para ajudar a entender e organizar o conjunto de requisitos de uma maneira sistemática, pode-se utilizar vários modelos conceituais. Dessa forma, um processo de desenvolvimento ou metodologia é freqüentemente visto como uma seqüência de passos que transformam uma especificação abstrata em uma especificação detalhada que pode ser fabricada. Assim, a entrada de uma metodologia é a especificação dos requisitos de um sistema, e a descrição final ou saída é a implementação.

Existem diversas maneiras de descrever o comportamento de um sistema. No caso do comportamento ser descrito em linguagem natural o sistema está descrito de maneira informal, podendo ter ambigüidades, o que dificultará o tratamento computacional automatizado. Por esse motivo é essencial inserir modelos formais ou semiformais, que sejam capazes de especificar os requisitos, viabilizando o emprego de métodos automatizados de projeto de sistemas computacionais, além de prover uma visão mais abstrata do sistema [30].

Antes de iniciar a etapa de modelagem do sistema é necessário a execução de uma etapa muito importante e fundamental, a etapa de análise de requisitos. A análise de requisitos é uma etapa informal, onde as características do sistema são descritas em

linguagem natural em documento de texto de forma abstrata, sem preocupação com os detalhes e regras metodológicos de uma especificação.

A modelagem de sistemas embutidos não é uma tarefa simples diante da heterogeneidade de seus componentes e diversidade de aplicações. Atualmente vários estilos de modelagem são utilizados dependendo da aplicação. Os modelos de especificação podem ser divididos em dois tipos:

- modelagem homogênea;
- modelagem heterogênea.

Modelagem homogênea permite que um único modelo possa ser utilizado desde a especificação inicial até mesmo após o particionamento entre *hardware* e *software*. Como exemplo pode-se citar: modelo orientado a estados, modelo orientado a atividades e modelo orientado a estrutura [01].

Devido a dificuldade de encontrar um modelo com todas as características necessárias para a descrição de sistemas embutidos, utiliza-se uma combinação de vários modelos, caracterizando uma abordagem heterogênea. Esse tipo de modelagem é encontrado em linguagem de descrição e programação, na tentativa de aumentar a capacidade de modelagem destas linguagens. Como exemplo pode-se citar o modelo orientado a objetos, linguagem de programação e linguagem de descrição de *hardware* [01].

5.1 Motivação para SysML

Devido aos avanços tecnológicos da indústria de semicondutores, e consequentemente a redução do custo dos processadores, os sistemas embutidos ganharam espaço em praticamente todos os segmentos de produtos industriais e residenciais. Esses produtos, mais aprimorados, possuem um *software* embutido que é responsável pelo funcionamento e controle do mesmo. Com o aumento da complexidade e variedade de aplicações para estes sistemas embutidos, surge a necessidade de colaboração interdisciplinar no processo de desenvolvimento, envolvendo conhecimento nas áreas de engenharia de software embutido, engenharia mecânica, elétrica e eletrônica. A integração

destas áreas resultará no sistema embutido como produto. Entretanto, cada área utiliza metodologias e ferramentas de desenvolvimento diferentes. A engenharia de sistema (*system engineering*) emprega diferentes técnicas para integrar concorrentemente as diferentes áreas de engenharia.

Segundo [31], para obter as melhores oportunidades, produtividade e qualidade no desenvolvimento do *software* embutido, muitas empresas estão utilizando, em situações específicas, tecnologias de engenharia de *software*. O maior problema é que a maioria destas tecnologias de desenvolvimento de *software* disponíveis não consideram as necessidades específicas para o desenvolvimento de sistemas embutidos, como restrições temporais, limitação de memória, consumo de energia, hardware específico, etc.

Devido a falta de uma ferramenta de engenharia de *software* embutido, atualmente, segundo pesquisa realizada por [31], é comum no projeto de *software* embutido, o emprego de diagramas com formato livre e diagramas de blocos que lembram a UML (*Unified Modeling Language*), diagrama de fluxo de dados (DFD) e fluxogramas. É utilizado uma ferramenta de desenho de propósito geral para fazer os diagramas do projeto de *software*. A falta de sintaxe e semântica padronizada proporciona a interpretação errada dos diagramas. Essa má interpretação é mais evidente quando o projeto é interdisciplinar, onde os membros possuem formações distintas.

Uma ferramenta de modelagem específica para sistemas embutidos poderia evitar as ambigüidades entre as diferentes áreas envolvidas no desenvolvimento. Isto através de uma sintaxe e semântica padronizada para descrição das funcionalidades de *hardware* e de *software*. Diante desta lacuna, a OMG (*Object Management Group* ou Grupo de Gerenciamento de Objetos) propôs a criação de uma linguagem específica para atender os requisitos da engenharia de sistemas. A proposta desta nova linguagem de modelagem, denominada SysML (*System Modelling Language* ou Linguagem de Modelagem de Sistema), é uma adaptação da UML 2.0 que possibilita em uma única especificação a descrição completa do sistema, incluindo *hardware* e *software*.

5.2 Linguagem de Modelagem SysML

SysML é uma linguagem de modelagem de propósito geral para aplicações de engenharia de sistema, fornecendo um padrão de descrição para uma grande variedade de

sistemas complexos. Esses sistemas podem incluir *hardware*, *software*, dados, métodos, pessoal e instrumentos. Além de fornecer um padrão de modelagem, a SysML busca o aperfeiçoamento da qualidade do sistema, assegurando a troca de informações e diminuindo a distância semântica entre a engenharia de sistemas, *software*, e outras áreas da engenharia.

De maneira similar como a UML unificou as linguagens de modelagem utilizadas na industria de *software*, SysML pretende unificar as diversas linguagens de modelagem utilizadas atualmente por engenheiros de sistema.

A SysML está sendo definida com base na UML 2, utilizando em parte sua sintaxe e semântica. Como conseqüência, engenheiros de sistemas que modelam com SysML e engenheiros de *software* que modelam com UML 2, estarão aptos a colaborarem intensivamente no modelos do *software* e do sistema. Isso irá melhorar a comunicação entre os vários projetistas que participam do desenvolvimento do sistema, promovendo a interoperabilidade entre as ferramentas de modelagem [32].

A linguagem SysML está fundamentada em duas normas de interoperabilidade, a ISO AP-233 – *data interchange standard for system engineering tools* (Padrão de troca de dados para ferramentas de engenharia de sistemas) e a OMG XMI 2.0 – *model interchange standard for UML 2.0 modeling tools* (Padrão de troca de modelo para ferramentas de modelagem UML 2).

A linguagem de modelagem SysML está sendo desenvolvida por um consórcio formado pela OMG e um grupo de empresas. Atualmente a SysML encontra-se disponível na versão 0.9, mas existem propostas para atualização desta versão. A dissertação utiliza a especificação 0.9 da SysML para modelagem dos sistemas embutidos.

SysML reutiliza e estende muitos dos pacotes da UML 2. Um pacote é um mecanismo de propósito geral utilizado para a organização de elementos em grupos. O metamodelo da UML define pacotes para as partes estrutural, comportamental e construções auxiliares, como os perfis de construções para personalização da linguagem. O pacote da parte estrutural inclui Classes, Estruturas Compostas, Componentes e Estratégias. O pacote da parte comportamental inclui Ações, Atividades, Interações, Máquina de Estados e Casos de Uso.

Como a linguagem SysML reutiliza um subconjunto da UML 2 e cria extensões para atender os requisitos específicos da engenharia de sistemas, muitos pacotes do metamodelo da UML não foram utilizados, pois não foram considerados relevantes para as aplicações da engenharia de sistema. São incluídos na SysML sem modificações os seguintes pacotes do metamodelo: Máquina de Estados, Interações e Casos de Uso. Outros pacotes como Atividades, Classes e Auxiliares são reutilizados com adição de algumas extensões. Novos pacotes como Requisitos, Paramétrico e Alocação, estão sendo adicionados para sustentar novas construções. Os pacotes Comportamento comum e Perfil são importados para dentro da SysML [32]. A figura 16 ilustra a estrutura de pacotes da linguagem SysML.

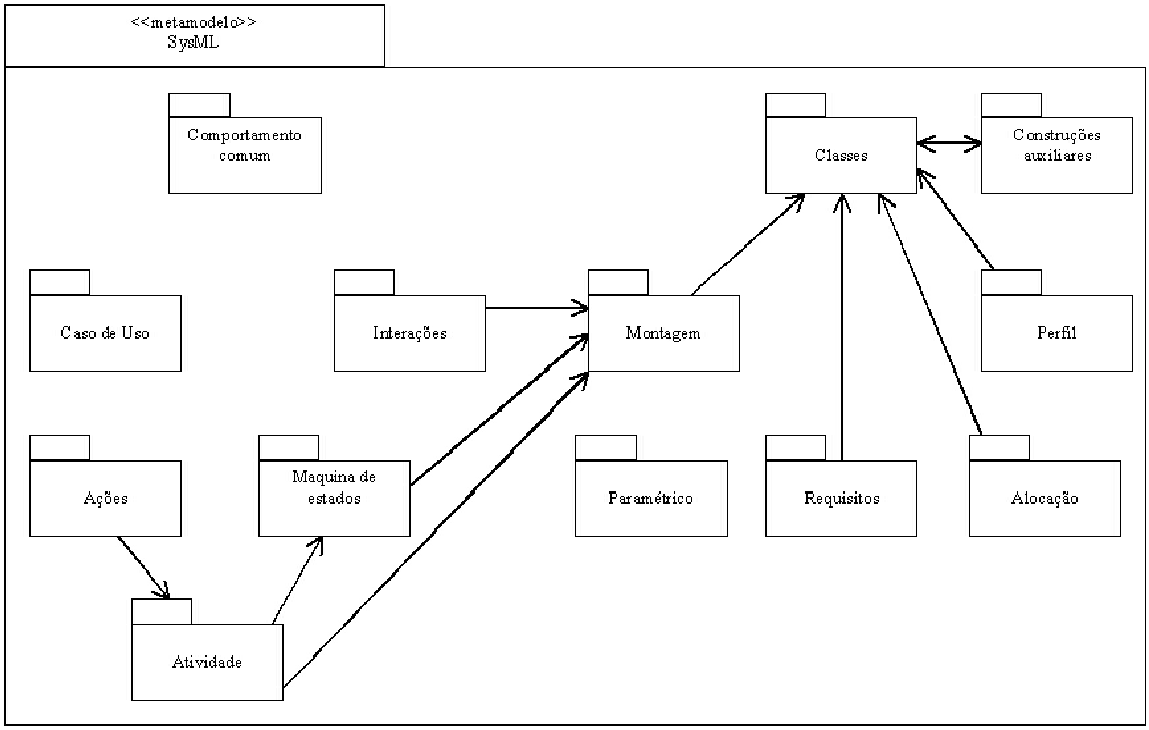


Figura 16: Estrutura de pacotes da SysML

Modelar um sistema complexo é uma tarefa extensa, sendo necessária a descrição de vários aspectos diferentes, incluindo o funcional (estrutura estática e interação dinâmica), o não funcional (tempo de processamento, confiabilidade, produção) e o organizacional (organização do trabalho, mapeamento e código). Ao fazer uma modelagem, cria-se uma simplificação da realidade para entender melhor o sistema em desenvolvimento. Um modelo é uma abstração semanticamente fechada de um sistema, o

que significa que representa uma simplificação completa da realidade, criada com a finalidade de permitir uma melhor compreensão a respeito do sistema [35].

Assim como a UML, a SysML fornece múltiplas visões do sistema a ser modelado, analisando-o e modelando-o sob diversos aspectos, utilizando diferentes diagramas. Cada diagrama analisa o sistema, ou parte, sob uma determinada ótica. É como se o sistema fosse modelado em fatias, sendo que alguns diagramas focam o sistema de forma mais geral, enquanto outros oferecem uma visão mais técnica do sistema ou de uma característica especifica do mesmo.

Um diagrama é a apresentação gráfica de um conjunto de elementos, geralmente representados como gráficos de vértices (itens) e arcos (relacionamentos). São desenhados para permitir a visualização de um sistema sob diferentes perspectivas, ou seja, um diagrama constitui uma projeção de um determinado sistema. Em todos os sistemas, com exceção dos mais triviais, um diagrama representa uma visão parcial dos elementos que compõem o sistema. O mesmo elemento pode aparecer em todos os diagramas, apenas alguns (o caso mais comum) ou em nenhum (caso mais raro) [35].

Os diagramas da SysML representam os elementos do modelo da SysML, como pacotes, classes e associações. A classificação dos 10 diagramas da SysML, a qual reutiliza muitos dos principais tipos de diagramas da UML, é ilustrada na figura 17. Em muitos casos, os diagramas da UML são exatamente reutilizados dentro da SysML. Entretanto, em outros casos eles são modificados para atender as extensões da SysML. Existem alguns diagramas da UML que não são identificados como um único tipo de diagrama dentro da SysML. Um exemplo é o relacionamento de implantação da SysML, que representa a implantação do software para o hardware, que é integrado com o diagrama de Montagem da SysML, excluindo a utilização do diagrama de Implantação da UML. Foram criados dois tipos de diagramas, Requisitos e Paramétrico, para a versão inicial da SysML e outros diagramas são planejados para as futuras versões [32].

Os 10 diagramas da SysML podem ser divididos de acordo com sua utilização em três partes: estrutural, comportamental e genérica.

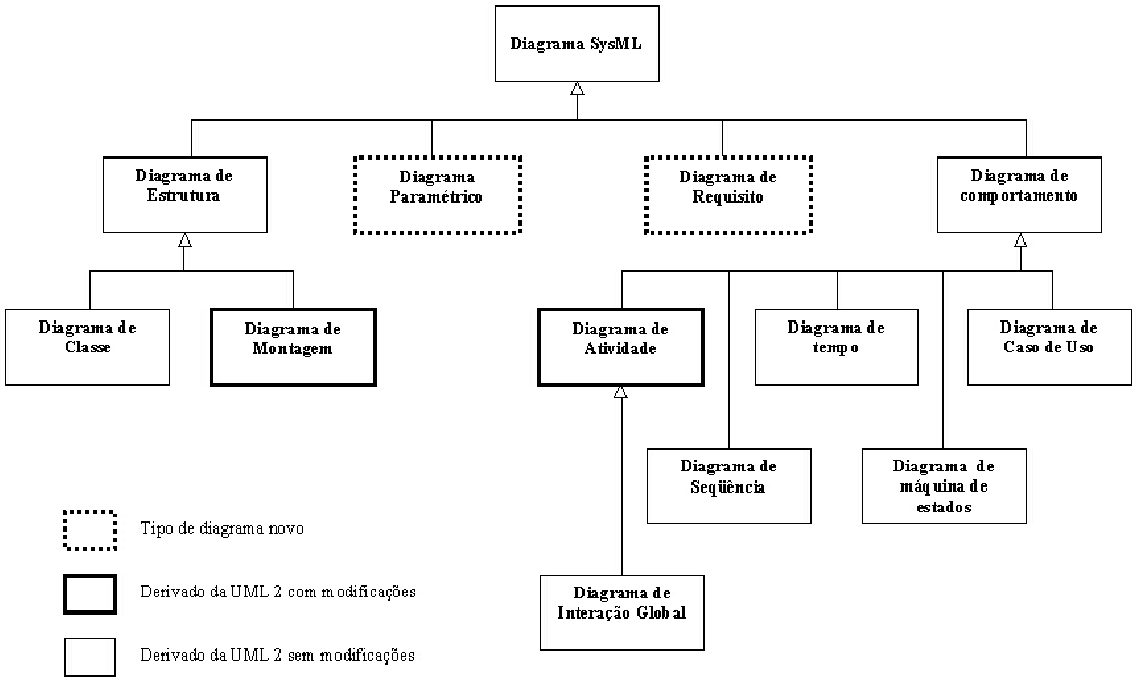


Figura 17: Estrutura de diagramas da linguagem SysML versão 0.9

5.3 Parte estrutural da SysML

A parte estrutural é definida como a parte estática do modelo, representando elementos conceituais ou físicos. A parte estrutural é formada dos seguintes diagramas: Classes, Montagem e Paramétrico.

5.3.1 Diagrama de Classes

A linguagem SysML adicionou duas extensões no Diagrama de Classes da UML 2. Um conjunto de dependência (*Dependency Set*) foi adicionado no grupo de relacionamentos de dependência e uma anotação de raiz (*Root notation*) foi adicionado para descrever múltiplos níveis de especialização.

O Diagrama de Classes define classes e relacionamentos entre as mesmas. Segundo [33], uma classe é a representação de um conjunto de coisas reais ou abstratas que são reconhecidas como sendo do mesmo tipo por compartilhar as mesmas características de atributos, operações, relações e semântica. Um relacionamento é o compartilhamento de informações e a colaboração entre as classes, permitindo a execução dos diversos

processos do sistema. Um relacionamento entre classes pode ser: associação, especialização/generalização, dependência ou realização.

O principal enfoque do Diagrama de Classes, ilustrado na figura 18, está em permitir a visualização das classes que compõem o sistema com seus respectivos atributos e métodos, demonstrando como as classes se relacionam, complementam e transmitem informações entre si. O Diagrama de Classes serve ainda como base para a construção da maioria dos outros diagramas.

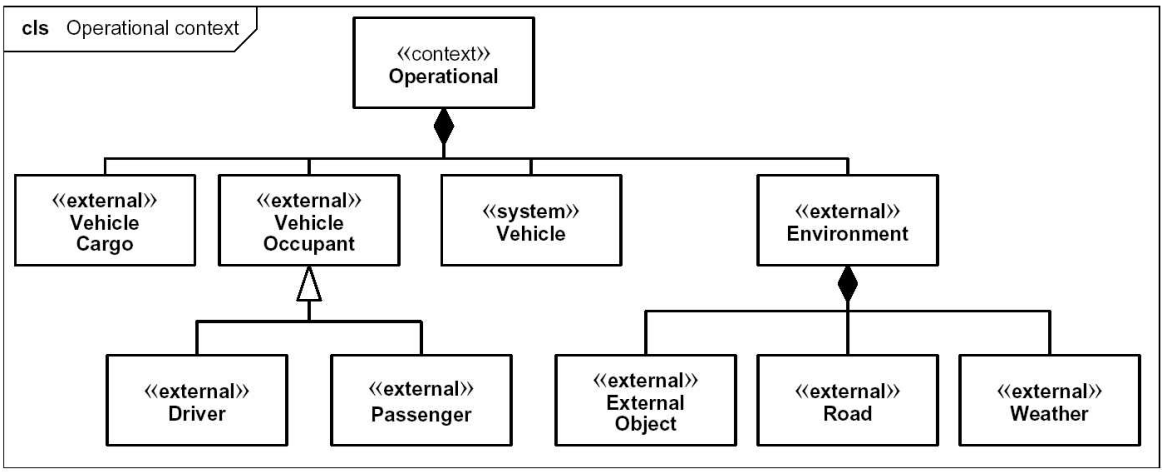


Figura 18: Exemplo de Diagrama de Classes da especificação SysML [32]

5.3.2 Diagrama de Montagem

O Diagrama de Montagem da SysML e seus diagramas associados são baseados na especificação da UML para o diagrama de Estruturas Compostas. A Estrutura Composta (*Composite Structures*) da UML já oferece mecanismos essenciais para definição de um componente, designando características estruturais e integrando-o a uma classe.

O Diagrama de Montagem pode representar qualquer nível da hierarquia do sistema, incluindo o nível superior do sistema, um subsistema, um componente lógico ou físico do sistema ou o ambiente [34].

As portas são um tipo especial de componente que dão acesso à estrutura interna do sistema partindo do meio externo ao objeto. Na SysML, diferentemente da UML, as portas podem ser estruturas com hierarquia complexa. A SysML cria o estereótipo chamado *NestedConenctedEnd* que é uma extensão do *ConnectedEnd* da UML, permitindo que

qualquer terminação de conector, mesmo encapsulada em mais de um nível, seja referenciada através da propriedade chamada *path reference* (caminho de referência).

O Diagrama de Montagem descreve um sistema como uma coleção de partes com função específica dentro de um conjunto, como ilustra a figura 19. O diagrama também mostra as conexões entre as partes que permitem a interoperabilidade com outras partes do sistema. Algumas partes podem ser mostradas no Diagrama de Montagem como portas, indicando que esta parte pode ser conectada externamente ao contexto no qual elas são usadas. Cada parte pode ser definida como uma classe com suas próprias partes, portas e estrutura interna, podendo ser definido um conjunto de elementos uniformes aplicados a múltiplos níveis da hierarquia do sistema [32].

O modelo de montagem da SysML possui a capacidade de generalização do modelo do sistema como uma árvore de componentes modular. Conforme as particularidades do sistema, pode-se selecionar tipos específicos de componentes, os tipos de conexão entre eles e a combinação destes elementos, definindo o sistema total.

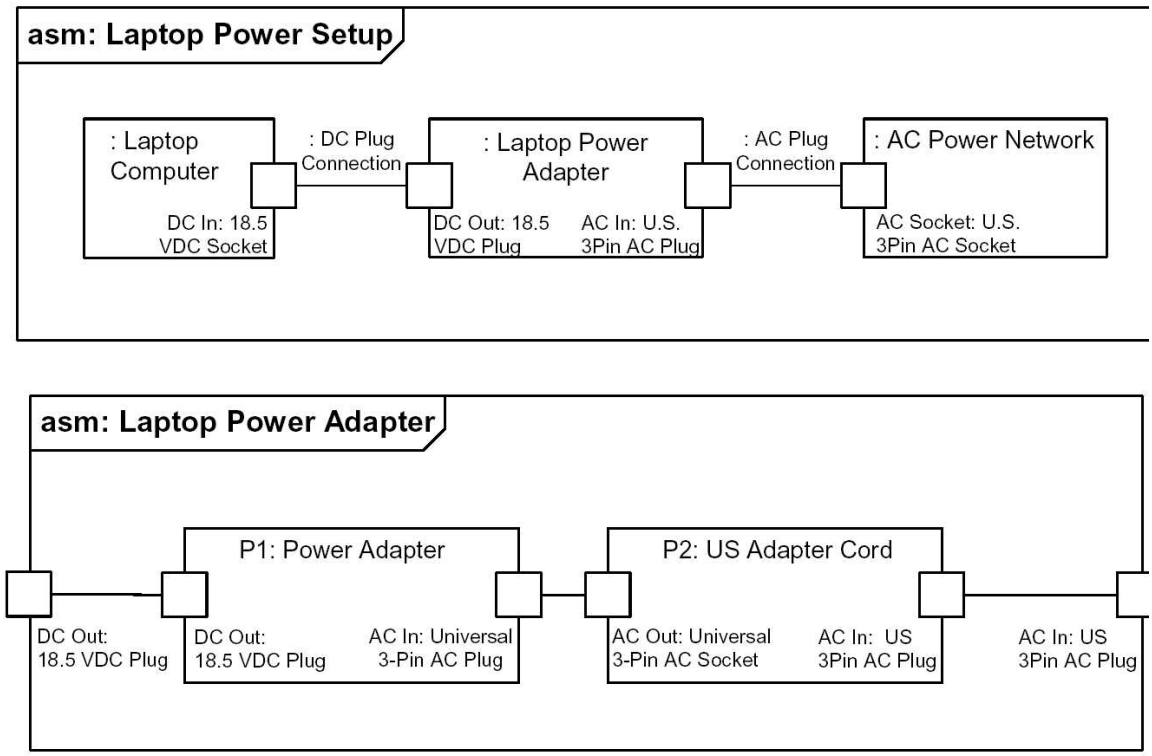


Figura 19: Exemplo de Diagrama de Montagem da especificação SysML [32]

O modelo de montagem da SysML pode ser utilizado durante todas as etapas de especificação e projeto do sistema, incluindo a modelagem lógica ou física do sistema,

especificação do *software*, *hardware* e as partes que atuam em diferentes meios, como operações de *software*, fluxo de entrada e saídas de dados ou interações contínuas.

5.3.3 Diagrama Paramétrico

O Diagrama Paramétrico, desenvolvido para a SysML, modela as propriedades e seus relacionamentos, os quais representam uma expressão ou restrição, matemática ou lógica. Geralmente existe uma referência para a linguagem utilizada para representar as expressões e restrições. O modelo paramétrico pode conter equações diferenciais, expressões lógicas como {enquanto Y=7 ou X<1}, ou outra restrição como {Y<3+7}, retirada de uma linguagem de programação específica [34].

As restrições paramétricas oferecem mecanismos para a integração dos modelos da engenharia para desempenho e segurança com o modelo de montagem da SysML. Restrições paramétricas representam uma rede de restrições entre as propriedades do sistema, que podem ser utilizadas para representar expressões matemáticas que descrevem as propriedades físicas de um sistema [32].

Um modelo paramétrico descreve as restrições entre as propriedades de um sistema, onde uma restrição paramétrica pode ser utilizada para expressar os relacionamentos entre as propriedades que são identificadas no modelo estrutural do sistema. Os modelos paramétricos são geralmente capturados da análise de modelos que descrevem o controle e o desempenho do sistema, e modelos de engenharia para reutilização, segurança, custo de projeto, etc [34].

A UML oferece uma linguagem de restrição, mas a mesma não oferece suporte a reutilização de equações. A SysML resolveu tal deficiência introduzindo um modelo de restrição chamado relações paramétricas.

Segundo [35], o modelo paramétrico da SyML possui duas partes:

- Relações paramétricas definidas como uma combinação que pode ser utilizada novamente em outras relações;

- Aplicação de uma relação paramétrica em uma propriedade particular do objeto.

O tempo pode ser modelado como uma propriedade adicional, onde outras propriedades podem ser dependentes. A referência de tempo pode ser estabelecida por um relógio local ou global que produz um valor de tempo contínuo ou discreto que é definido em qualquer nível do sistema.

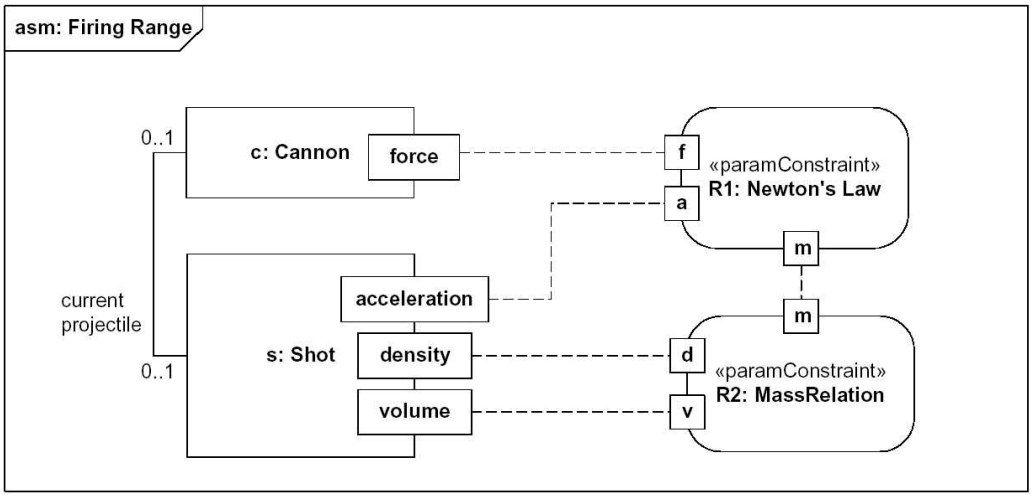


Figura 20: Exemplo das restrições paramétricas em conjunto com o Diagrama de Montagem [32]

Restrições paramétricas são geralmente utilizadas em combinação com o diagrama de montagem da SysML, como ilustra a figura 20. As restrições paramétricas são definidas como um estereótipo <<*paramConstraint*>> aplicado à definição da classe. As propriedades desta classe definem os parâmetros das restrições. Entretanto, as restrições paramétricas podem ser descritas utilizando um diagrama específico, como ilustra a figura 21.

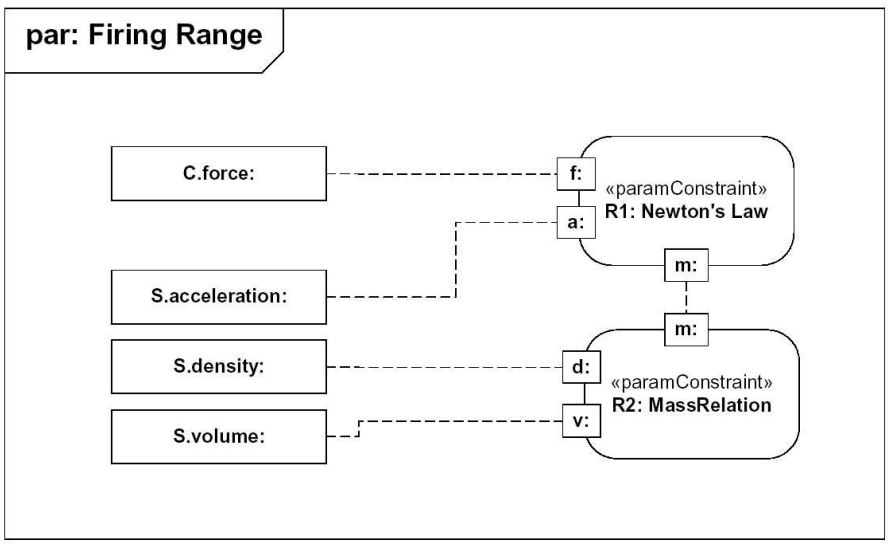


Figura 21: Exemplo do Diagrama Paramétrico da especificação SysML [32]

5.4 Parte comportamental da SysML

A parte comportamental especifica a dinâmica do sistema, representando comportamentos no tempo e espaço. A parte comportamental é formada dos seguintes diagramas: Atividades, Interações (Seqüência e Tempo), Máquina de Estados e Casos de Uso.

5.4.1 Diagrama de Atividades

O Diagrama de Atividades é o diagrama com maior ênfase ao nível de algoritmo e provavelmente um dos mais detalhistas. Este diagrama apresenta muitas semelhanças com os antigos fluxogramas utilizados para desenvolver a lógica de programação [36]. A figura 22 ilustra o Diagrama de Atividades

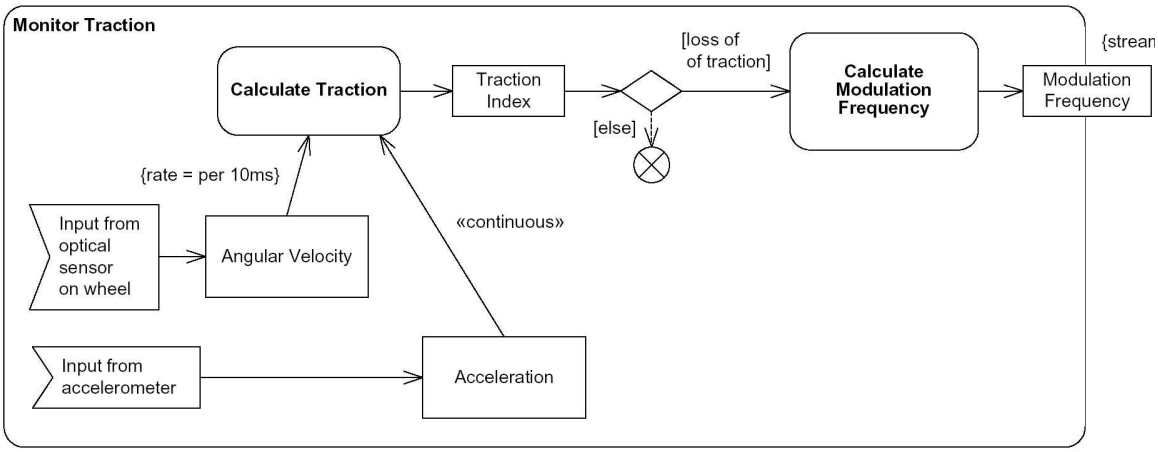


Figura 22: Exemplo do Diagrama de Atividades da especificação SysML [32]

A modelagem da atividade dá ênfase às entradas e saídas, seqüência e condições de coordenação de outros comportamentos. As modificações no Diagrama de Atividade da UML 2.0 incluem os seguintes aspectos: controle com dados, sistemas contínuos, probabilidade e o suporte para a tradução do EFFBD (*Enhanced Functional Flow Block Diagrams* – Diagrama de fluxo de blocos funcionais melhorado) para o Diagrama de Atividades.

Controle com dados: a linguagem SysML aumentou o controle dentro do Diagrama de Atividade em três caminhos inter-relacionados, melhorando a descrição dos dados. No diagrama de Atividade da UML 2.0, o controle somente pode habilitar a partida de ações. Já na SysML, o controle pode desabilitar as ações que já estão executando através do

estereótipo <<*controlOperator*>> e pode declarar ações ininterruptas, onde as ações não podem ser desabilitadas após sua partida.

A SysML permite que os sinais de controle sejam inseridos em uma fila e disponibiliza um pino de controle (tipo de nó de objeto) para receber os mesmos. Ela também cria um tipo, <<*enumeration*>>, para controle de valores, os quais podem ser entradas para o fluxo de uma atividade ou saídas desta mesma atividade.

Sistemas contínuos: SysML oferece extensões que podem ser agrupadas sob o termo contínuos, mas geralmente são aplicados para vários tipos de fluxo de informação e itens físicos através do sistema.

SysML cria o estereótipo <<*rate*>> para modelar as restrições na velocidade do fluxo de entidades na fronteira de uma atividade ou parâmetro de um comportamento. Isto inclui fluxo discreto ou contínuo de material, energia ou informação.

Probabilidade: foi inserido na biblioteca de modelos da SysML o estereótipo <<*probability*>> que descreve um comportamento para o cálculo de probabilidade. O cálculo da probabilidade cria a possibilidade para que um valor levado ao nó de decisão ou ao nó de um objeto seja um valor de saída do mesmo conforme a probabilidade atribuída.

EFFBD: o Diagrama de Fluxo de Bloco Funcional (FFBD) é um tipo de diagrama de controle de fluxo muito utilizado por engenheiros de sistema. Porém, o FFBD não contém a semântica para a modelagem do fluxo de dados entre as funções, consequentemente não representando qualquer disparo de dados das funções. O EFFBD representa funções, fluxo de controle e fluxo de dados, permitindo a modelagem da dependência dos dados. Eles são muito parecidos com os Diagramas de Atividades e a SysML acrescentou extensões que os tornam mais compatíveis, possibilitando a tradução.

5.4.2 Diagrama de Interação

O Diagrama de Interação da SysML inclui o Diagrama de Seqüência e o Diagrama de Tempo, os quais vem da UML 2 sem modificações ou extensões. Entretanto, o diagrama de Interação da SysML não permite a comunicação entre diagramas e o Diagrama de Interação Geral [32].

Diagrama de Seqüência: este diagrama procura descrever a seqüência de eventos que ocorrem em um determinado processo, identificando quais métodos devem ser disparados entre os atores e os objetos envolvidos e em que ordem. O Diagrama de Seqüência baseia-se no Diagrama de Casos de uso, havendo normalmente um Diagrama de Seqüência para cada Caso de Uso, que em geral, refere-se a um processo disparado por um ator [36].

A seqüência de eventos, ou o fluxo de controle, é definido através do envio e o recebimento de mensagens entre as linhas de vida, que representam o tempo de existência de um objeto ou ator durante um processo. Uma mensagem combina controle e fluxo de dados. A figura 23 traz um exemplo do Diagrama de Seqüência.

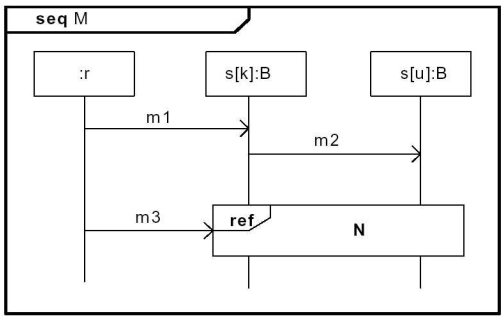


Figura 23: Exemplo de Diagrama de Seqüência da especificação SysML [32]

Diagrama de Tempo: representa uma mudança sobre um tempo associado com mudanças em estados, atividades ou valores de propriedades. O Diagrama de Tempo é um gráfico de duas dimensões que representa o tempo em um eixo e um estado, atividade ou propriedade, em outro eixo, como ilustra a figura 24.

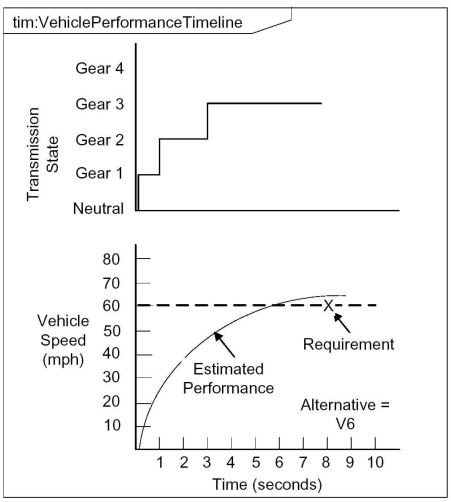


Figura 24: Exemplo do Diagrama de Tempo da especificação SysML [32]

5.4.3 Diagrama de Máquina de Estados

A UML 2 possui dois tipos de Diagrama de Máquina de Estados, o comportamental e o protocolo. O Diagrama de Máquina de Estados da SysML é importado da UML 2 sem modificações ou extensões. Entretanto, a SysML não inclui o tipo protocolo [32].

O Diagrama de Máquina de Estados demonstra o comportamento de um elemento através de um conjunto de transições de estado. O elemento modelado muitas vezes é uma instância de uma classe. No entanto, pode-se usar este diagrama para modelar o comportamento de um Caso de Uso ou mesmo o comportamento de um sistema completo [36]. A figura 25 ilustra um exemplo do Diagrama de Máquina de Estados.

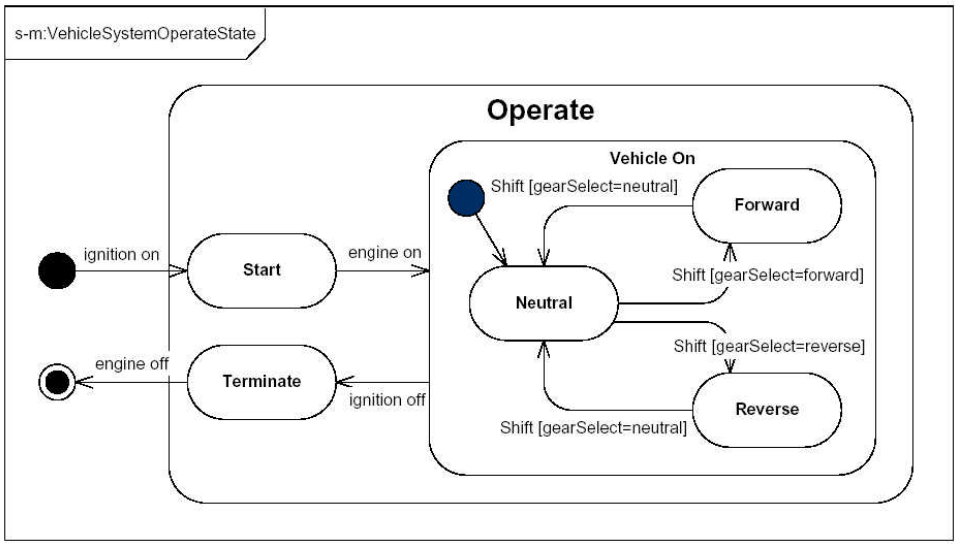


Figura 25: Exemplo do Diagrama de Máquina de Estados da especificação SysML [32]

Um estado representa a situação na qual um objeto encontra-se em um determinado momento. Um objeto pode passar por diversos estados dentro de um mesmo processo. Um estado pode demonstrar a espera pela ocorrência de um evento, a reação a um estímulo, a execução de alguma atividade ou a satisfação de alguma condição. Uma transição representa um evento que causa uma mudança no estado de um objeto, gerando um novo estado [36].

5.4.4 Diagrama de Casos de Uso

O Diagrama de Casos de Uso da SysML é importado da UML 2 sem modificações ou extensões [32].

O Diagrama de Casos de Uso possibilita a compreensão do comportamento externo do sistema. Procura apresentar o sistema através de uma perspectiva do usuário, demonstrando as funções e serviços oferecidos e quais usuários poderão utilizar cada serviço, como ilustra a figura 26. É o diagrama mais abstrato, flexível e informal, sendo utilizado principalmente no início da montagem do sistema, embora venha a ser consultado e modificado durante todo o processo de engenharia. O Diagrama de Casos de Uso serve como base para a modelagem de outros diagramas [36].

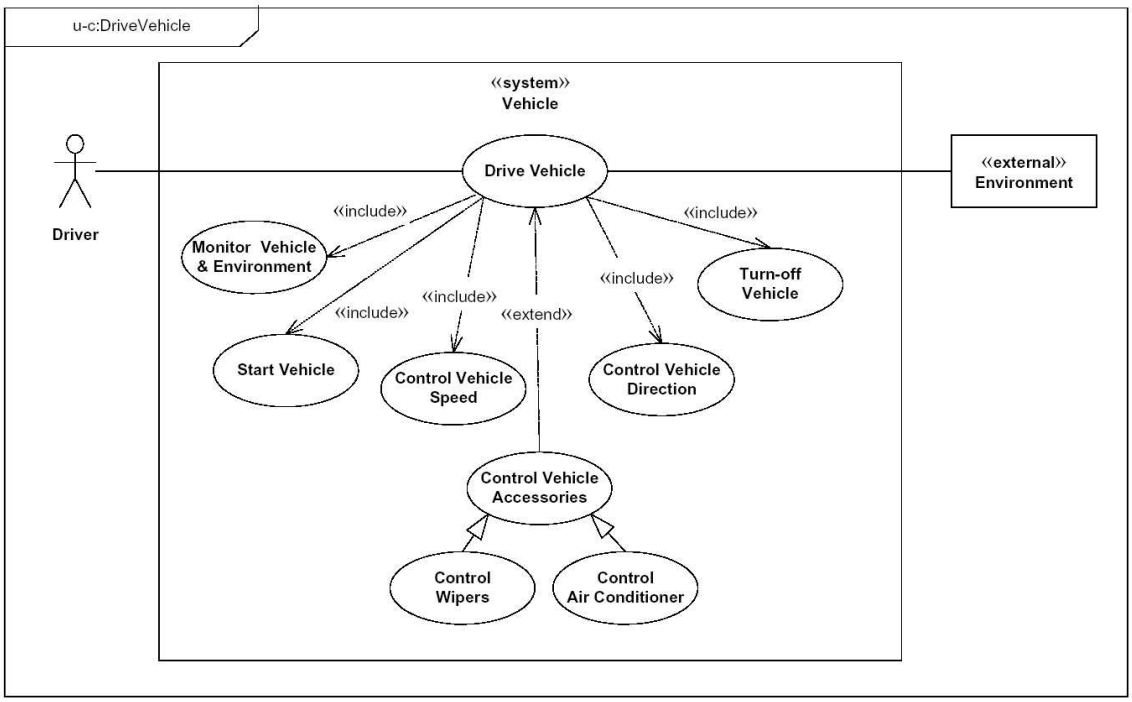


Figura 26: Exemplo do Diagrama de Casos de Uso da especificação SysML [32]

Os atores representam elementos externos, tais como usuários, hardware e software, que interagem com os serviços e funções do sistema. Os atores são representados por símbolos de bonecos, com uma pequena descrição que identifica o papel do ator dentro do diagrama.

Os casos de uso referem-se aos serviços, tarefas e funções oferecidas pelo sistema e que são executadas através de interações entre o sistema e os atores.

O Diagrama de Caso de Uso possui os seguintes relacionamentos: inclusão, extensão, generalização. O relacionamento de inclusão indica obrigatoriedade, pois quando um determinado caso de uso possui um relacionamento de inclusão com outro, a execução do primeiro obriga a execução do segundo. Um relacionamento de inclusão pode ser

comparado à chamada de uma sub-rotina. O relacionamento de extensão é utilizado para descrever cenários opcionais de um caso de uso, que ocorrerão se uma determinada condição for satisfeita. O relacionamento entre casos de uso com características semelhantes, apresentando pequenas diferenças entre si, é chamado de generalização.

5.5 Parte Genérica da SysML

A parte genérica especifica construções que podem ser aplicadas nas partes estrutural e comportamental. A parte genérica é formada pelo Diagrama de Requisitos, relacionamentos de alocação e construções auxiliares para o modelo.

5.5.1 Diagrama de Requisitos

O Diagrama de Requisitos é um diagrama desenvolvido para a SysML, sendo utilizado para a descrição textual dos requisitos, relacionado-os com a especificação, análise e projeto do sistema, como ilustra a figura 27. Um requisito representa um comportamento, estrutura ou propriedade que um sistema, componente ou outro elemento do modelo possa satisfazer [34].

A modelagem de requisitos inclui a tradução das necessidades expressas da forma textual em linguagem natural para uma sintaxe formal dentro de um modelo, capturando a evolução dos requisitos e seus relacionamentos com o projeto do sistema. Segundo [35], a modelagem de requisitos compreende no mínimo três aspectos importantes:

- Tradução do texto para o modelo: inicialmente os requisitos surgem como um grande documento de texto. O modelo de requisitos é uma análise deste documento. O modelo possui um elemento para cada requisito, encapsulando uma porção do texto que corresponde ao requisito, adicionando uma referência à localização do mesmo no documento de origem.

- Derivação: os requisitos podem ser mais ou menos específicos sobre uma necessidade do sistema. Um norma que especifica um equipamento pode trazer várias derivações para itens específicos de desempenho, funcionamento e segurança. Para SysML, essa derivação para um requisito mais específico chama-

se <<trace>>. Cada estágio de derivação inclui considerações adicionais sobre o projeto de sistema.

- Conexão com o projeto do sistema: possibilidade de conexão com os outros diagramas da modelagem do sistema.

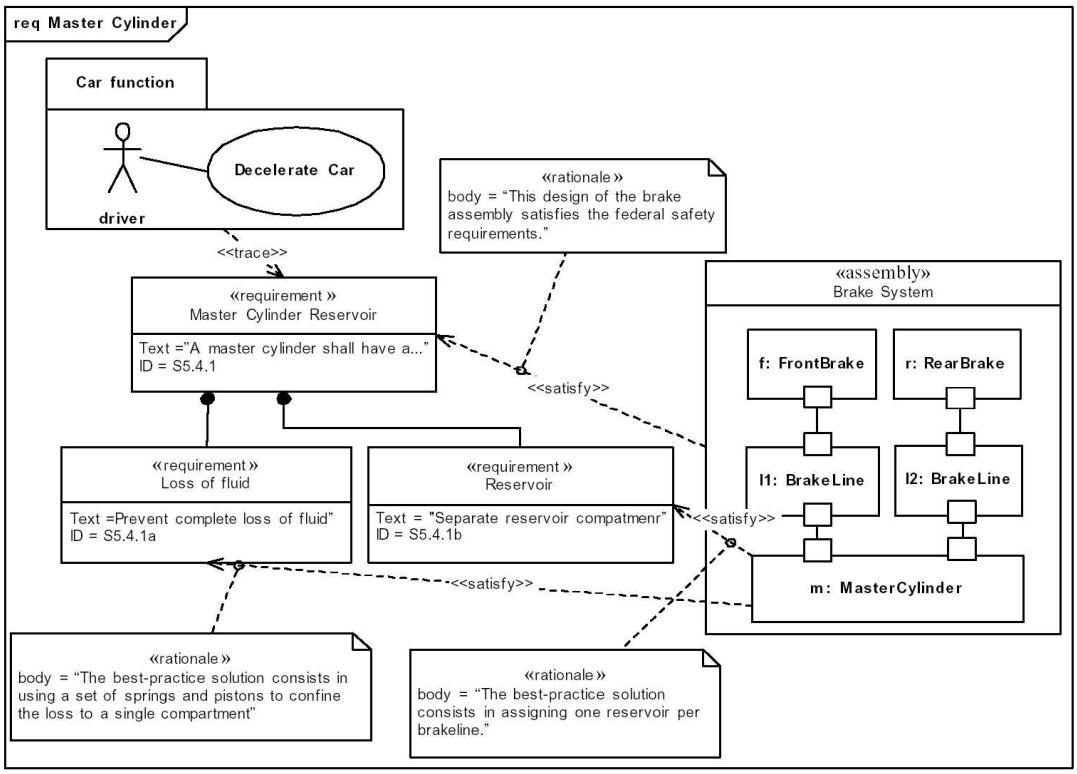


Figura 27: Exemplo do Diagrama de Requisitos da especificação SysML[32]

SysML oferece a modelagem de construções que representam os requisitos e as relações com outros elementos. Um requisito pode apresentar os seguintes relacionamentos:

- <<derive>>: permite a criação ou conclusão de um requisito a partir de outro;
- <<satisfy>>: permite o atendimento de um requisito por um outro elemento;
- <<verify>>: permite a verificação de um requisito utilizando um procedimento de teste;
- <<rationale>>: permite a explicação para o principio ou razão de uma decisão de modelagem, sendo possível incluir uma análise de mercado.

Todos estes relacionamentos são especializações do relacionamento <<trace>> da UML, que é utilizado para sinalizar os requisitos e mudanças entre os modelos.

Projetistas podem customizar a classificação dos requisitos com a adição da definição de subtipos do estereótipo <<requirement>> da SysML, como por exemplo, OperationalRequirement, FunctionRequirement, etc.

Um requisito pode definir suas próprias propriedades, oferecendo um valor computável acompanhado da definição textual do requisito.

5.5.2 Relacionamentos de Alocação

Alocação é um mecanismo para associação de diferentes tipos de elementos, ou diferentes hierarquias em um mesmo nível de abstração.

Alocação é o termo utilizado pelos engenheiros de sistemas para descrever a organização dos elementos de associação cruzada com as várias estruturas ou hierarquias de um modelo do usuário.

5.5.3 Construções Auxiliares

Construções auxiliares são elementos e notações para itens de fluxo, modelo de dado de referência, ponto de vista e observações, tipos de dados adicionais, dimensionamento de quantidades, distribuição probabilística e propriedade para restrição de valor.

A SysML refinou a concepção dos pontos de vista e observações da UML e adicionou notações para mostrar as observações com uma lista das partes interessadas e seus interesses. SysML também adicionou à definição da UML tipos de numeração real e complexo. Vetores e estruturas compostas podem ser modelados utilizando a norma de elementos múltiplos da UML 2.

5.6 Conclusão

No desenvolvimento deste capítulo, a SysML encontrava-se disponível na versão rascunho 0.9. Entretanto, no período de dezembro de 2005 a janeiro 2006, dois grupos de trabalho apresentaram duas propostas de atualização da versão 0.9, a versão 0.98 e a versão

1.0, que seriam votadas em fevereiro de 2006. Porém, a OMG e a INCOSE, solicitaram que os dois grupos de trabalho unificassem as duas versões, apresentando apenas uma proposta em fevereiro de 2006. A versão 0.99 foi o resultado desta solicitação e deverá ser aprovada em abril de 2006.

A dissertação está utilizando a especificação rascunho 0.9 da SysML na modelagem dos sistemas embutidos descritos no capítulo 6. A versão 0.9 da SysML apresenta uma linguagem de modelagem semelhante à UML 2, sendo em geral uma simplificação da mesma. Tal semelhança obriga o projetista a possuir conhecimento básico de UML e orientação a objetos para iniciar a modelagem do sistema embutido.

Como a equipe de desenvolvimento de um sistema embutido é interdiciplinar, envolvendo profissionais de áreas distintas, a SysML é uma proposta muito interessante para melhorar a comunicação entre as áreas afim. Um exemplo para ilustrar tal afirmação, seria uma reunião sobre determinado sistema embutido, onde um engenheiro eletrônico traz os diagramas eletrônicos, um engenheiro de controle traz os diagramas de controle e um programador traz os diagramas de software. A interação entre a equipe é mínima, pois um não tem conhecimento da linguagem de modelagem usada no diagrama do outro. Entretanto, se a SysML fosse utilizada, a linguagem de modelagem seria a mesma para todas as diferentes áreas de conhecimento, proporcionando a máxima interação entre a equipe, pois todos estaria falando o mesmo “idioma”.

Com base na metodologia de hardware/software co-design, a linguagem SysML é utilizada na etapa de especificação do sistema, possibilitando a modelagem do sistema embutido incluindo as partes de hardware e software. A partir dos modelos SysML, as equipes interdisciplinares podem gerar modelos mais formais, descrevendo com mais detalhes a solução final para as diferentes partes do sistema embutido.

Capítulo 6

Estudos de Caso Sobre o Uso de SysML

Neste capítulo serão utilizados dois sistemas embutidos como estudos de caso. O primeiro é um sistema de injeção eletrônica para motor a combustão e o segundo um relé de proteção para motor elétrico.

Cada estudo de caso será composto de uma breve descrição, uma especificação informal, a qual traz informações do sistema em linguagem natural, e uma especificação baseada em diagramas, utilizando a linguagem SysML para modelagem do sistema. O objetivo é modelar as partes de *hardware* e *software* que compõe o sistema embutido, procurando enfatizar todos os requisitos e aspectos que são pertinentes à seleção do processador embutido. Toda a modelagem SysML apresentada neste capítulo foi realizada pelo mestrando.

6.1 Sistema de Injeção Eletrônica de Combustível para Motor a Combustão

O sistema de injeção eletrônica de combustível para motor a combustão está sendo desenvolvido pelo Laboratório de Controle de Processos e Instrumentação (LCPI) do Departamento de Automação e Sistemas (DAS) da Universidade Federal de Santa Catarina (UFSC). Este sistema de injeção eletrônica será utilizado em um motor a combustão de um cilindro com capacidade de 200 cilindradas, potência de 7cv e rotação máxima de 12000 rpm.

As partes de um motor a combustão variam dependendo do tipo do motor. Para um motor de quatro tempos, as partes principais incluem o eixo manivela (virabrequim), eixos excêntrico (comando de válvulas) e válvulas (admissão e escape), como ilustra a figura 28.

O motor a combustão realiza trabalho queimando uma mistura de vapor de combustível e ar dentro de um cilindro. Por esta razão, é também chamado de motor de combustão interna. Quando a mistura de ar com combustível queima, formam-se gases quentes. Estes expandem-se rapidamente e empurram o pistão do motor, movimentando o virabrequim. Este movimento pode acionar rodas, hélices e operar máquinas. A potência de um motor a combustão, isto é, o trabalho que ele pode produzir, é geralmente expressa em cavalos-vapor ou *watts*.

Os motores a combustão são compactos e leves comparados a sua potência. Isto os torna mais utilizados em veículos como automóveis, cortadores de grama, motocicletas, ônibus, aviões e pequenos barcos.

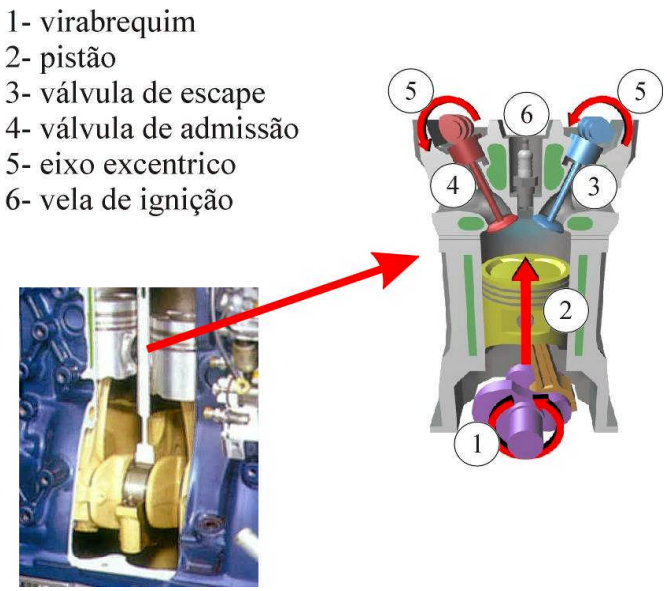
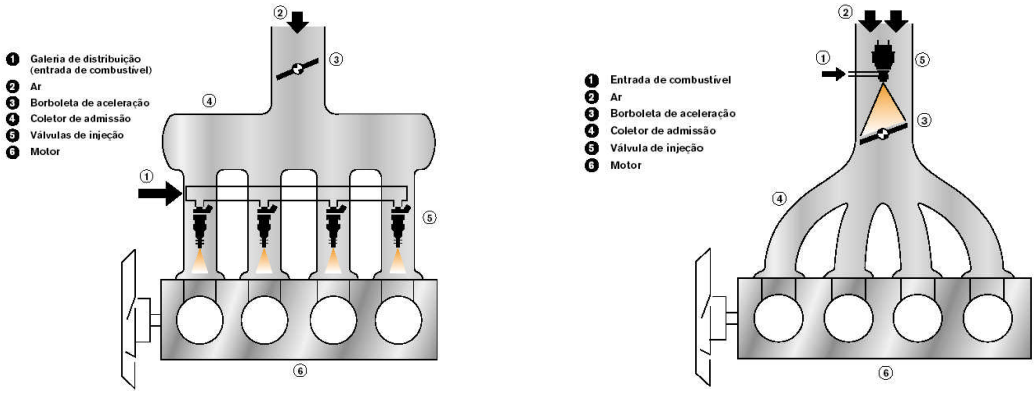


Figura 28: Motor a combustão

Devido á rápida evolução dos motores dos automóveis, onde o carburador não supria as necessidades dos novos veículos, no que se refere à poluição, economia de combustível, potência, respostas rápidas as acelerações e etc, desenvolveu-se os sistemas de injeção eletrônica de combustível, que tem por objetivo proporcionar ao motor melhor rendimento com mais economia, em todos os regimes de funcionamento. Para garantir tal objetivo, o sistema de injeção eletrônica de combustível fornece ao motor a perfeita

mistura ar/combustível em todas as faixas de rotação. Quanto mais adequada é esta mistura, melhor o rendimento e economia, e menor a emissão de gases poluentes [37].

Segundo [37], os sistemas de injeção eletrônica podem ser de dois tipos: multiponto e monoponto, como ilustra a figura 29.



Multiponto Monoponto

Figura 29: Tipos de sistemas de injeção eletrônica

6.1.1 Descrição informal

Inicialmente foi realizado a análise de requisitos, em reunião com o engenheiro de controle responsável pelo desenvolvimento do sistema de injeção eletrônica, definindo as características, restrições e funcionamento deste sistema.

O sistema de injeção eletrônica é constituído basicamente de sensores e atuadores, como ilustra a figura 30.

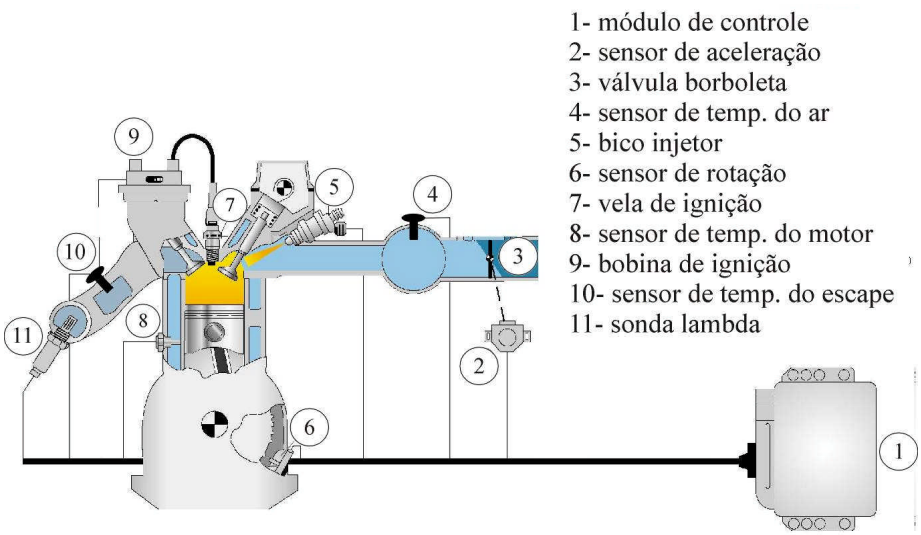


Figura 30: Sistema de injeção eletrônica

Os sensores são componentes que estão instalados em vários pontos do motor e servem para enviar informações para a unidade de controle. São utilizados os seguintes sensores:

- sensor de temperatura do motor;
- sensor de temperatura do ar de entrada;
- sensor de temperatura do escape;
- sensor indutivo de rotação;
- sensor de aceleração;
- sonda λ (responsável pela medição da concentração de combustível na mistura com o ar).

Os atuadores são componentes que recebem informações da unidade de controle e atuam no sistema de alimentação, variando o volume de combustível que o motor recebe. São utilizados os seguintes atuadores:

- bico injetor;
- vela de ignição;
- válvula de entrada de ar (borboleta).

A unidade de controle é responsável pela medição dos sensores e cálculo dos tempos de atuação para cada atuador, respeitando restrições de tempo para garantia da execução do controle. O diagrama do sistema de injeção é ilustrado na figura 31.

As restrições de tempo do sistema são impostas pelas características do motor a combustão a ser controlado. É definido que uma volta do motor (360°) é completada a cada 5us para 12000rpm. Outra definição é o período do pulso que ativa o bico injetor, o qual deve ter duração de 25us para 200 cilindradas. O atuador da válvula borboleta considera posição 0° como pulso de 1ms e 90° como pulso de 2ms, dentro de um período de 25ms. Considerando estas restrições temporais, a leitura dos sensores e o cálculo da composição

da mistura do ar/combustível, definindo os tempos de acionamento para os atuadores, devem ser computados em no máximo 15ms.

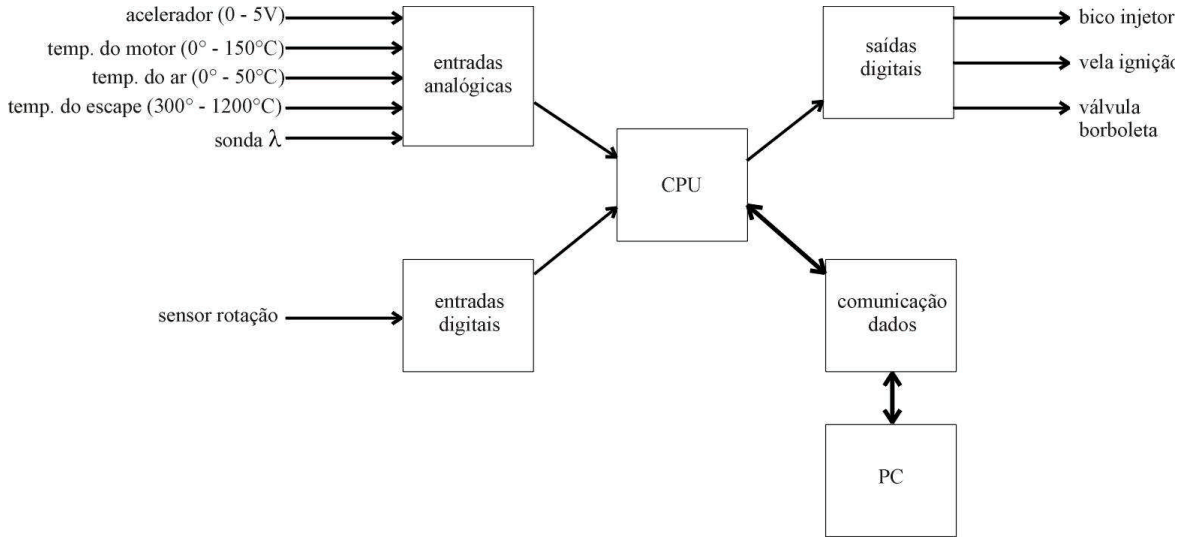


Figura 31: Diagrama do sistema de injeção eletrônica

6.1.2 Descrição em Diagramas

A próxima etapa é a especificação do sistema, transformando a descrição informal do mesmo em especificação baseada em diagramas. O sistema de injeção eletrônica foi descrito em diagramas utilizando a SysML.

Na modelagem do sistema embutido para injeção eletrônica, foram utilizados os seguintes diagramas da SysML: Diagrama de Casos de Uso, Diagrama de Classes, Diagrama de Requisitos, Diagrama de Montagem, Diagrama Paramétrico, Diagrama de Atividades e Diagrama de Tempo.

6.1.2.1 Diagrama de Casos de Uso

O Diagrama de Casos de Uso, ilustrado na figura 32, é utilizado para descrever o comportamento externo do sistema, apresentando as funcionalidades e serviços do sistema de injeção ao ponto de vista do usuário.

O sistema de injeção é formado por cinco serviços: Ligar motor, Desligar motor, Acelerar motor, Controlar injeção e Comunicar dados. Os serviços Ligar motor, Desligar motor e Acelerar motor estão incluídos no serviço Controlar injeção. O serviço Controlar injeção é responsável pela medição dos sensores e cálculo dos tempos de atuação para cada

atuador. Existe um serviço opcional ao Controlar injeção, o de Comunicar dados, que permite o monitoramento on-line do funcionamento do motor. O serviço de Acelerar motor fornece ao serviço Controlar injeção, informações sobre a necessidade da composição da mistura de ar/combustível.

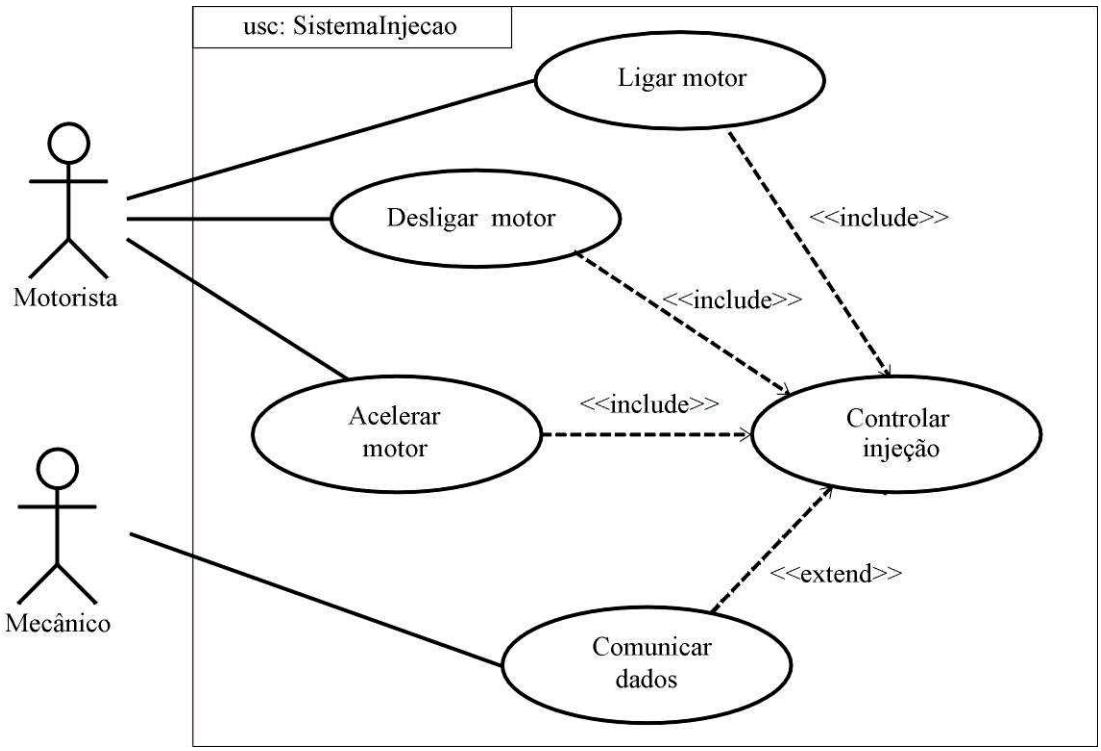


Figura 32: Diagrama de Casos de Uso do sistema de injeção

O sistema de injeção possui dois usuários, o motorista e o mecânico. O motorista interage com o sistema através dos serviços Ligar motor, Desligar motor e Acelerar motor. O mecânico, utiliza apenas o serviço opcional de Comunicar dados, para monitorar e ajustar o sistema de injeção.

6.1.2.2 Diagrama de Classes

A figura 33, ilustra o Diagrama de Classes do sistema de injeção. Esse diagrama apresenta uma visão estática de como as classes estão organizadas. Uma classe é um conjunto de objetos que compartilham os mesmos atributos, métodos e relacionamentos. Através deste diagrama é possível observar todas as partes que compõem o sistema e o relacionamento entre as mesmas.

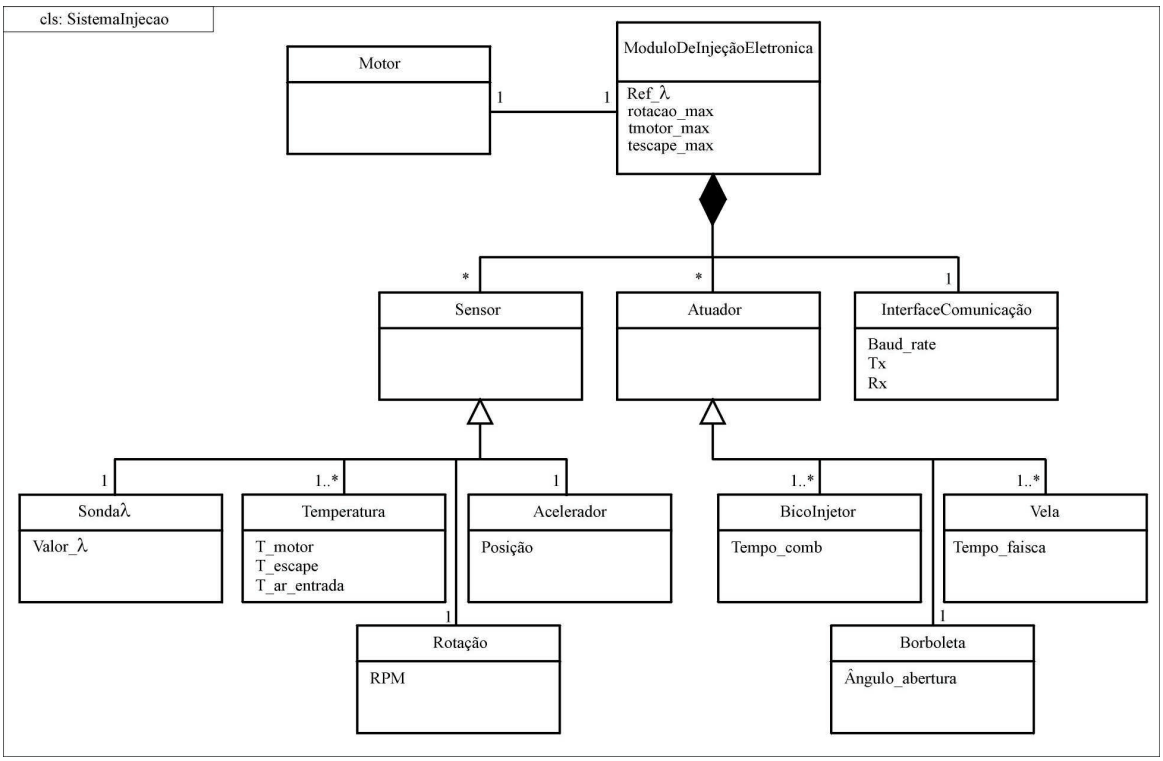


Figura 33: Diagrama de Classes do sistema de injeção

6.1.2.3 Diagrama de Requisitos

O Diagrama de Requisitos, como ilustra a figura 34, modela todas as restrições que foram verificadas na descrição em linguagem natural. Através dele pode-se modelar todos os requisitos de forma textual, referenciando normas técnicas, aspectos temporais e outros diagramas do sistema.

O módulo de injeção eletrônica deve satisfazer os requisitos da norma de poluição, executando todo o processamento em 15ms. O módulo de injeção contém outros três requisitos específicos: sensor de rotação, bico injetor e acelerador eletrônico. Cada um destes três requisitos é modelado com detalhes. Para descrever melhor o requisito do acelerador eletrônico e do bico injetor é utilizado o Diagrama de Tempo para o acionamento da válvula "borboleta" e para o acionamento do bico injetor respectivamente.

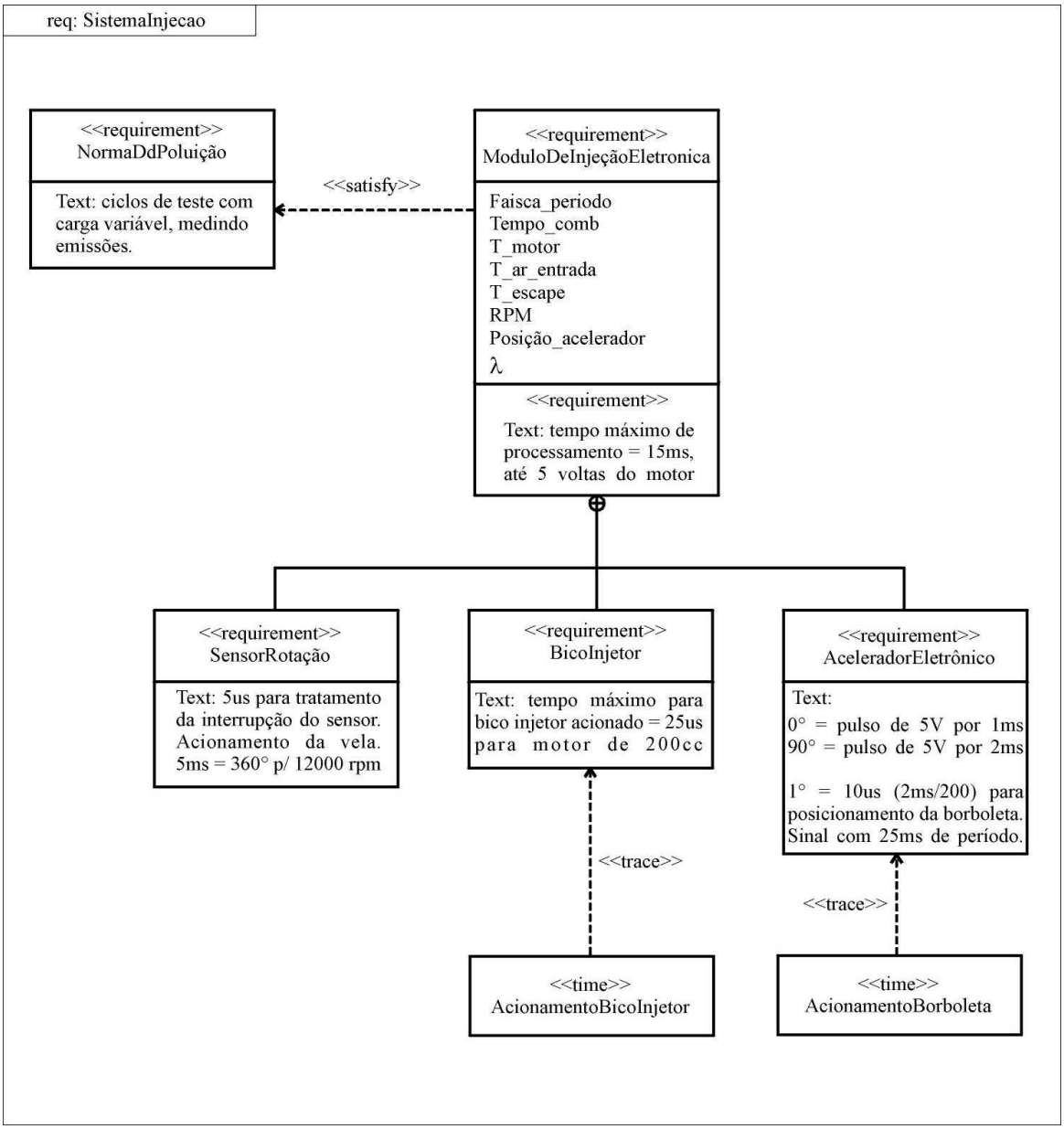


Figura 34: Diagrama de Requisitos do sistema de injeção

6.1.2.4 Diagrama de Montagem

O Diagrama de Montagem é utilizado para modelagem exclusiva das partes de *hardware* do sistema de injeção eletrônica. A modelagem das partes de *hardware* inicia com um nível de abstração alto, modelando o sistema de injeção como um todo. A medida que o modelo vai se tornando mais específico, o nível de abstração vai diminuindo, melhorando a especificação da parte ou componente de *hardware*.

Através do Diagrama de Montagem é possível modelar um conjunto de partes com função especifica dentro de uma parte maior do sistema, especificando as conexões e

interfaces entre as mesmas. A figura 35 ilustra o Diagrama de Montagem do sistema de injeção eletrônica como um todo, modelando a interface do módulo de injeção com o motor a combustão.

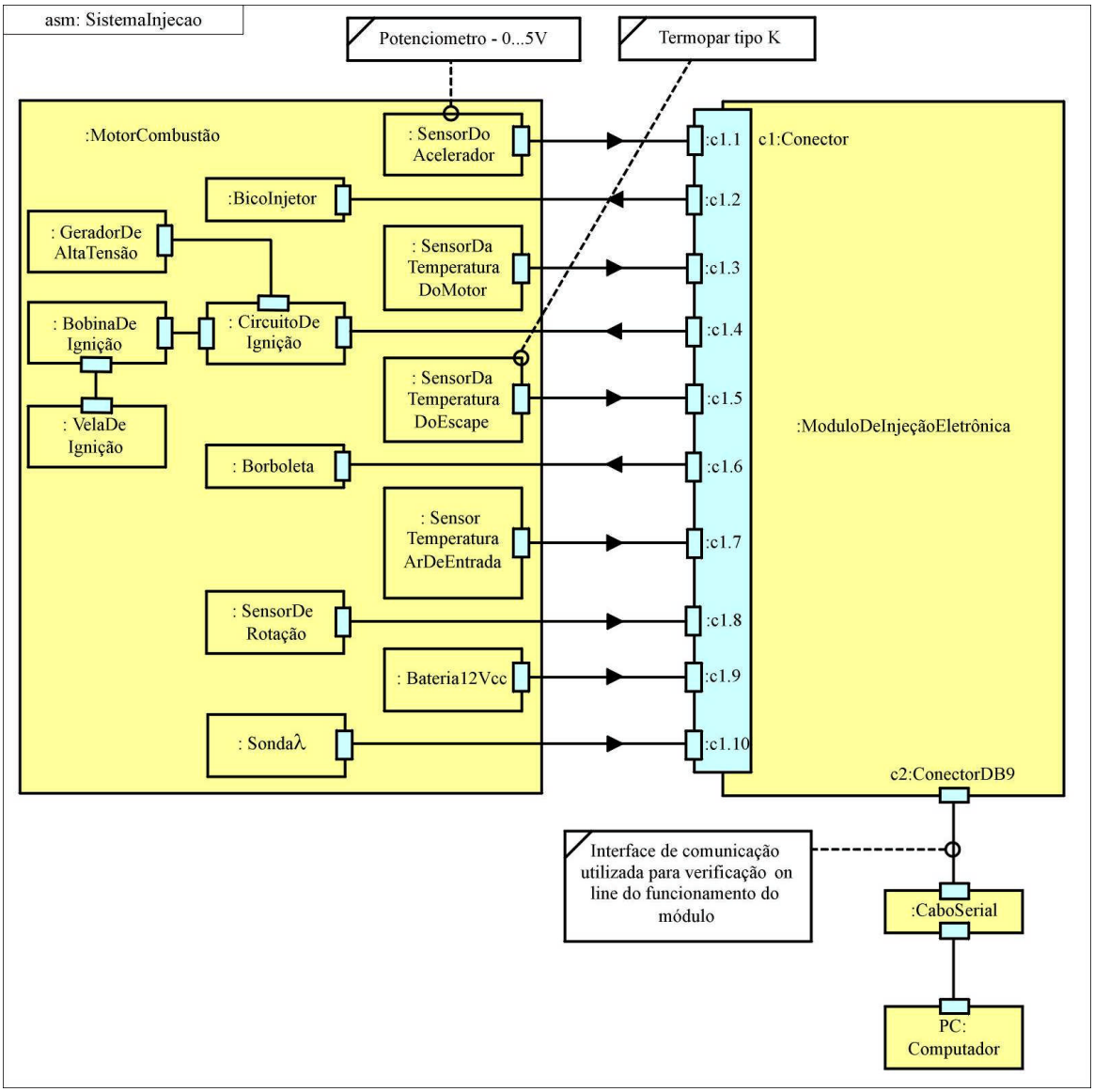


Figura 35: Diagrama de Montagem do sistema de injeção

No Diagrama de Montagem da figura 36, o módulo de injeção é modelado com detalhes, destacando suas principais partes, conexões e sinais elétricos. O processador é representado como um componente de hardware, possibilitando a identificação de algumas restrições elétricas e físicas, como tensão de alimentação, número de pinos e a necessidade de periférico específico.

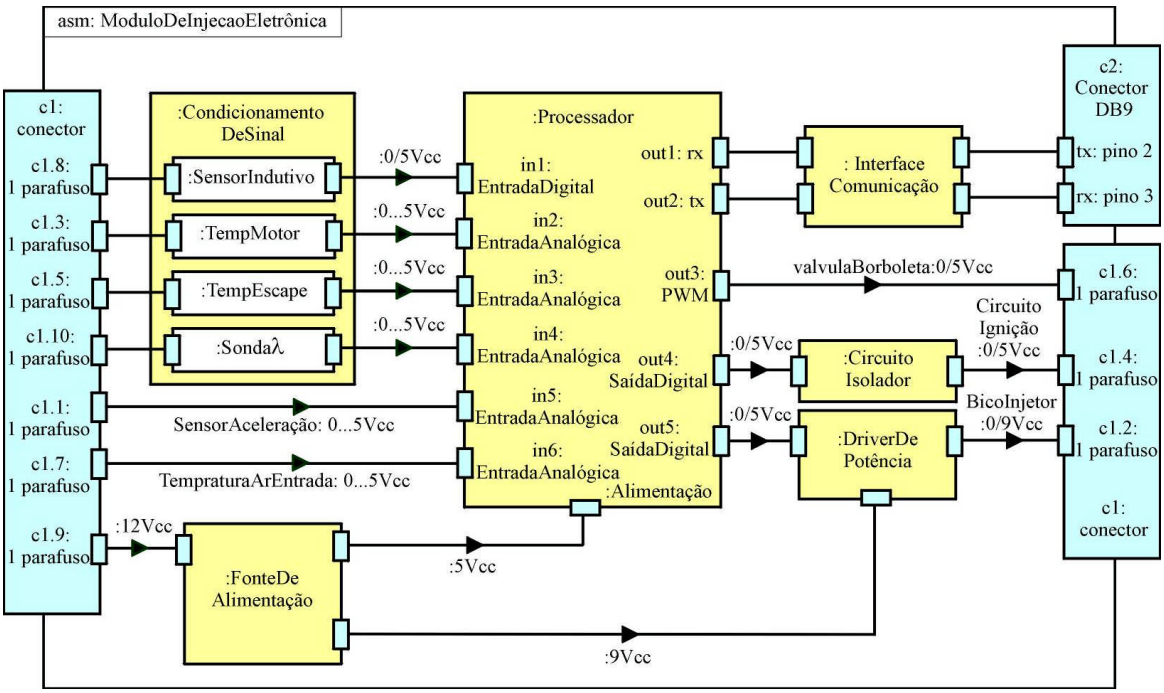


Figura 36: Diagrama de Montagem do modulo de injeção eletrônica

Entretanto, para modelar com mais consistência as características específicas do processador é desenvolvido um Diagrama de Montagem do processador que modela com detalhes os periféricos do processador, como ilustra a figura 37.

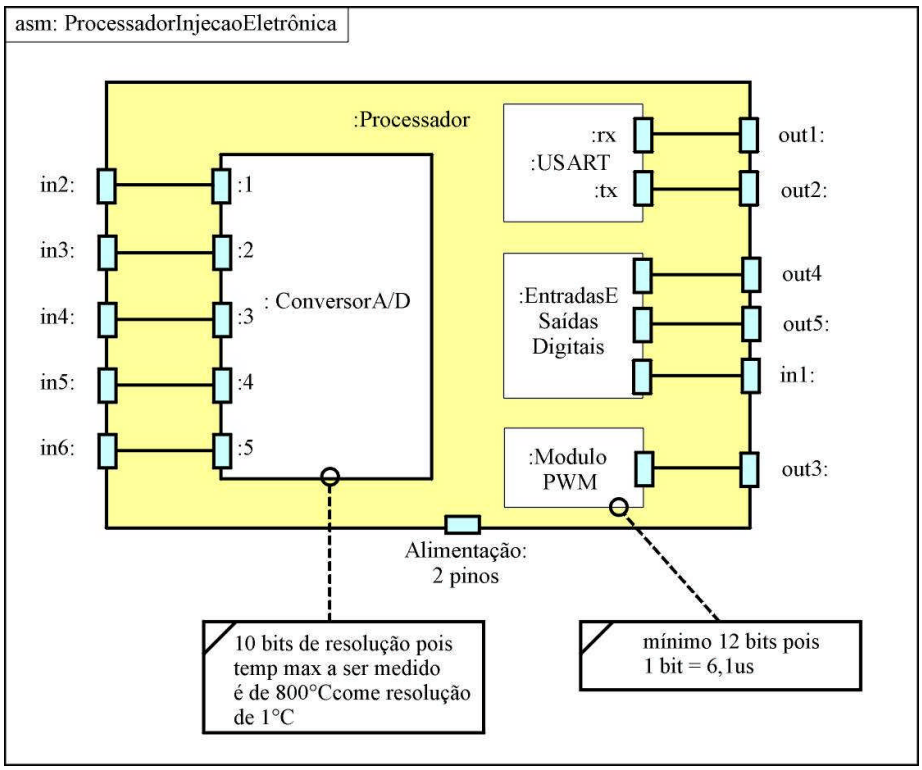


Figura 37: Diagrama de Montagem do processador

6.1.2.5 Diagrama Paramétrico

O Diagrama Paramétrico modela as equações envolvidas no desenvolvimento do sistema embutido. A figura 38 ilustra o Diagrama Paramétrico do sistema de injeção eletrônica, destacando o cálculo da mistura de ar/combustível para o motor a combustão.

Como pode ser observado, o Diagrama Paramétrico descreve os cálculos que deverão ser implementados no processador embutido.

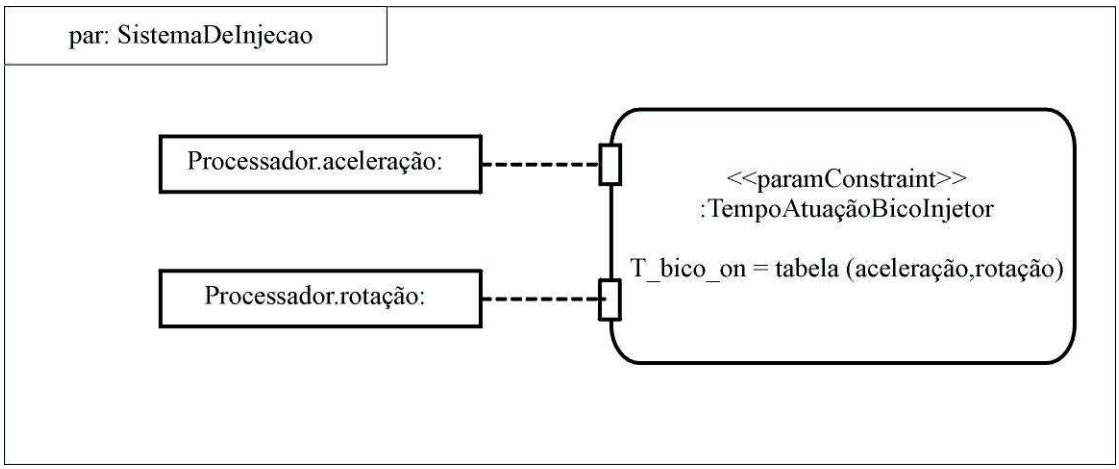


Figura 38: Diagrama Paramétrico do sistema de injeção eletrônica

6.1.2.6 Diagrama de Atividades

A modelagem da parte de software do sistema embutido para injeção eletrônica fica a cargo do Diagrama de Atividades. A figura 39 ilustra um Diagrama de Classes para representar todos os Diagramas de Atividades que compõe o sistema de injeção.

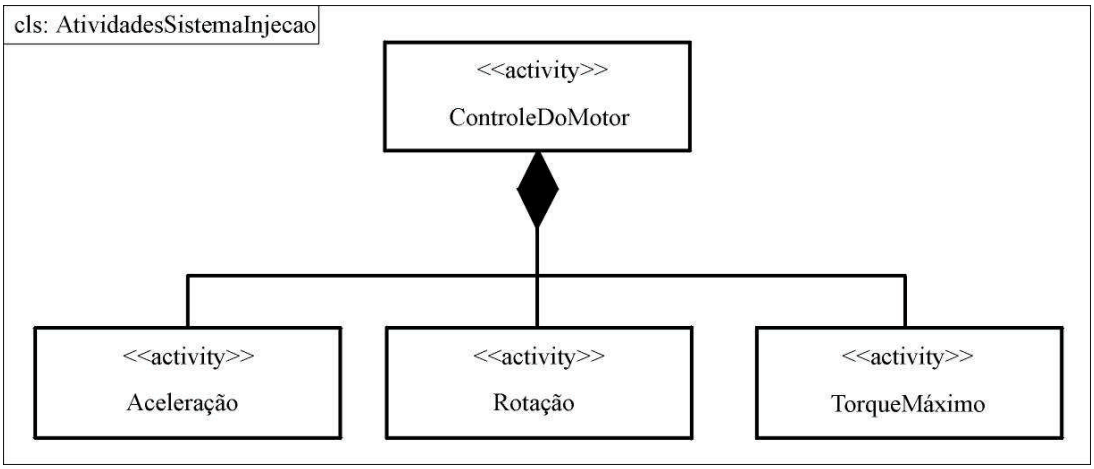


Figura 39: Diagrama de Classes para as atividades

O Diagrama de Atividades da figura 40 ilustra a classe Controle do motor, a qual modela o programa principal do módulo de injeção. É possível identificar as rotinas de Aceleração, Torque máximo e a interrupção da Rotação, que são referenciadas no Diagrama de Classes para as atividades.

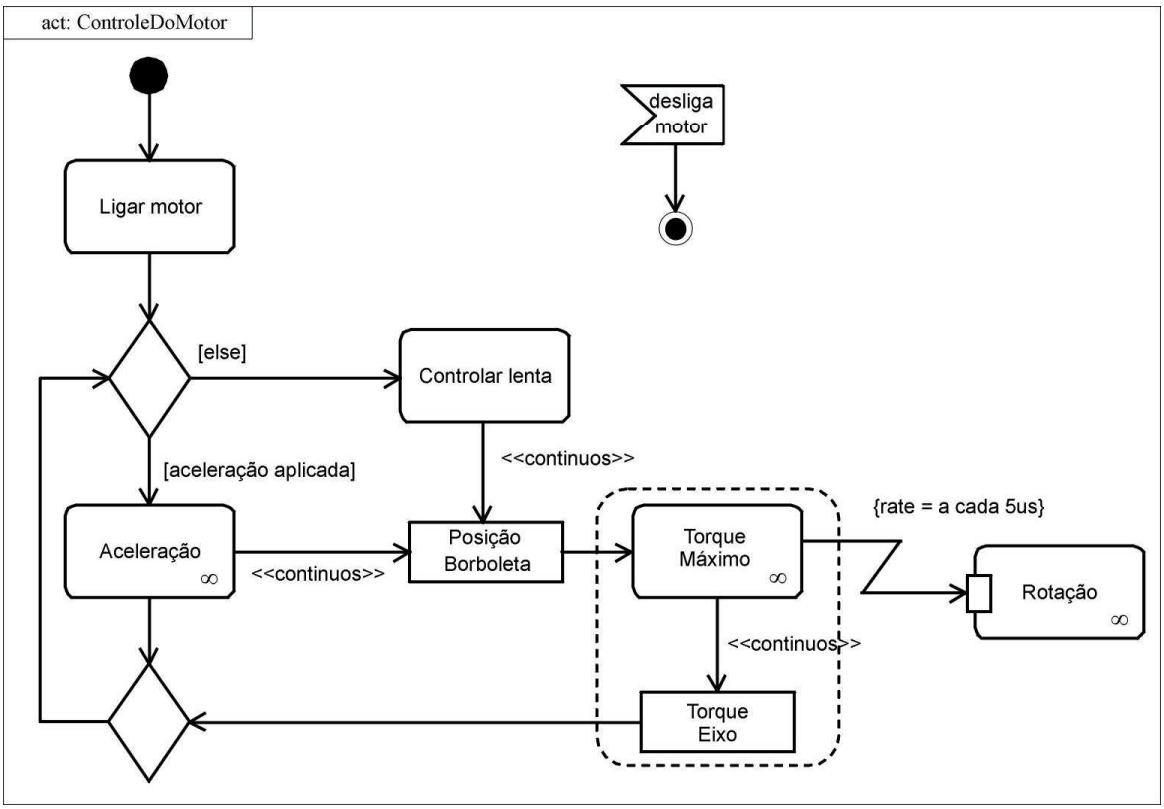


Figura 40: Diagrama de Atividades do controle da injeção eletrônica

6.1.2.7 Diagrama de Tempo

O Diagrama de Tempo modela a restrição informal em um gráfico bidimensional de fácil compreensão. Este diagrama auxilia na compreensão de determinado requisito e é de grande utilidade na implementação do sistema.

A figura 41, ilustra o Diagrama de Tempo para o sinal de controle aplicado a válvula “borboleta”. O Diagrama de Tempo da figura 42, especifica o sinal de controle para o bico injetor.

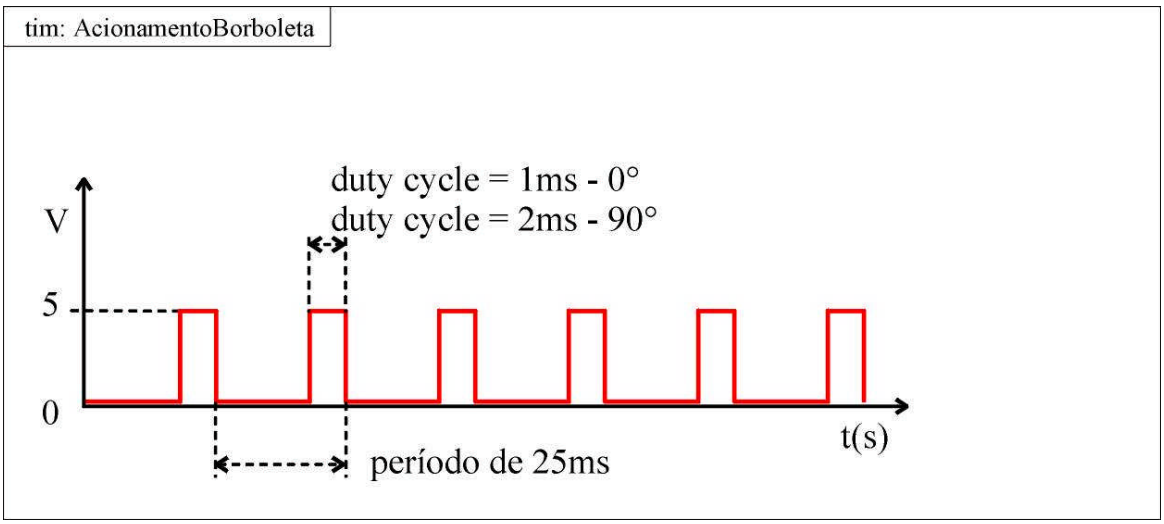


Figura 41: Diagrama de Tempo do acionamento da borboleta

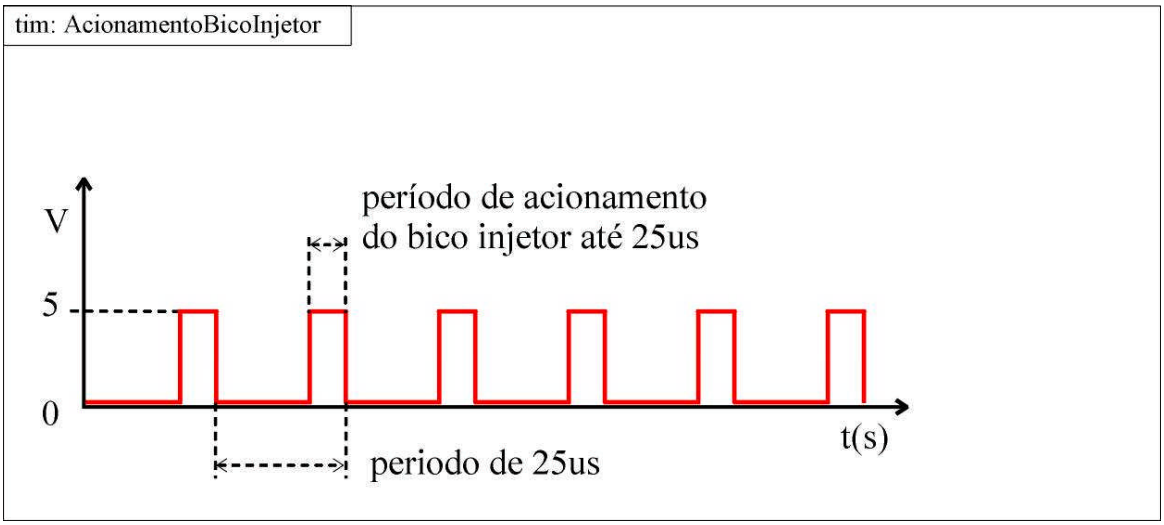


Figura 42: Diagrama de Tempo do bico injetor

6.2 Relé para Proteção de Motores Elétricos

O relé para proteção de motor elétrico é um estudo de caso hipotético. Ele foi selecionado devido a ascensão dos Centros de Controle de Motores (CCM) inteligentes e das redes industrias de comunicação. O relé de proteção é utilizado para monitoramento e proteção de motores elétricos de indução.

O motor elétrico é uma máquina que transforma energia elétrica em energia mecânica, usualmente disponibilizada num eixo em rotação, como mostra a figura 43 [38].

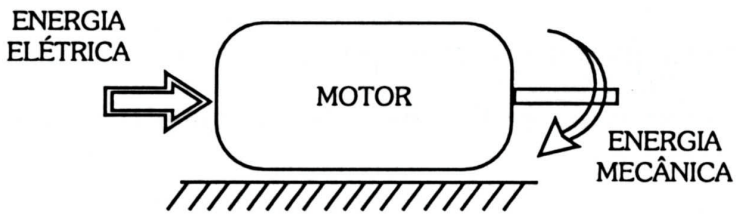


Figura 43: Transformação de energia no motor

O motor elétrico é a máquina mais utilizada na indústria, no meio rural, no comércio, nos serviços e nas residências. Isto se deve às qualidades inerentes da energia elétrica, ou seja, trata-se de uma energia limpa, de baixo custo e de fornecimento instantâneo. Pelo lado dos motores, eles são fabricados desde potências minúsculas de alguns *watts*, até potências gigantescas de milhares de quilowatts. Suas características operacionais atendem a quaisquer tipos de cargas, aliadas a um rendimento na transformação de energia. A instalação e manutenção são razoavelmente simples. É uma máquina extremamente segura. Sob o ponto de vista econômico, são imbatíveis frente a quaisquer outros tipos de motores [38].

Existem diversos tipos de motores elétricos, os quais são divididos em duas grandes famílias. A primeira é dos motores acionados por corrente alternada (motores ca). A segunda é a dos motores acionados por corrente contínua (motores cc) [38].

O motor assíncrono, também conhecido como motor de indução (ou gaiola), como ilustrado na figura 44, é o mais utilizado no meio industrial. Opera com velocidade constante, mas pode variar a velocidade com a variação da carga. Com a evolução da tecnologia dos conversores de freqüência, ele veio a substituir em grande parte os motores de corrente continua [39].

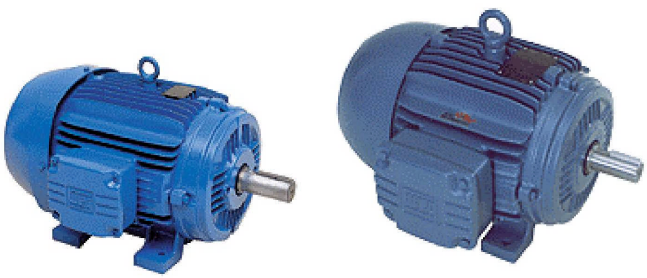


Figura 44: Motor assíncrono

O relé monitora a corrente elétrica e a temperatura do motor de indução e implementa as seguintes proteções: sobrecarga de corrente, desbalanceamento de corrente, falta de fase e sobre temperatura, como ilustra a figura 45.

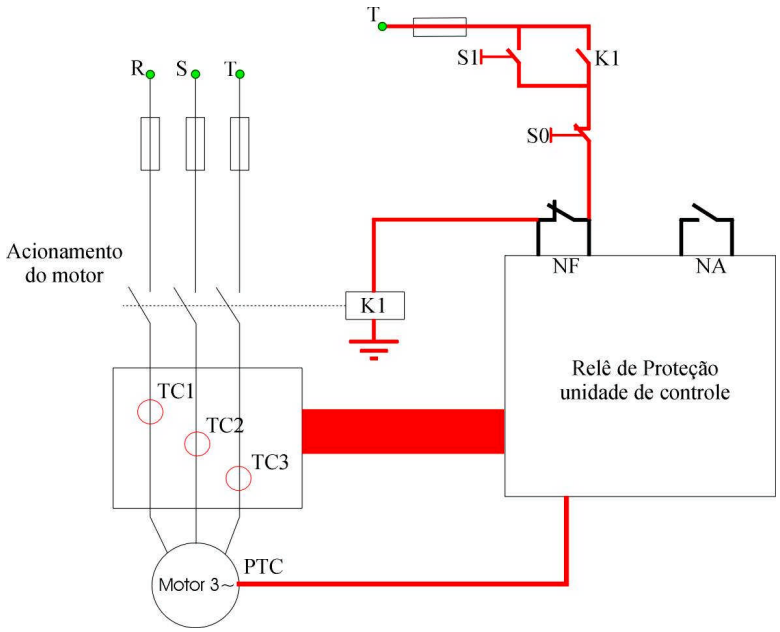
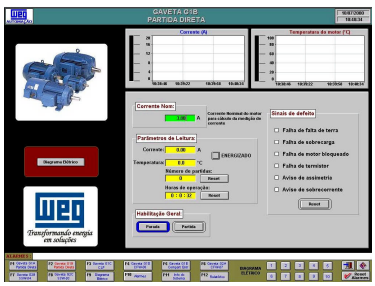


Figura 45: Esquema de ligação do relé de proteção

Além de proteger o motor elétrico, o relé pode ser conectado à rede de comunicação industrial, disponibilizando todas as informações referentes ao motor para o nível da automação mais elevado.

Segundo [40], graças ao desenvolvimento tecnológico dos dispositivos de proteção e comando e o aparecimento das redes industriais, surgem o CCM (Centro de Controle de Motores) inteligente, que se caracteriza pela utilização de componentes com capacidade de comunicação via rede como o relé de proteção, inversor de frequência ou *soft starter,* como ilustra a figura 46.



Software Supervisório

Figura 46: CCM inteligente

O CCM inteligente proporciona vantagens como otimizar o processo industrial no qual os motores são empregados, maximizar a produção, minimizar os riscos de operação. Outra característica importante: a facilidade de comunicação possibilita o gerenciamento de dados e a troca de informações sobre motores, aumentando a confiabilidade do processo através de uma visão ampla do seu comportamento. Tudo isso centralizado em uma sala de controle [40].

Com a possibilidade de inserção das partidas de motores à uma rede de comunicação, é possível diagnosticar rapidamente as causas que originaram a parada dos motores e consequentemente do processo, diminuindo o tempo de máquina parada.

O relé de proteção é o responsável pela característica do CCM como inteligente. Entretanto, precisa estar associado com acionamentos para manobra do motor.

6.2.1 Descrição Informal

Na etapa de análise de requisitos foi realizada uma de pesquisa de mercado, verificando as características dos relés de proteção disponíveis. Todas as informações pesquisadas foram compiladas em uma especificação básica, a qual descreve todas as características, requisitos e funcionalidades de maneira informal.

O relé é baseado em um sistema microcontrolado, onde a CPU é responsável pelo processamento de todos os sinais e implementação das funções de proteção do motor, funções lógicas, funções de controle e comunicação em rede industrial. A figura 47 ilustra o diagrama de blocos do relé.

O relé de proteção é composto por uma unidade de controle e uma unidade de medição com faixa de corrente de 0,2 a 20A.

O relé apresenta as seguintes funções de proteção para motores trifásicos e monofásicos: classe de disparo ajustável, sobrecarga, falta de fase, desbalanceamento, rotor bloqueado, fuga à terra, proteção térmica via sensor de temperatura PTC.

As funções de controle do relé são as seguintes: partida direta, partida estrela-triângulo, partida reversora, motor com duas velocidades (*dahlander*), operação como relé

(monitoramento) e modo transparente (acionamento das entradas e saídas via rede de comunicação).

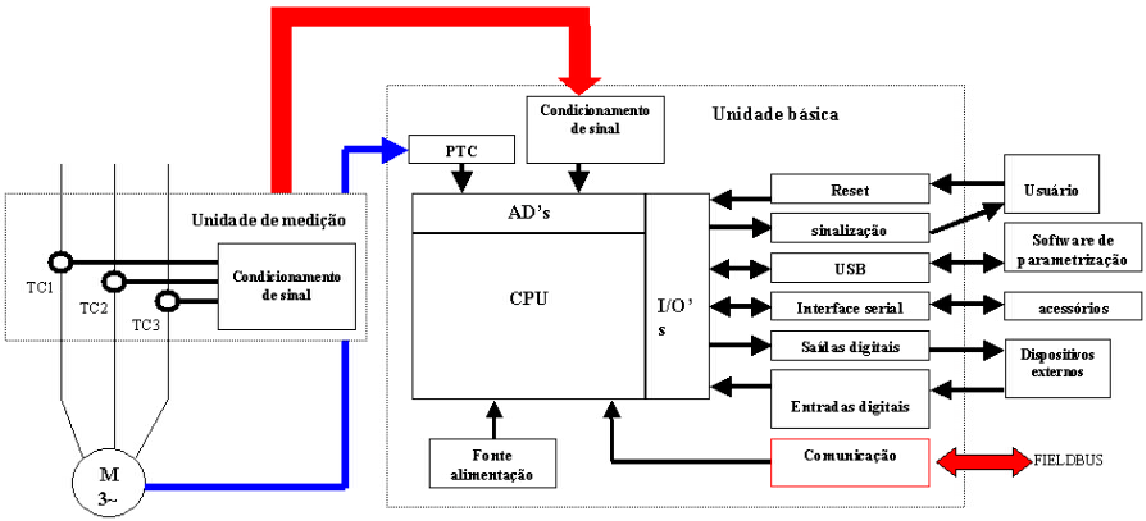


Figura 47: Diagrama de bloco do relé de proteção

O relé de proteção disponibiliza as seguintes informações: corrente de cada fase do motor em porcentagem e RMS, capacidade térmica utilizada em porcentagem da corrente ajustada, número de desarmes por tipo de falha, número de partidas, horas de funcionamento do motor, nível de desbalanceamento e temperatura do motor via PTC.

O relé de proteção pode ser conectado aos seguintes protocolos de comunicação: DeviceNet, ModBus-RTU e ProfiBus-DP.

6.2.2 Descrição em Diagramas

A próxima etapa é a especificação do sistema, transformando a descrição informal do mesmo em especificação baseada em diagramas. Na modelagem do sistema embutido para o relé de proteção, foram utilizados os seguintes diagramas da SysML: Diagrama de Casos de Uso, Diagrama de Classes, Diagrama de Requisitos, Diagrama de Montagem, Diagrama Paramétrico, Diagrama de Atividades e Diagrama de Tempo.

6.2.2.1 Diagrama de Casos de Uso

O Diagrama de Casos de Uso, ilustrado na figura 48, descreve o comportamento externo do sistema, apresentando as funcionalidades e serviços do relé de proteção do ponto de vista do usuário.

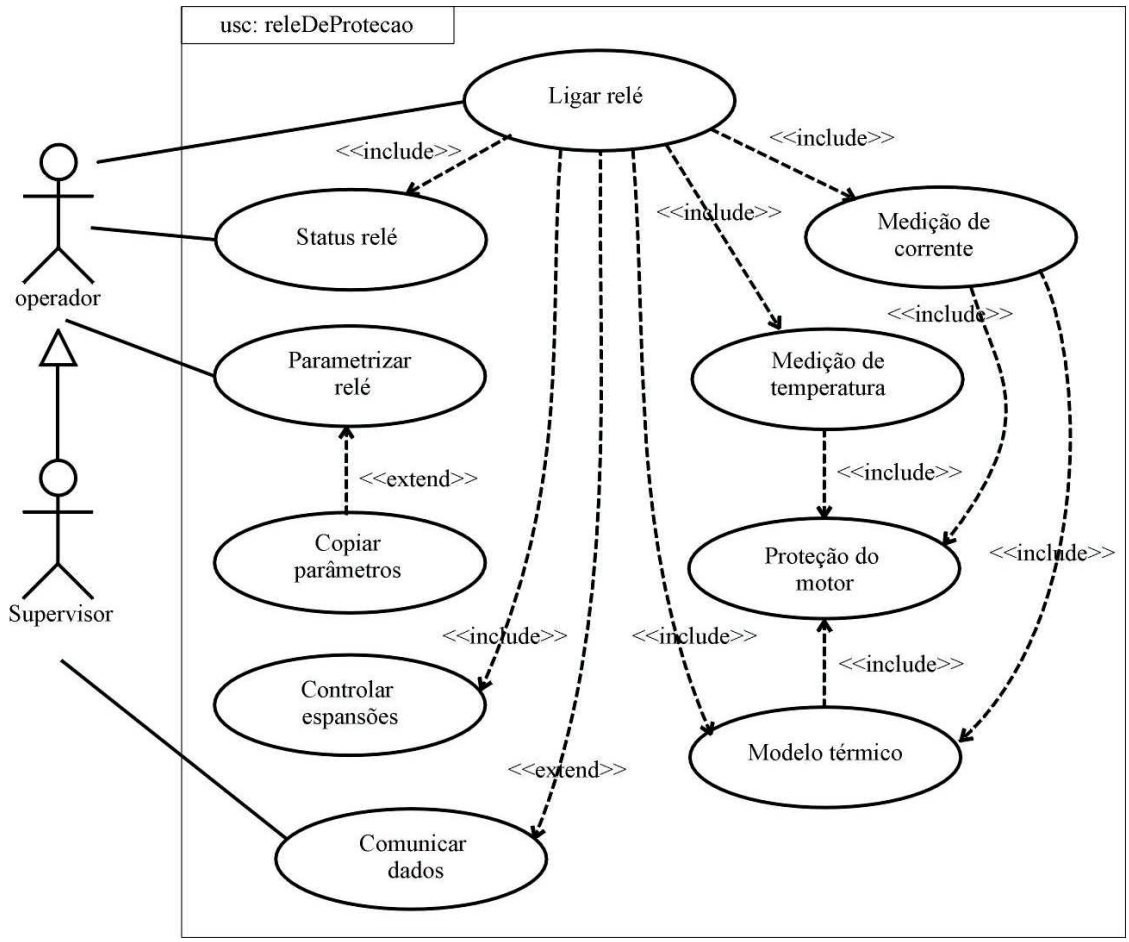


Figura 48: Diagrama de Casos de Uso do relé de proteção

O Serviço Ligar relé é responsável pelo disparo de todos os serviços de operação do relé. Os serviços de Parametrizar relé e Copiar parâmetros são ativados diretamente pelo operador ou pelo supervisor da planta industrial. Os serviços Medição de corrente, Medição de temperatura, Proteção do motor e Modelo térmico, implementam a seção de proteção do motor elétrico. O serviço de Medição de corrente está incluído na serviço de Proteção do motor e Modelo térmico. O serviço Comunicar dados oferece o suporte para a conexão com a rede de comunicação, entretanto este serviço é opcional, pois o relé pode trabalhar sem estar conectado a uma rede industrial. O serviço *Status* disponibiliza todas as informações referentes ao relé e ao motor. O serviço Controlar expansões é responsável pela comunicação entre os módulos de expansão e o relé.

6.2.2.2 Diagrama de Classes

O Diagrama de Classes, ilustrado na figura 49, descreve o relé de proteção de um ponto de vista estático. Com este diagrama é possível visualizar todas as partes que compõem o sistema e o relacionamento entre as mesmas.

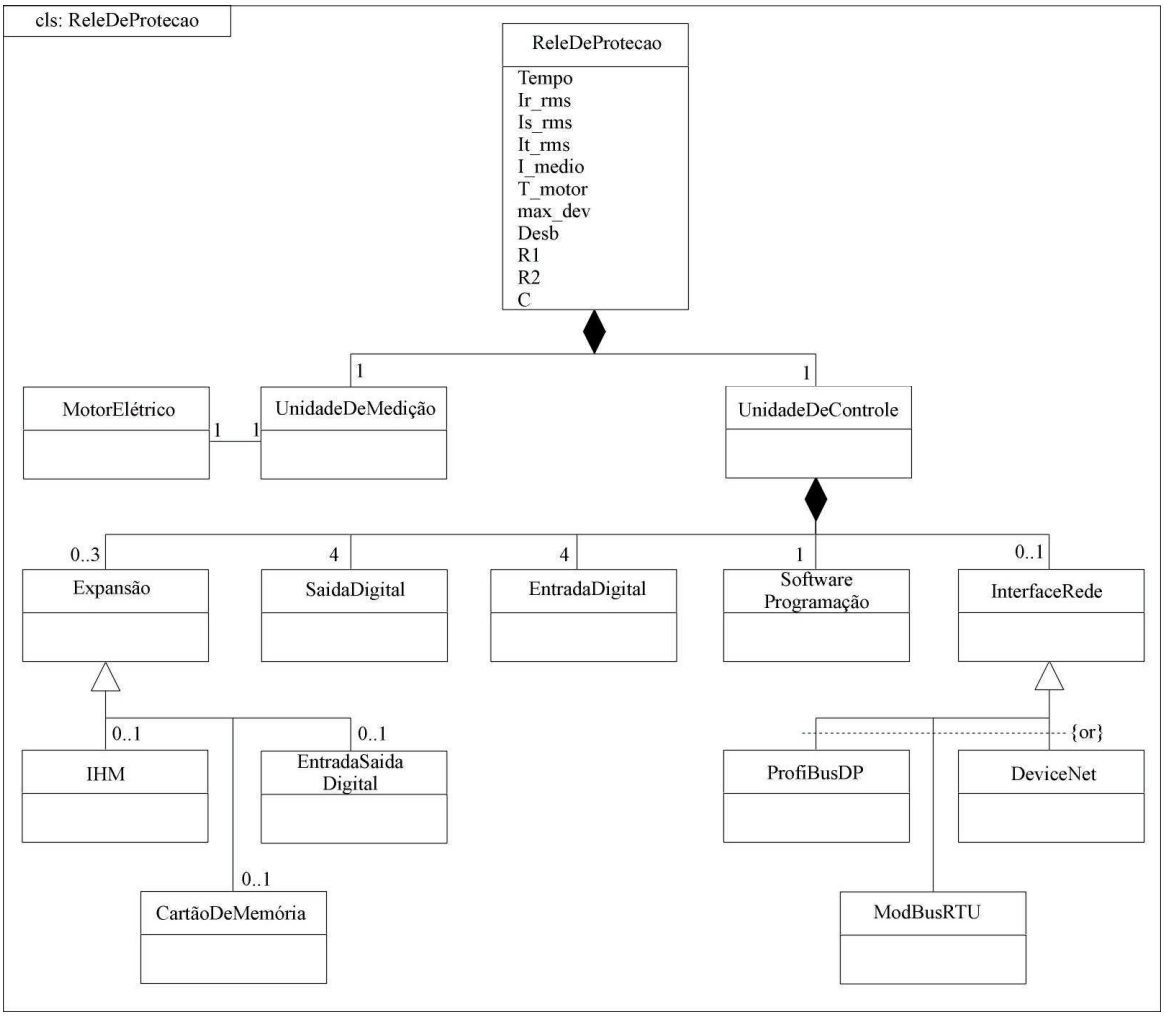


Figura 49: Diagrama de Classes do relé de proteção

Através do Diagrama de Classes pode-se visualizar algumas características do relé, como possibilidade de conexão a três protocolos de rede industrial, sendo um de cada vez, conexão de módulos de expansão e apenas um motor pode ser conectado a unidade de medição.

6.2.2.3 Diagrama de Requisitos

O relé de proteção deve atender os requisitos de várias normas técnicas para proteção de motores elétricos como IEC 947-4-1 [41], UL 508 [42] e Nema MG-1 [43],

como ilustra a figura 50. Os requisitos para comunicação e transformador de corrente são gerados a partir da especificação básica do relé de proteção.

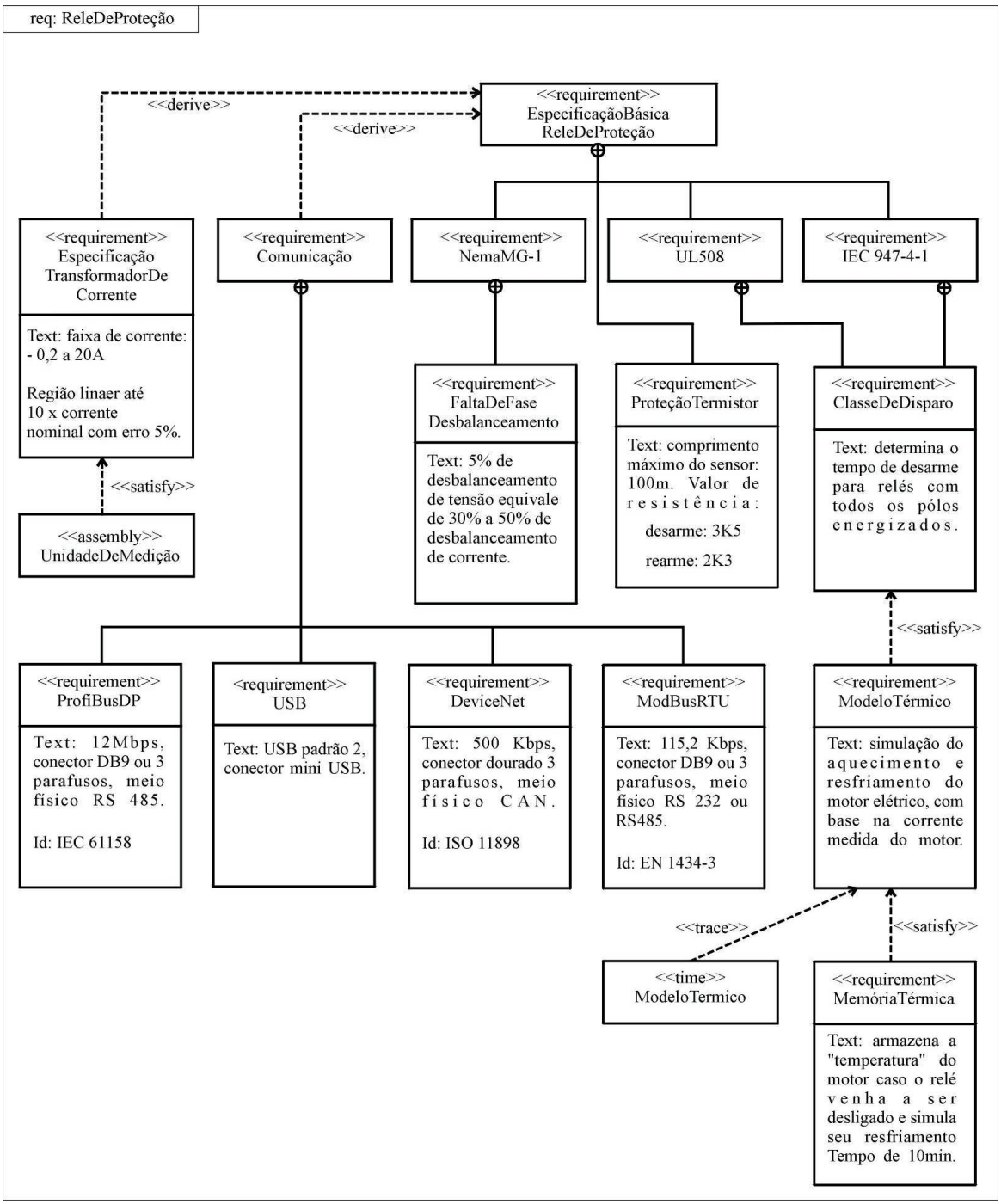


Figura 50: Diagrama de Requisito do relé de proteção

Para cada protocolo de rede de comunicação industrial que o relé pode ser conectado, existem inúmeros requisitos que devem ser atendidos, destacando a camada física e taxa de transferência.

O Diagrama de Tempo do Modelo térmico é utilizado para facilitar a compreensão das características do modelo térmico do motor elétrico utilizado no relé de proteção.

6.2.2.4 Diagrama de Montagem

O Diagrama de Montagem, ilustrado na figura 51, modela todos os conectores e as interfaces do relé de proteção com o sistema de controle do motor. São utilizados quadros de nota para descrever características e informações importantes sobre o sistema.

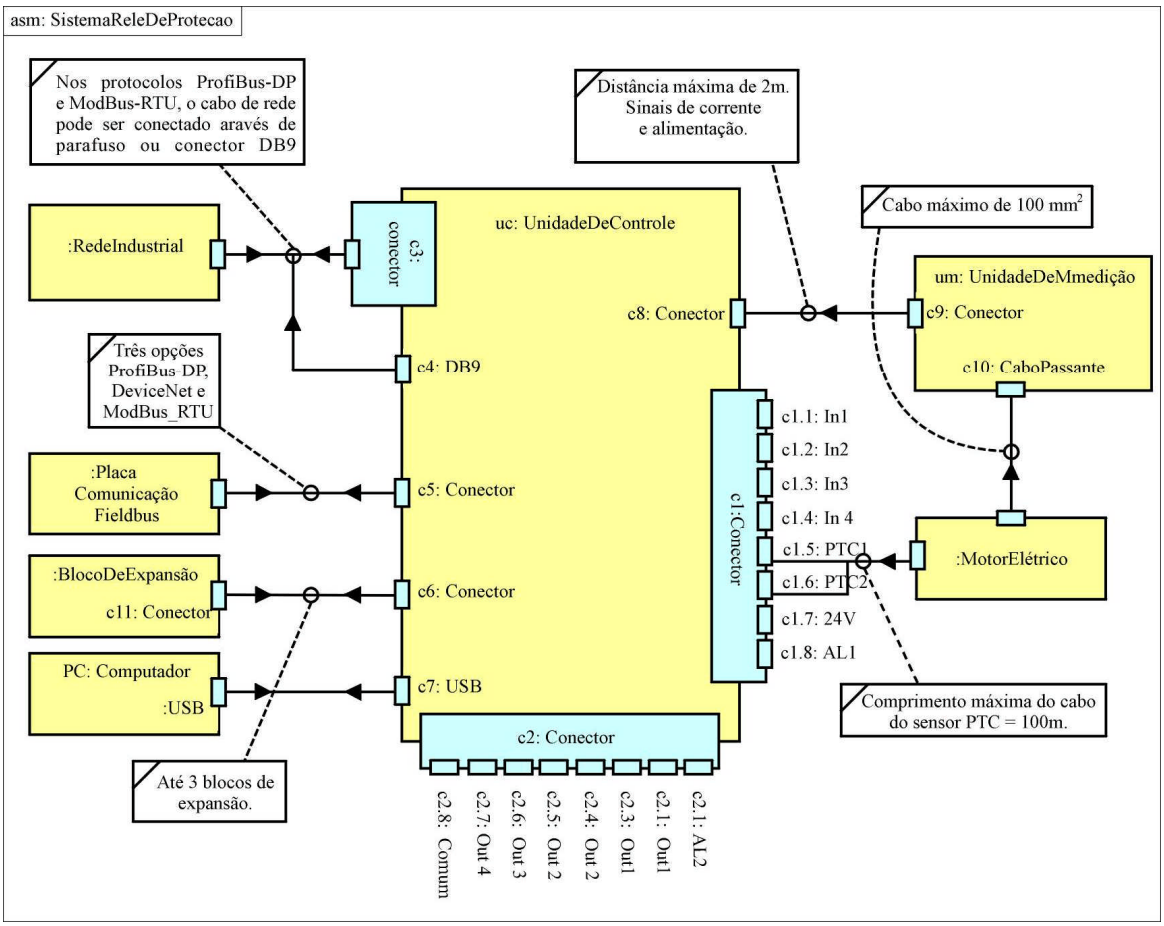


Figura 51: Diagrama de Montagem do sistema completo do relé de proteção

A figura 52 ilustra o Diagrama de Montagem da unidade de medição. São modelados os sinais elétricos de entrada e saída e as partes que compõem a unidade. A corrente elétrica do motor é o sinal de entrada da unidade de medição, a qual é condicionada para um sinal de 0 a 20mA, sinal de saída, no bloco de condicionamento de sinal e enviada para a unidade de controle. O bloco sensor de corrente informa que existe 3 transformadores de corrente (TC), um para cada fase do motor.

A unidade de medição recebe da unidade de controle energia para alimentação dos circuitos eletrônicos

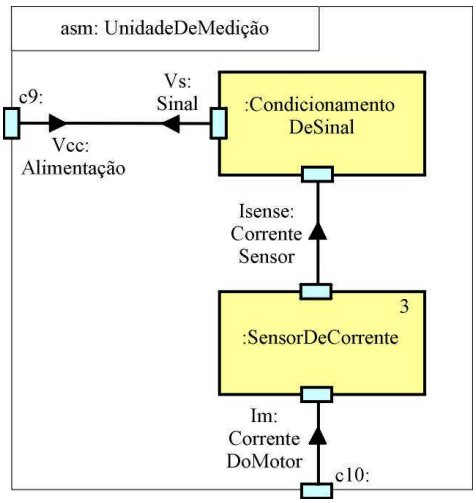


Figura 52: Diagrama de Montagem da unidade de medição do relé de proteção

O Diagrama de Montagem da figura 53, modela a unidade de controle detalhando suas partes, conectores, interfaces e sinais elétricos. Neste diagrama pode-se visualizar o bloco da CPU, o qual modela as características e requisitos que o processador embutido deve apresentar. O sistema possui dois barramentos, um de dados e outro de endereçamento, conectando memórias, controlador de USB e placa de comunicação.

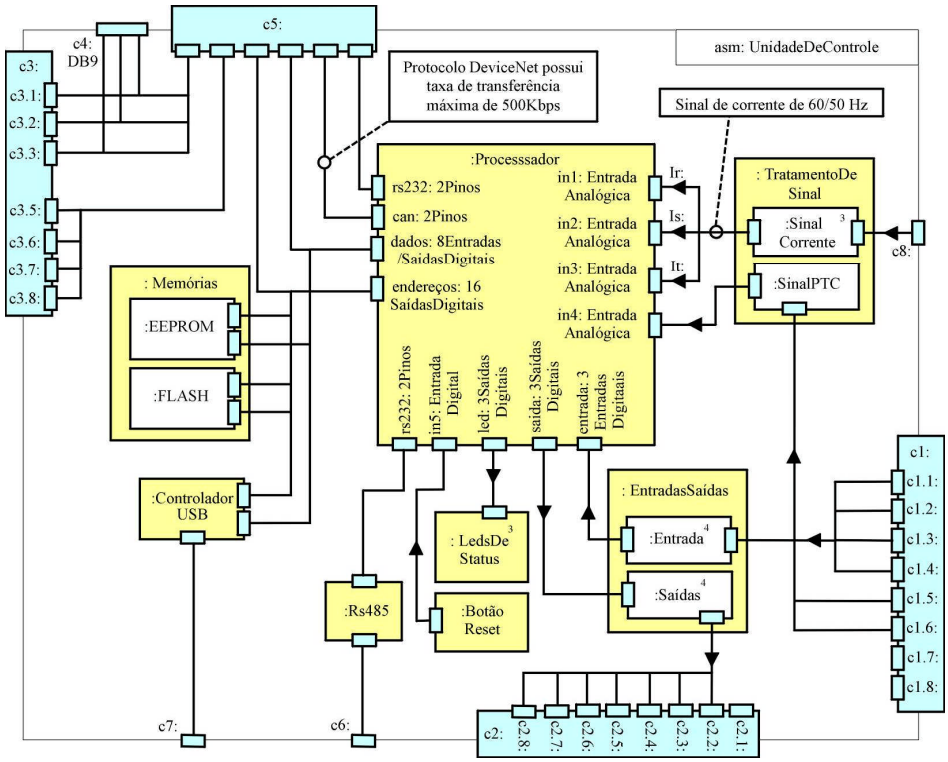


Figura 53: Diagrama de Montagem da unidade de controle do relé de proteção

Entretanto, para modelar com mais consistência as características específicas do processador é desenvolvido um Diagrama de Montagem do processador que modela com detalhes os periféricos do processador, como ilustra a figura 54.

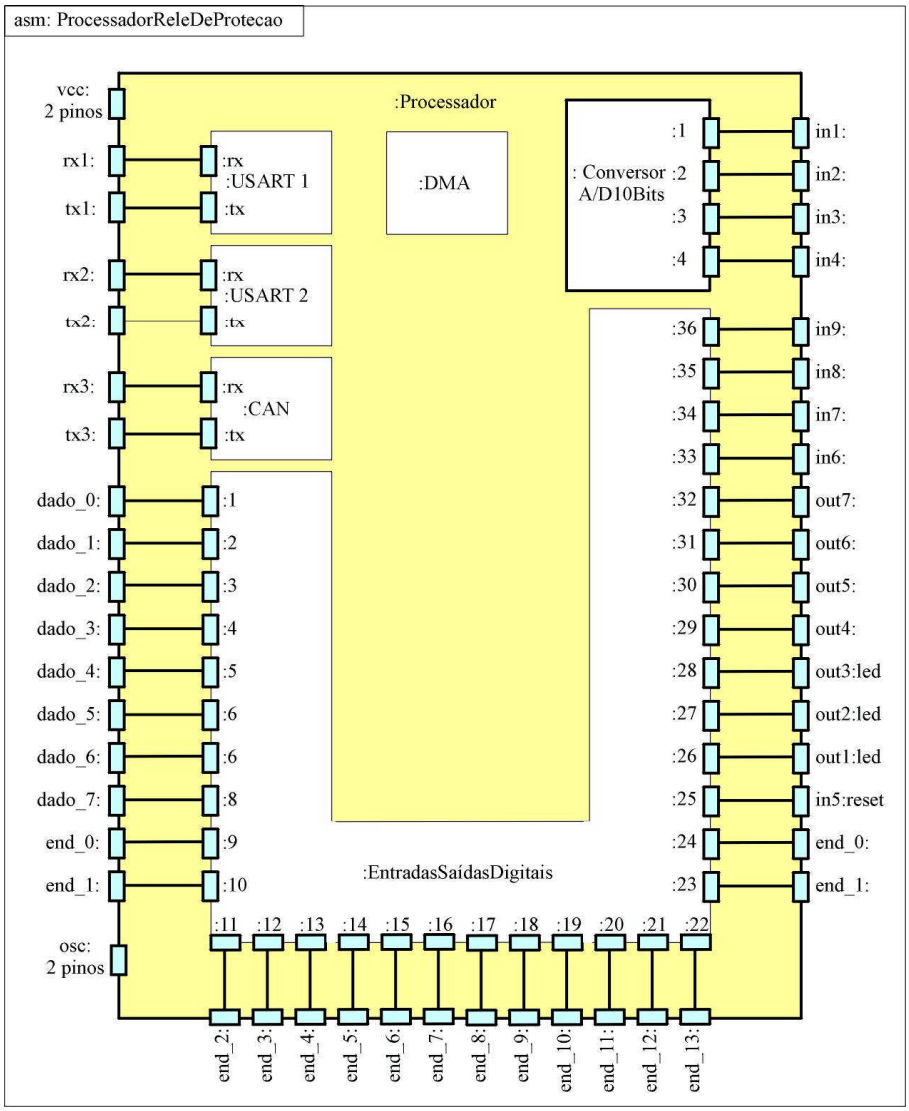


Figura 54: Diagrama de Montagem do processador da unidade de controle

6.2.2.5 Diagrama Paramétrico

O Diagrama Paramétrico modela todas as equações envolvidas no desenvolvimento do sistema embutido, no caso o relé de proteção. A figura 55, ilustra o Diagrama Paramétrico da unidade de controle, destacando o cálculo da corrente eficaz (RMS) para cada fase do motor, o cálculo de desbalanceamento de corrente e o modelo térmico do motor elétrico.

O cálculo do desbalanceamento de corrente utiliza o valor eficaz de cada corrente e a corrente média do motor, aplicando os requisitos da norma Nema MG-1 [43] para identificar se o valor calculado de desbalanceamento está dentro dos limites.

O modelo térmico do motor utiliza uma função para descrever o aquecimento e resfriamento do motor, utilizando o valor médio da corrente do motor como entrada. O modelo térmico deve atender os requisitos de tempo e classe de desarme das normas IEC 947-4-1 [41] e UL 508 [42].

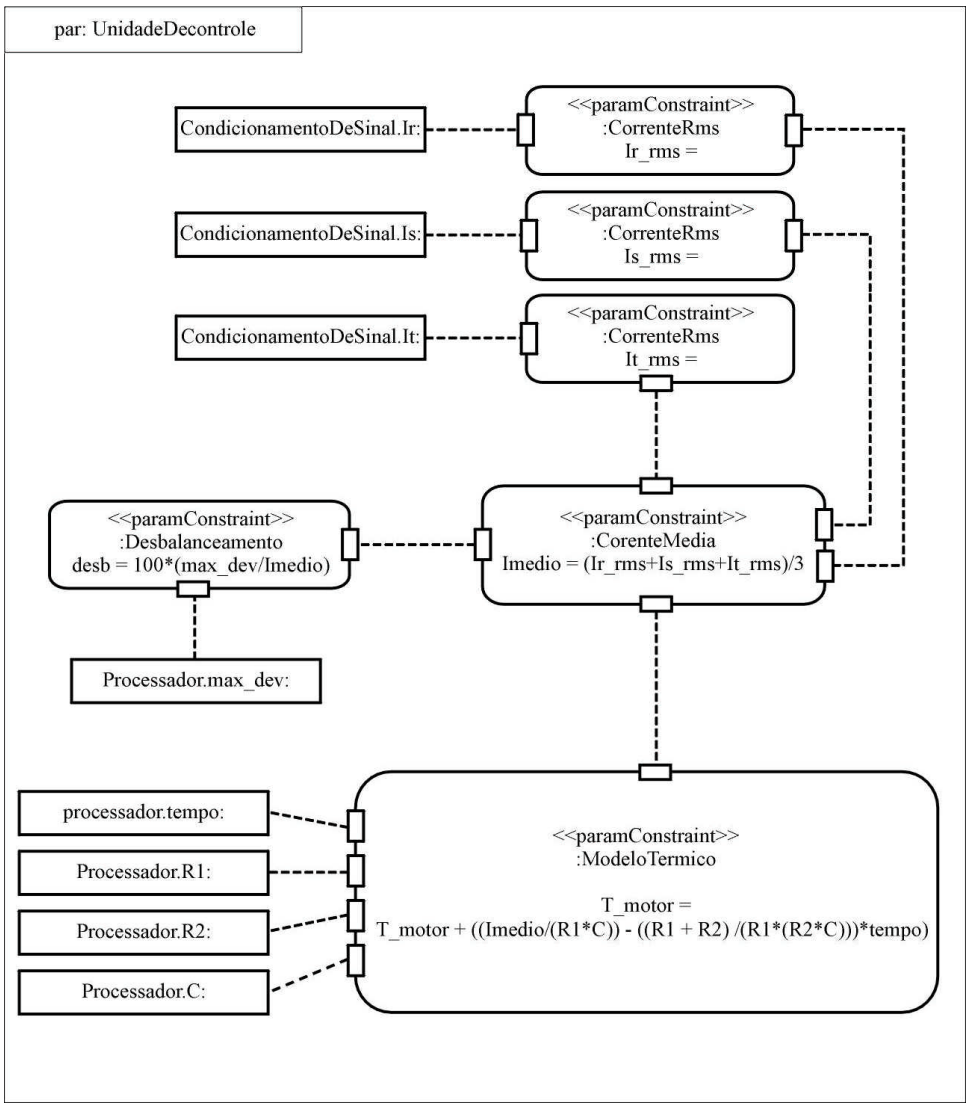


Figura 55: Diagrama Paramétrico da unidade de controle do relé de proteção

A figura 56, ilustra o Diagrama Paramétrico dentro do Diagrama de Montagem da unidade de medição. Este Diagrama Paramétrico descreve a marcha de cálculo para os sensores de corrente e componentes do circuito de condicionamento de sinal.

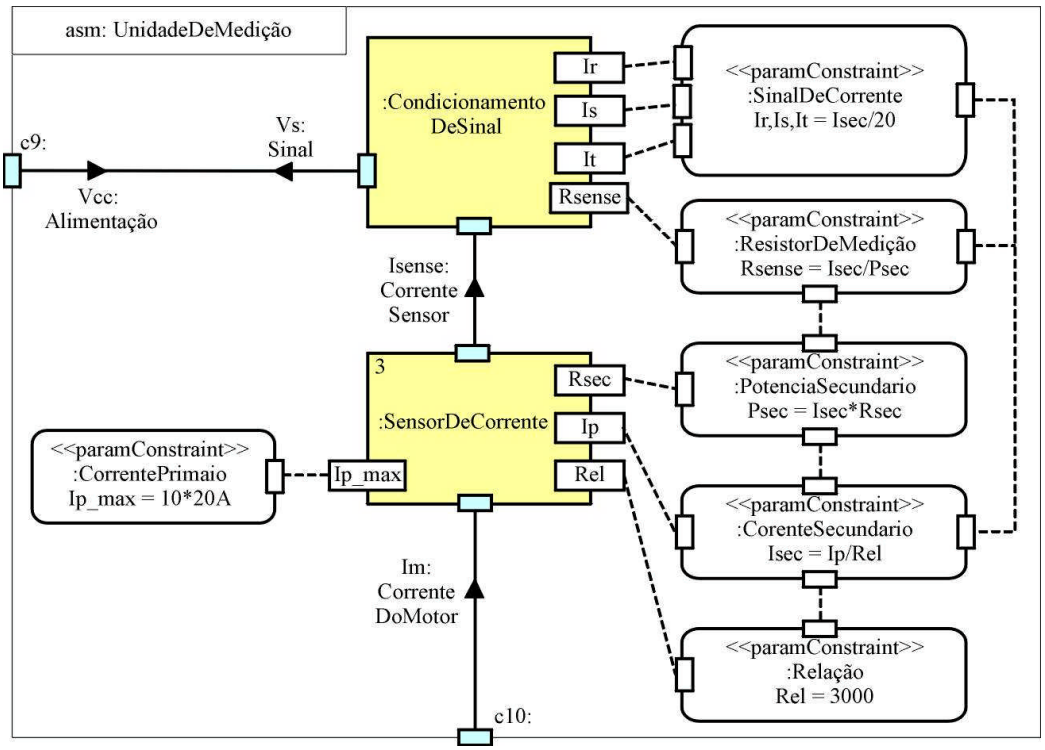


Figura 56: Diagrama Paramétrico em conjunto com o Diagrama de Montagem da unidade de medição do relé de proteção

Como pode ser observado, o Diagrama Paramétrico da unidade de controle descreve os cálculos que deverão ser implementados no processador embutido do relé e o Diagrama Paramétrico da unidade de medição descreve os cálculos para o projeto de *hardware*.

6.2.2.6 Diagrama de Atividades

O Diagrama de Atividades é utilizado para a modelagem da parte de *software* do relé de proteção. A figura 57, ilustra um Diagrama de Classes para representar os Diagramas de Atividades do relé de proteção.

A figura 59, ilustra o Diagrama de Atividades do controle do relé, o qual modela o programa principal do relé de proteção, destacando as rotinas de interrupção para botão Reset, USB, comunicação com rede e controle de expansão. Também são modeladas rotinas para executar o programa do usuário, desarmar o relé, teste de hardware, gravação de parâmetros e medição da corrente do motor.

6.2.2.7 Diagrama de Tempo

O Diagrama de Tempo ilustrado na figura 58, modela as características de saída do modelo térmico para diferentes múltiplos de corrente do motor.

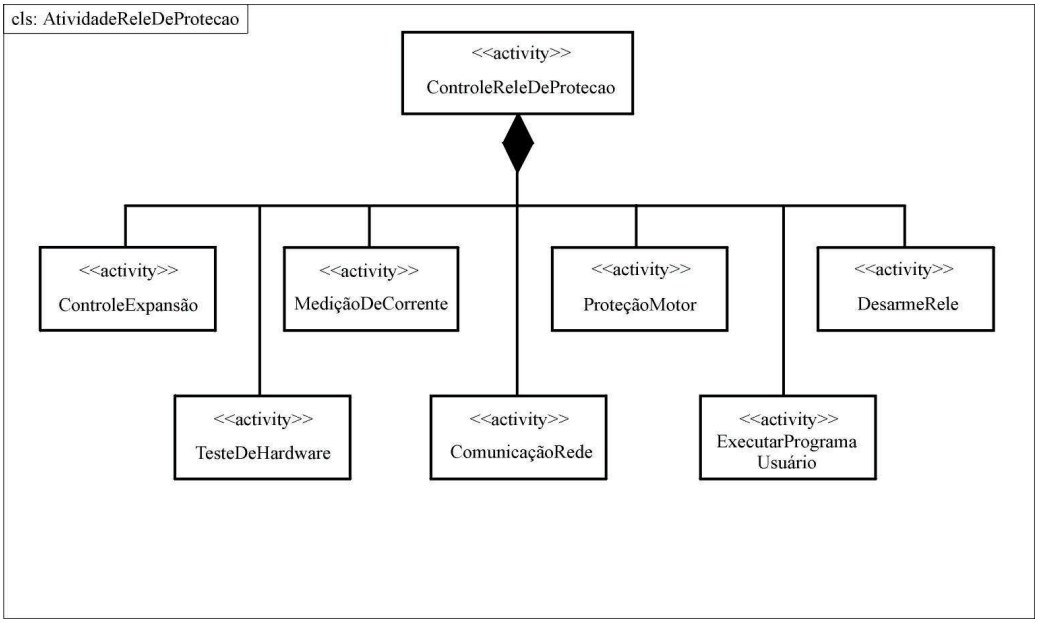


Figura 57: Diagrama de Classe para as atividades do relé de proteção

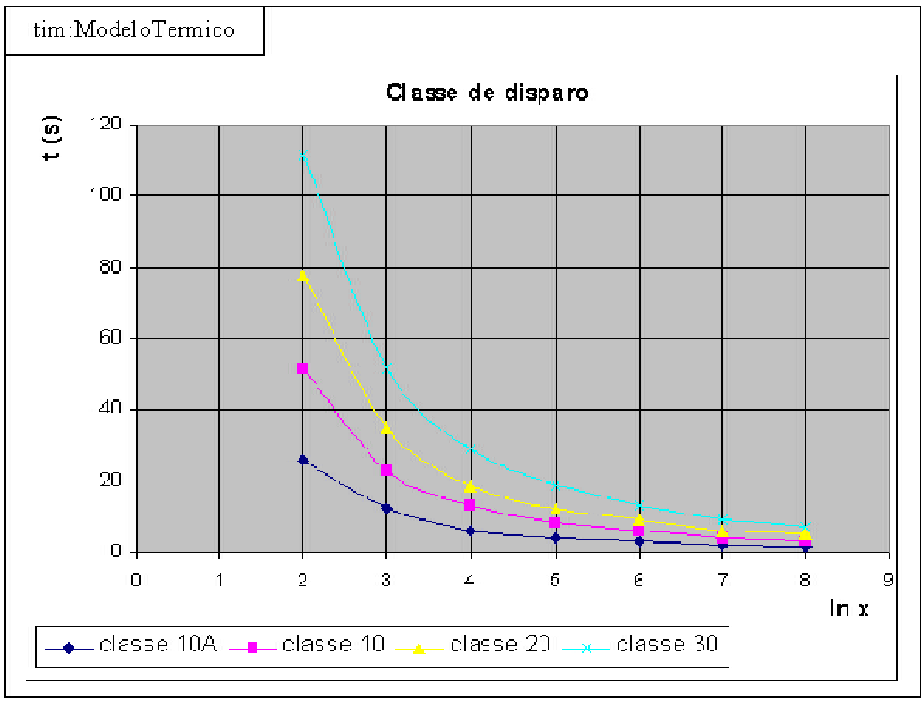


Figura 58: Diagrama de Tempo do modelo térmico do relé de proteção

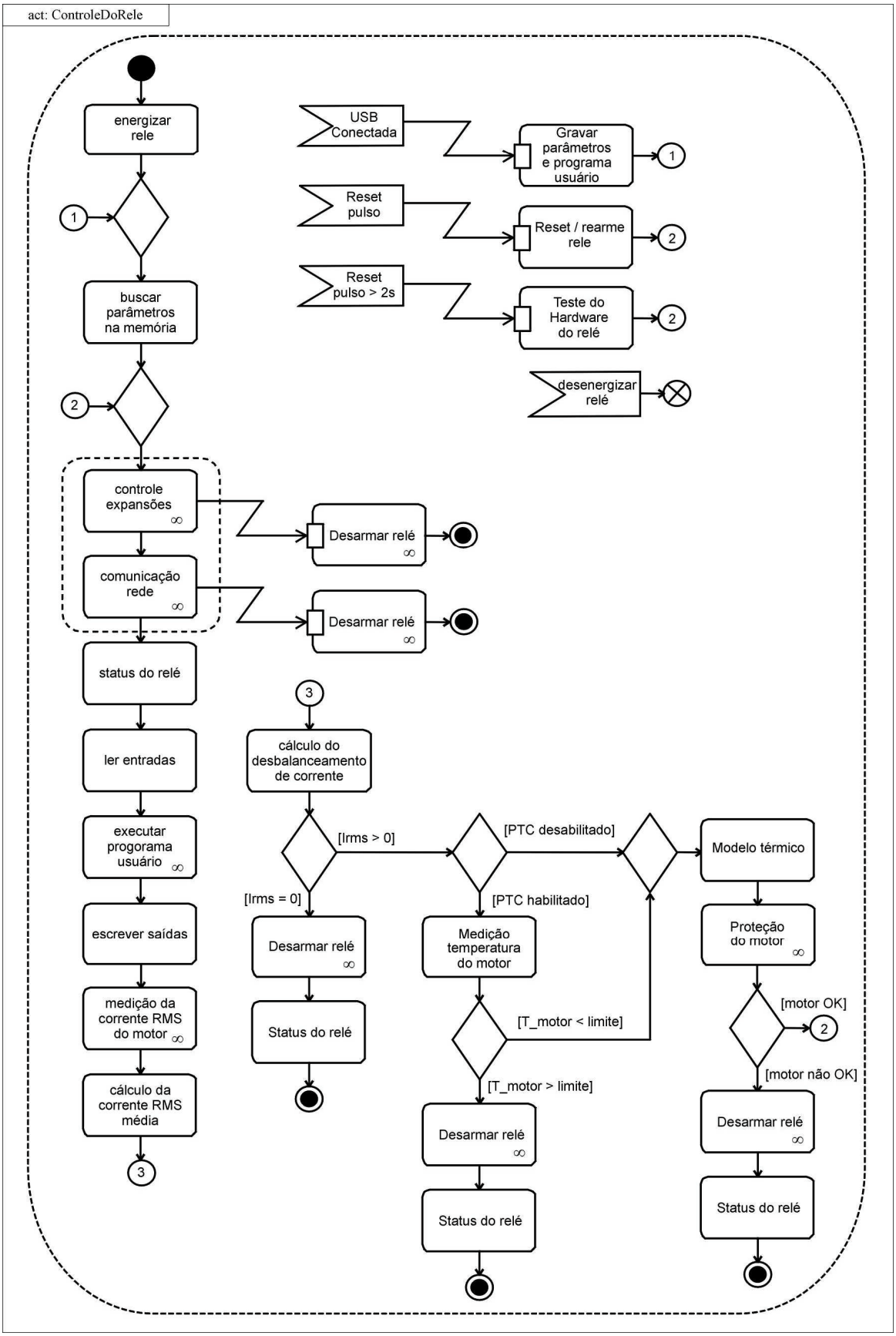


Figura 59: Diagrama de Atividade do controle do relé de proteção

6.3 Conclusão

A SysML fornece uma especificação do sistema embutido, partindo de uma especificação em linguagem natural, que engloba requisitos mecânicos, elétricos, hardware, software, normas técnicas e funções matemáticas.

A modelagem dos estudos de caso apresentou duas etapas bem distintas. A primeira foi o levantamento dos requisitos e o conhecimento do funcionamento do sistema. A Segunda etapa foi a aplicação da linguagem SysML na versão 0.9.

Como SysML é uma simplificação da UML 2 e alguns diagramas são importados na integra, se fez necessário a utilização de material de apoio sobre UML, e modelagem orientada a objeto durante a modelagem dos estudos de caso.

O Diagrama de Requisitos e o Diagrama Paramétrico são diagramas desenvolvidos para a SysML e apresentam função importante na modelagem do sistema. Inicialmente, o Diagrama de Requisitos mostrou-se um grande diferencial em relação a UML, para a modelagem de sistemas embutidos, porém ele apresenta um alto grau de informalidade. Os requisitos são descritos através de texto em linguagem natural, inserindo a possibilidade de inconsistência na extração das informações do diagrama. Entretanto, os requisitos podem ser agrupados de forma clara, documentando sua origem. O Diagrama de Tempo foi utilizado em conjunto com o Diagrama de Requisitos para modelar o sinal elétrico que deve ser aplicado para controlar determinado atuador. Através dele o requisito para controle do atuador ficou modelado livre de inconsistências.

O Diagrama Paramétrico é o diagrama utilizado para a modelagem das funções matemáticas envolvidas no desenvolvimento do sistema embutido. Como são sistemas de aplicação específica, apresentam cálculos para controle e processamento de sinais, que uma vez modelados auxiliam na definição de rotinas com restrições temporais, que influenciam na seleção da arquitetura do processador, assim como ajudam a estimar o tamanho do programa.

Apesar do Diagrama de Montagem estar baseado no Diagrama de Estrutura Composta da UML 2, foi desenvolvido o *NestedConnectorEnd* que é um estereótipo do

ConnectorEnd da UML, possibilitando a modelagem de qualquer conector abrigado em diferentes níveis de uma parte ou porta do sistema.

Construções auxiliares como o *itemFlow* são utilizados para a modelagem dos sinais elétricos e dados dentro do Diagrama de Montagem.

Os Diagramas de Casos de Uso e de Classes, são os mesmos da UML. Como no projeto de sistemas de software, esses diagramas auxiliam a modelagem de sistemas embutidos na etapa inicial, onde o nível de abstração é alto.

A parte de software é modelada através do Diagrama de Atividades, apresentando mais recursos em relação aos tradicionais fluxogramas e suporte para tradução dos diagramas de blocos funcionais (EFFBD), muito utilizados por engenheiros de sistema.

Para estes sistemas, os diagramas utilizados foram aqueles que mostraram os aspectos de interesse. Para outros sistemas, possivelmente a escolha dos diagramas SysML usados seria diferente.

Para os estudos de caso utilizados no capítulo 6, os diagramas de Sequência e Máquina de Estados não forneciam informações relevantes para a seleção do processador, e por isto não foram utilizados.

Capítulo 7

Uso dos Modelos SysML na Seleção do Processador

Os dois estudos de caso foram modelados com o objetivo de subsidiar o levantamento das características técnicas dos respectivos processadores a serem utilizados nos sistemas embutidos. Com base na modelagem do sistema embutido, será verificado se a proposta da linguagem SysML, além de documentação do projeto, é eficaz na descrição das características técnicas do processador, auxiliando o projetista no processo de seleção do processador embutido.

7.1 Considerações para a Seleção do Processador

A seleção do processador para determinado sistema embutido é uma tarefa complexa, abrangendo os seguintes aspectos: técnico, econômico, recursos humanos e equipamentos, logística e específicos para cada sistema a ser desenvolvido. Estes aspectos envolvem análises em várias áreas de conhecimento como engenharia, computação, economia e administração.

A seguinte questão ilustra a problemática: é melhor utilizar um processador que já está sendo utilizado em outros sistemas da empresa, possibilitando a reutilização de código e com isso diminuir o tempo de projeto, porém, de maior custo, ou utilizar outro processador de menor custo mas, com todo o software por ser desenvolvido, resultando em um tempo maior de projeto e até mesmo a contratação de pessoal? Este tipo de

interrogação traz consigo aspectos difíceis de serem contemplados em uma metodologia para seleção de processadores.

Conforme o estudo realizado no capítulo 4, constatou-se que não existe uma metodologia que leve a seleção do processador ideal para a implementação do sistema embutido, apenas recomendações relacionando aspectos importantes que devem ser observados durante a seleção. Uma metodologia completa deve iniciar com a especificação e modelagem do sistema embutido, pois através da modelagem é possível identificar todas as características do sistema, e por consequência, determinar as características e requisitos do processador, inclusive a avaliação do desempenho, como ilustra a figura 60.

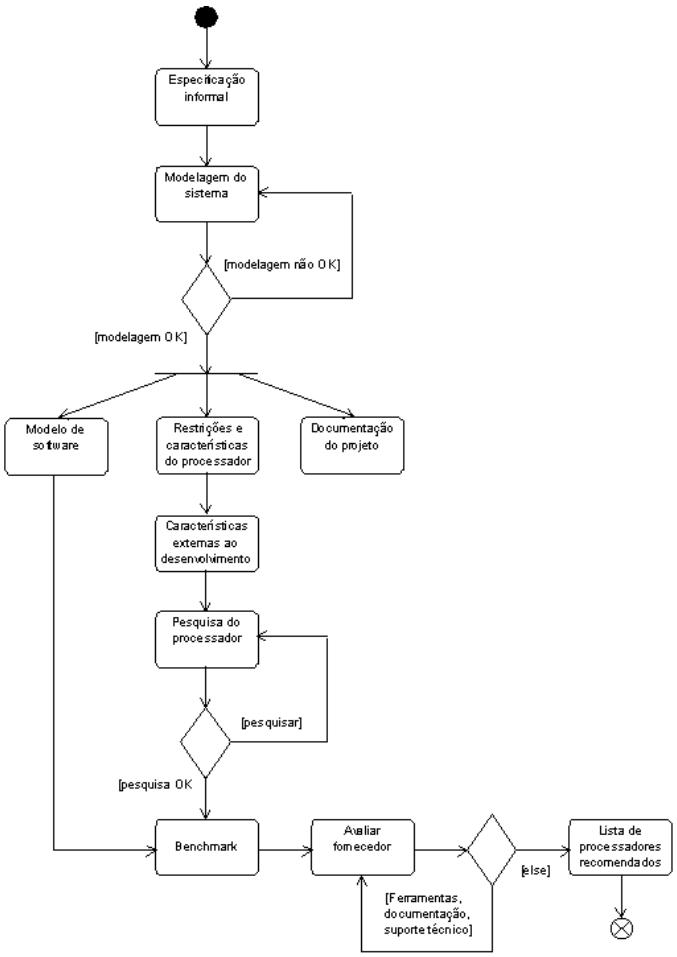


Figura 60: Etapas de uma metodologia completa para seleção de processadores

Conforme a metodologia de *hardware/software co-design*, após a modelagem do sistema embutido, a próxima etapa é o particionamento. Conforme [01], o particionamento consiste em decidir como será o mapeamento de uma descrição da funcionalidade de um sistema digital em uma arquitetura composta de componentes programáveis de propósito

geral (processadores) e componentes de hardware de aplicação específica (ASICs, FPGAs).

Entretanto, esta dissertação considera que toda a funcionalidade do sistema será implementada por software, pois o objetivo é identificar as características técnicas do componente programável de propósito geral que atenda os requisitos do sistema embutido modelado em SysML. As características técnicas do processador podem ser classificadas em físicas, elétricas, periféricos, arquitetura e memória, como pode ser verificado na tabela 2.

7.2 Uso dos Diagramas SysML

Com base na modelagem dos estudos de caso do capítulo 6, será analisado cada diagrama, com o objetivo de extrair informações referentes às características técnicas do processador embutido.

7.2.1 Diagrama de Casos de Uso

O Diagrama de Casos de Uso não apresenta informações relevantes do processador, pois é um diagrama muito abstrato, descrevendo os serviços e funções do sistema do ponto de vista do usuário, sem se preocupar sobre como serão implementados.

No Diagrama de Casos de Uso do relé de proteção existem os serviços de Medição de corrente e Medição de temperatura, que indicam que será necessário a utilização de um conversor analógico/digital como componente externo ou como periférico do processador. Entretanto, é uma informação muito abstrata, necessitando de informações de outros diagramas para definição das características.

7.2.2 Diagrama de Requisitos

O Diagrama de Requisitos também é abstrato, onde os requisitos são inseridos no modelo na forma de texto em linguagem natural, apresentando os mesmos problemas de uma especificação informal, a interpretação dúbia. Entretanto, o texto em linguagem natural pode descrever requisitos temporais da aplicação, os quais incidem diretamente nas características elétricas. Mais especificamente, na freqüência de operação, e na arquitetura, influenciando no desempenho do processador. Tais requisitos temporais podem representar

sinais de saída do processador para controle de algum atuador, ou comportamento dinâmico de alguma variável do sistema.

No Diagrama de Requisitos do sistema de injeção eletrônica são modelados dois requisitos temporais que representam sinais de controle do processador para um atuador. Porém, a descrição textual do requisito não deixa específico como o deve ser este sinal. Para modelar de forma menos abstrata tal sinal, é utilizado um estereótipo chamado <<trace>>, que faz referência a um Diagrama de Tempo que modela o sinal em um gráfico de duas dimensões.

No Diagrama de Requisitos do relé de proteção são modelados requisitos temporais e físicos para cada protocolo de comunicação, influenciando nas características do processador quanto aos periféricos, arquitetura e freqüência de operação.

7.2.3 Diagrama de Classes

O Diagrama de Classes traz um visão das partes que compõem o sistema. O relacionamento entre elas através da cardinalidade, pode dar uma idéia inicial do número de pinos do processador. Os atributos, que são as variáveis do sistema, possibilitam uma estimativa de tamanho de memória de dados e de programa.

7.2.4 Diagrama de Montagem

O Diagrama de Montagem modela as partes de hardware do sistema embutido. Este diagrama é o mais importante do ponto de vista da seleção do processador, pois é rico em informações que traduzem as características técnicas do processador.

O processador inicialmente é uma parte dentro do sistema embutido, com nível de abstração alto. Nesta etapa é possível identificar o número de pinos e suas funções, definindo as entradas e saídas, a tensão de alimentação e os sinais elétricos em cada pino.

Diminuindo a abstração, pode-se modelar as partes do processador, ou seja, modelar os periféricos que serão utilizados no sistema embutido. Os periféricos são modelados com detalhes de forma a definir suas características.

Na modelagem é utilizado o elemento nota para descrever ou explicar com mais detalhes alguma característica relevante das partes.

7.2.5 Diagrama Paramétrico

No Diagrama Paramétrico são modeladas de forma simples as funções que serão utilizadas no sistema embutido. Estas funções podem representar uma seqüência de cálculo de alguma parte de hardware ou os cálculos que deverão ser executados pelo software.

Na modelagem das funções é possível identificar e determinar o tipo das variáveis, as operações matemáticas que devem ser executadas, possibilitando estimar o tamanho do software e consequentemente o tamanho da memória de programa. Através da análise das funções pode-se estimar o tempo de resolução das mesmas, auxiliando na seleção da arquitetura do processador.

No relé de proteção são utilizados dois Diagramas Paramétricos, um para modelar a seqüência de cálculo para a unidade de medição e outro para modelar os cálculos que serão executados pelo processador, como cálculo da corrente eficaz, desbalanceamento e modelo térmico.

No sistema de injeção eletrônica, o Diagrama Paramétrico é utilizado para modelar a interpolação dos valores de aceleração e rotação, resultando no tempo de atuação do bico injetor.

7.2.6 Diagrama de Atividades

O Diagrama de Atividades modela o software a ser executado pelo processador. Através da análise deste modelo e a experiência do projetista, pode-se estimar a quantidade de instruções do programa, determinando seu tamanho e consequentemente o tamanho da memória de programa.

Em análise conjunta com os Diagrama Paramétrico e Diagrama de Requisitos, é possível identificar tarefas que precisam ser executadas dentro de um determinado tempo para não comprometer o funcionamento do sistema, determinando os tempos críticos do sistema. Esses tempos críticos influenciam na seleção da arquitetura do processador.

7.2.7 Diagrama de Tempo

O Diagrama de Tempo possibilita exemplificar facilmente, através de gráficos, determinados requisitos descritos de forma textual no Diagrama de Requisitos. Tais gráficos podem auxiliar na implementação do software e na identificação de tempos críticos de execução.

No sistema de injeção eletrônica foi utilizado dois Diagramas de Tempo para modelar o sinal de controle sobre os atuadores válvula "borboleta" e bico injetor. No relé de proteção foi utilizado um Diagrama de Tempo para modelar o funcionamento do modelo térmico do motor elétrico para vários níveis de sobrecarga do motor.

7.2.8 Compilação das Informações dos Diagramas SysML

A análise realizada em cada diagrama utilizado na modelagem dos estudos de caso, resultou em um guia na forma de uma tabela, com o objetivo de orientar a extração das informações dos diagramas SysML. Este guia, proposto na tabela 2, relaciona as características técnicas do processador que podem ser observadas em cada diagrama da linguagem SysML.

Características técnicas		**Diagramas**						
		Casos de Uso	Requisitos	Classes	Tempo	Montagem	Paramétrico	Atividades
Físicas	Número de pinos			**X**		**X**		
	Número de entradas e saídas					**X**		
	Temperatura de operação		**X**			**X**		
Elétricas	Tensão de alimentação					**X**		
	Consumo de corrente		**X**			**X**		
	Frequência de operação		**X**		**X**			**X**
	Corrente máxima dos pinos de saída					**X**		
Periféricos		**X**	**X**	**X**		**X**		
Memória			**X**	**X**			**X**	**X**
Arquitetura			**X**		**X**		**X**	**X**

Tabela 2: Diagramas da SysML e as características técnicas do processador

7.3 Requisitos e Características do Processador Extraídos da Montagem dos Estudos de Caso

As características técnicas do processador embutido para cada estudo de caso modelado no capítulo 6, foram extraídas dos diagramas SysML, conforme propõe a tabela 2. As características extraídas dos diagramas são compiladas na tabela 3, seguindo as seguintes etapas:

- Identificação das características físicas e elétricas;
- Periféricos;
- Identificação da arquitetura do processador;
- Estimativa do tamanho da memória de programa e de dados;

Características técnicas		**Injeção eletrônica**	**Relé de proteção**
Físicas	Número de pinos	Mínimo 13	Mínimo 50
	Número de entradas e saídas	5 saídas 6 entradas	23 Saídas 17 Entradas
	Temperatura de operação	Requisito não especificado	Temperatura industrial -20 a 80C
Elétricas	Tensão de alimentação	5 Vcc	5Vcc
	Consumo de corrente	Sistema alimentado por bateria, mas sem restrições de consumo	Sistema sem restrições de consumo
	Freqüência de operação da UCP	5MHz	12MHz
	Corrente máxima dos pinos de saída	Sinal de saída sem necessidade de corrente	Sinal de saída sem necessidade de corrente
Periféricos		Conversor A/D 10 bits com 5 canais Módulo PWM 12 bits Módulo Serial USART	Conversor A/D 10 bits com 6 canais 2 Módulo Serial USART Módulo CAN DMA
Arquitetura		8 bits	16 bits
Memória	Dados	Memória de Programa/20	Memória de Programa/20
	Programa	Maior capacidade da família	Maior capacidade da família

Tabela 3: Características técnicas do processador para cada caso de uso

7.3.1 Características Físicas e Elétricas

As características físicas e elétricas utilizam basicamente três diagramas: Diagrama de Requisitos, Diagrama de Tempo e Diagrama de Montagem, conforme a tabela 2. Os requisitos número de pinos, número de entradas e saídas, tensão de alimentação, consumo de corrente e corrente máxima de saída podem ser extraídos diretamente do Diagrama de Montagem. Os requisitos de consumo e temperatura de operação podem ser modelados como uma nota no Diagrama de Montagem ou descritos no Diagrama de Requisitos.

O requisito freqüência de operação é um requisito que requer uma análise das informações dos Diagramas de Requisitos, Tempo e Atividades.

O sistema de injeção eletrônica possui uma restrição de tempo modelada no Diagrama de Requisito de 5us para tratamento da interrupção do sensor de rotação. Considerando que são necessárias 25 instruções da UCP para execução do tratamento da interrupção, se dividirmos 5us por 25 instruções, encontraremos 200ns, que resulta em uma freqüência de *clock* de 5MHz. Quanto mais instruções forem necessárias para o tratamento, maior a freqüência de *clock* do processador.

Na modelagem do sistema de injeção eletrônica, a rotina de interrupção do sensor de rotação não foi modelada, impossibilitando a estimativa da quantidade de instruções e consequentemente a definição da freqüência mínima de *clock*.

O relé de proteção apresenta restrições temporais referentes aos protocolos de comunicação. O protocolo Profibus-DP pode operar com taxa de transferência de no máximo 12Mbps, representando um bit a cada 80ns. Considerando que um pacote Profibus-DP possui 8 bytes, a cada 5,12us a pilha de protocolo deve ser executada. A freqüência de *clock* do processador depende da quantidade de instruções utilizada para implementação da pilha do protocolo Profibus-DP. Considerando que a pilha do protocolo utilize 1500 instruções, dividindo 5,12us por 1500 instruções, encontramos 3,4ns, resultando em uma freqüência de *clock* de 290MHz. A primeira conclusão é que o protocolo Profibus-DP não pode ser implementando no processador, pois sua restrição temporal exige *clock* de aproximadamente 300MHz e exige um processador com desempenho altíssimo e portanto muito caro.

O protocolo Devicenet aparece como segunda pior restrição temporal, trabalhando com taxa de transferência de no máximo 500Kbps, representando um bit a cada 2us. Considerando que um pacote Devicenet também possui 8 bytes, a cada 12,8ms a pilha de protocolo deve ser executada. Considerando que a pilha do protocolo Devicenet utiliza 1500 instruções, dividindo 12,8ms por 1500 instruções, encontramos 85ns, resultando em uma freqüência de *clock* de 12MHz.

Na modelagem do relé de proteção existe apenas informações referentes à camada física, a taxa de transferência e a identificação da norma para cada protocolo de comunicação, o que não é suficiente para determinar a freqüência de *clock* do processador com mais precisão.

A freqüência de *clock* do processador em determinadas situações pode ser definida de forma empírica, durante o processo de desenvolvimento.

7.3.2 Periféricos

Os periféricos exigidos pelo sistema embutido podem ser capturados no Diagrama de Casos de Uso, Requisitos e Classes, mas no Diagrama de Montagem do processador são modelados com detalhes. Para os dois estudos de caso desenvolveu-se um Diagrama de Montagem específico para as partes do processador, detalhando as características de cada periférico. Em determinados sistemas embutidos, um requisito pode exigir um periférico especial que pode restringir a seleção do processador embutido.

Os periféricos podem ser os mais variados como conversor analógico/digital, conversor digital/analógico, módulo PWM, temporizador, *watchdog*, comparador, módulo serial, módulo CAN, módulo ethernet, módulo I2C, DMA, memória EEPROM, oscilador interno, etc.

7.3.3 Arquitetura do Processador

A arquitetura do processador define o tamanho da palavra e o barramento de dados do mesmo, podendo ser de 4, 8, 16 ou 32 bits. Esta característica está diretamente relacionada com o desempenho e o custo do processador. Quanto maior o número de bits, maior o desempenho do processador, possibilitando a execução de instruções mais complexas e o processamento de informações (dados) de maior tamanho em menos tempo.

A arquitetura do processador é uma característica que pode ser definida analisando os Diagramas de Requisitos, Tempo, Paramétrico e Atividades. Os Diagramas de Requisitos e Tempo fornecem informações sobre restrição de tempo para execução de uma tarefa. Com base neste tempo deve-se analisar o Diagrama Paramétrico, verificando a complexidade dos cálculos e os tipos de variáveis envolvidas (*char*, *int*, *float*, etc). Por fim, deve-se analisar o Diagrama de Atividades para estimar o número de instruções para execução da tarefa.

Com base nas orientações descritas, pode-se fazer a seguintes ponderações:

1- Se a restrição de tempo é pesada, na casa dos microsegundos, mas os cálculos são simples com variáveis do tipo inteiro, os processadores de 8 bits podem ser a alternativa mais econômica com desempenho suficiente para atender a aplicação.

2- Se a restrição de tempo é pesada, na casa dos microsegundos, os cálculos utilizam muitas operações de multiplicação ou divisão, com variáveis do tipo inteiro e ponto flutuante, os processadores de 8 bits, mesmo operando na sua freqüência máxima de *clock*, podem não atender ao requisito temporal. Neste caso processadores de 16 ou 32 bits, incluindo DSPs e DSCs, apresentam desempenho suficiente para atender ao requisito temporal da aplicação.

Analisando o Diagrama de Requisitos do sistema de injeção eletrônica, pode-se observar uma tarefa com restrição temporal de 5us. Entretanto, analisado o Diagrama Paramétrico, observa-se que os cálculos são simples, necessitando apenas encontrar um valor de tempo de atuação para o bico injetor em uma tabela na memória. Considerando as informações acima pode-se concluir que a opção de menor custo seria um microcontrolador de 8 bits.

O relé de proteção apresenta no Diagrama de Requisitos, restrições temporais para os protocolos de comunicação na ordem de 83,3ns (12Mbps), 2us (500Kbps) e 8,7us (115,2Kbps) por bit para os protocolos Profibus-DP, Devicenet e Modbus-RTU respectivamente. Porém, conforme análise na seção 1.3.1, a restrição temporal que deve ser considerada é a do protocolo Devicenet. Conforme o Diagrama Paramétrico, os cálculos apresentam muitas multiplicações e divisões com variáveis de 16 e 32 bits. Diante deste

contexto, conclui-se que um microcontrolador de 16 bits é suficiente para atender os requisitos da aplicação.

A seleção da arquitetura do processador embutido é um item que exige conhecimento e experiência do projetista.

7.3.4 Memória

As memórias podem ser classificadas em memória de programa e memória de dados. A estimativa da quantidade de memória que será alocada para armazenar o programa é uma tarefa complexa, onde a experiência do projetista, mais o projeto de software e código exemplo, auxiliam neste processo.

Uma tentativa de estimar o tamanho da memória de programa é fazer uma análise dos Diagramas de Classes, Atividades e Paramétrico. Através dos Diagramas de Classes e Paramétrico, pode-se determinar as variáveis e as operações matemáticas que serão utilizadas no software. Como o Diagrama de Atividades é o projeto de software, o projetista baseia-se nele e em sua experiência para estimar o número de instruções para realização de determinada rotina.

Entretanto, o número de instruções e o tamanho do programa podem variar conforme a linguagem em que o programa será escrito e o compilador utilizado.

Na prática, devido a dificuldade de estimar o tamanho da memória de programa, seleciona-se uma família de processadores que atenda os requisitos do sistema, e seleciona-se o dispositivo com maior capacidade de memória, deixando a otimização da memória para a etapa do desenvolvimento do software, onde o tamanho do mesmo é encontrado de forma precisa.

A memória de dados pode ser definida utilizando a regra sugerida por [22], que define que o tamanho da memória de dados é de 12 a 20 vezes menor que o tamanho da memória de programa.

7.4 Conclusão

A linguagem SysML possibilita uma modelagem do sistema embutido a nível de possibilitar a identificação de algumas características técnicas do processador embutido.

As características físicas, elétricas e periféricos são facilmente modeladas e extraídas dos diagramas, em especial do Diagrama de Montagem.

A arquitetura do processador, tamanho de memória e freqüência de operação, são características complexas demais para serem extraídas da modelagem SysML.

A freqüência de operação e tamanho da memória são características que dependem do número de instruções que o programa todo ou apenas uma rotina irá executar. Quem estima o número de instruções é o projetista, baseado no projeto de software, ou seja, no Diagrama de Atividades. Um dos problemas é que o Diagrama de Atividades é bastante abstrato, do ponto de vista de linha de código. Outro agravante é que o número de instruções e consequentemente o tamanho do programa variam conforme o compilador utilizado. Para definir tais características técnicas do processador, a modelagem SysML auxilia mas é insuficiente, dependendo muito da experiência do projetista.

A arquitetura do processador é uma característica complexa de ser definida. Basicamente deve-se definir o menor tempo de execução de algum processamento e definir os cálculos que serão utilizados. A solução ideal seria a realização de um benchmarking utilizando a rotina crítica como carga, comparando microcontroladores de 8 e 16 bits, com DSPs e DCSs, verificando qual seria a solução mais econômica e com desempenho suficiente para a aplicação. Entretanto, tal solução demandaria muito trabalho, tempo e ferramentas de hardware e software.

Utilizando a modelagem SysML, pode-se identificar algumas restrições de tempo através do Diagrama de Requisitos ou pelo estereótipo <<rate>> no Diagrama de Atividade e os cálculos podem ser analisados no Diagrama Paramétrico. Entretanto, não é possível extrair da modelagem o desempenho ou a arquitetura para o processador. A modelagem SysML pode auxiliar o projetista estimar o desempenho, mas é a experiência, um fator empírico, que vai definir a arquitetura do processador, ou no decorrer do desenvolvimento.

A modelagem SysML não permite extrair com consistência características importantes do processador embutido como arquitetura, tamanho da memória de programa e freqüência de operação.

Capítulo 8

Conclusão

O objetivo desta dissertação é avaliar através de dois estudos de caso a linguagem SysML com respeito a modelagem de sistemas embutidos. Em especial, avaliar a utilidade desta modelagem na seleção do processador. Este estudo limita-se a identificar as características técnicas, englobando requisitos elétricos, físicos, periféricos, memória e arquitetura. Os aspectos econômicos, estratégicos e específicos à cada sistema não serão abordados nesta dissertação.

A dissertação pode ser dividida em duas partes distintas. Na primeira foi feito um estudo dos sistemas embutidos, abrangendo seu projeto através da metodologia de *hardware/software co-design*, um estudos dos componentes que podem ser utilizados como elemento de processamento em sistemas embutidos. Também foi realizado uma pesquisa na literatura das metodologias para seleção do processador embutido, identificando o que deve ser observado no processo de seleção.

A segunda parte trata de um estudo sobre a especificação e modelagem de sistemas embutidos, focando a linguagem de modelagem SysML. Atualmente, devido a falta de uma ferramenta de engenharia de *software* embutido, é comum no projeto do *software* embutido, o emprego de diagramas com formato livre e diagramas de blocos que lembram a UML (*Unified Modeling Language*), diagrama de fluxo de dados (DFD) e fluxogramas. É utilizado uma ferramenta de desenho de propósito geral para fazer os diagramas do projeto de *software*. A falta de sintaxe e semântica padronizada proporciona a interpretação errada dos diagramas. Essa má interpretação é mais evidente quando o projeto é

interdisciplinar, onde os membros possuem formação distinta.

Diante desta lacuna, a OMG (*Object Management Group* ou Grupo de Gerenciamento de Objetos) propôs a criação de uma linguagem específica para atender os requisitos da engenharia de sistemas. A proposta desta nova linguagem de modelagem, denominada SysML (*System Modelling Language* ou Linguagem de Modelagem de Sistema), é baseada na UML 2.0 e possibilita em uma única especificação uma visão abrangete do sistema, incluindo *hardware, software* e partes eletro-mecânicas.

Como a equipe de desenvolvimento de um sistema embutido é interdiciplinar, envolvendo profissionais de áreas distintas, a SysML é uma proposta muito interessante para melhorar a comunicação entre as áreas afim. Um exemplo para ilustrar tal afirmação, seria uma reunião sobre determinado sistema embutido, onde um engenheiro eletrônico traz os diagramas eletrônicos, um engenheiro de controle traz os diagramas de controle e um programador traz os diagramas de software. A interação entre a equipe é mínima, pois um não tem conhecimento da linguagem de modelagem do diagrama do outro. Entretanto, se a SysML fosse utilizada, a linguagem de modelagem seria a mesma, proporcionando a máxima interação entre a equipe.

Foi realizada uma avaliação prática da linguagem de modelagem SysML, com o objetivo de identificar suas capacidades e suas limitações na modelagem de sistemas embutidos. Para tanto, foram utilizados dois sistemas como estudos de caso, um sistema de injeção eletrônica desenvolvido em laboratório do Departamento de Automação e Sistemas da UFSC e um relé de proteção de motores elétricos hipotético. Tal modelagem em diagramas engloba os requisitos mecânicos, elétricos, hardware, software, normas técnicas e funções matemáticas. Na modelagem dos estudos de caso com a linguagem SysML, se fez necessário a utilização de material de apoio sobre UML, e modelagem orientada a objeto, comprovando que SysML é fortemente baseada na UML 2.

Verificou-se que os diagramas da linguagem SysML fornecem subsídio ao desenvolvimento tanto da parte de hardware como a de software, apresentando uma descrição completa do sistema. A partir desta modelagem através de diagramas, pode-se destacar alguns aspectos positivos:

- Diminuir o espaço de soluções do projetista através das restrições e características já definidas nos diagramas SysML;

- Tradução da modelagem SysML para linguagens formais específicas, como determinada linguagem de programação para o software e VHDL para o hardware;

- Capacidade de diminuir a distância semântica entre as equipes envolvidas no desenvolvimento do sistema embutido, pois apresenta um modelo com sintaxe padronizada;

- Os diagramas podem ser utilizados como documentação do sistema embutido, requisito que geralmente é precário na prática.

Através da modelagem dos estudos de caso, pode-se destacar também alguns aspectos negativos da SysML:

- A modelagem dos requisitos temporais devem ser aprimorados, pois são modelados de forma textual no Diagrama de Requisitos, possibilitando interpretações incorretas;

- Necessidade de conhecimento de UML e orientação a objeto.

Com base na metodologia de hardware e software co-design, a SysML é posicionada na etapa de especificação do sistema, fornecendo subsídios para a modelagem formal das partes de hardware e software, a qual será utilizada na etapa de particionamento.

Após o estudo e a modelagem dos dois sistemas utilizando a linguagem SysML, o passo seguinte foi o estudo dos modelos para a extração de informações para a seleção do processador. Como a seleção do processador para determinado sistema embutido é uma tarefa complexa, abrangendo aspectos técnicos, econômicos, recursos humanos e equipamentos, logística e específicos para cada sistema a ser desenvolvido, a extração de informações dos diagramas limitou-se aos aspectos técnicos.

Neste processo de compilação de características, observou-se que a modelagem SysML do sistema embutido possibilitou a identificação de algumas características técnicas do processador.

As características físicas, elétricas e periféricos foram facilmente modeladas e extraídas dos diagramas, em especial do Diagrama de Montagem. Entretanto, a arquitetura do processador, tamanho de memória e freqüência de operação, são características complexas demais para serem extraídas da modelagem SysML. Tais características dependem mais da experiência do projetista do que dos diagramas da SysML.

Do ponto de vista da seleção do processador embutido, os diagramas da SysML não foram suficientes para possibilitar a extração e definição de algumas características complexas do processador.

Um dos problemas enfrentados neste trabalho foi a busca por material de apoio para o estudo da linguagem de modelagem SysML. Além da especificação da linguagem em versão rascunho, foram encontrados apenas alguns relatórios técnicos e *white papers* disponibilizados por empresas que estão trabalhando na sua especificação.

Como toda linguagem, a SysML também exige tempo para que o projetista ganhe experiência e domínio na modelagem de sistemas. É possível que os estudos de caso pudessem ser modelados apresentando diagramas com mais detalhes, utilizando elementos avançados da linguagem, ou até mesmo mais diagramas como os Diagramas de Máquina de Estados e de Seqüência.

A SysML é uma aposta de um consórcio de empresas, gerenciadas pela OMG, que buscam um padrão para projeto de sistemas embutidos. A academia não participa como co-autora desta especificação. Esta ausência demonstra a escassez de referências bibliográficas sobre SysML, o que dificultou seu estudo. Entretanto, acredita-se que após a definição de uma especificação final, a academia comece a avaliar tal proposta de modo a gerar experimentos que demonstrem e comprovem sua potencialidade. Adiantando-se a essa tendência, esta dissertação gerou artigos sobre o uso da SysML que foram submetidos ao *International Conference on Systems, Man, and Cybernetics* (SMC) e ao Congresso Brasileiro de Automática (CBA).

Também foi gerado um artigo sobre a primeira parte da dissertação e submetido a Conferência Internacional de Aplicações Industriais (INDUSCON 2006). O artigo submetido a INDUSCON [44] foi aprovado e apresentado.

Referências Bibliográficas

[01] VALDERRAMA, C. A.; LIMA, M. E.; CAVALCANTE, S. et al. *Hardware/Software co-design: projetando hardware e software concorrentemente*. Escola de computação, São Paulo: IME-USP, 2000.

[02] CARRO, L.; WGNER, F. R. Sistemas computacionais embarcados. *JAI'2003 - Jornada de Atualização em Informática*, UNICAMP, 2003.

[03] HENKEL, J.; HU, X. S.; BHATTACARYYA, S. S. Taking on the embedded system design challenge. *IEEE Computer*, v. 36, n. 4, p. 35-37, Abr. 2003.

[04] SILVA, D. C.; COELHO, C. J. N.; FERNANDES, A. O. Desenvolvimento de Sistemas Integrados de Hardware e Software. In: VI ESCOLA DE INFORMÁTICA DA SBC – REGIONAL SUL (Curitiba). *Anais*. Curitiba, 1998. p. 160-170.

[05] WOLF, W.; FREY, E. Tutorial on embedded system design. *Computer Design: VLSI in Computers and Processors*, Cambridge, p.18-21, Out. 1992.

[06] WILMSHURST, T. *Introduction of the Desing of Small-Scale Embedded System.* 1 ed.: Paperback, 2001.

[07] TANENBAUM, A. S. *Organização Estruturada de Computadores.* 4 ed. Rio de Janeiro: LTC editora, 2001.

[08] CUNHA, A. F. Comparativo de microcontroladores. *Revista Saber Eletrônica*, n. 360, p.42-48, Jan. 2003.

[09] BANNATYNE, R.; VIOT, G. Introduction to microcontrollers. *WESCON/97 - Conference Proceedings*, p. 564-574, Nov. 1997.

[10] ARNOLD, K. *Embedded Controller Hardware Desing.* LLH Technology Publishing, 2000.

[11] KILIAN, C. T. *Modern Control Technology: Components and Systems.* 2 ed.: Thomson Delmar Learning, 2000.

[12] BERGER, A. S. *Embeded System Desing – An introdustion to Processs Tolls, & Techniques.* Kansas: CMP Books, 2002.

[13] MARVEN, C.; EWERS, G. *A Simple Approach to Digital Signal Processing.* New York: Wiley-Interscience Publication, 1996.

[14] FOWLER, K. DSP Selection for Real-Time Systems. *IEEE Instrumentation & Measurement Magazine*, v. 5, n. 4, p. 42-45, Dez. 2002.

[15] BANNATYNE, R. Hybrid digital signal controller enables next-generation automotive control systems. *Embedded Computing Design*, Nov. 2003.

[16] DUECK, R. *Digital Design with CPLD Applications and VHDL*. 1 ed.: Thomson Delmar Learning, 2000.

[17] XILINX (2005). Lógica programável. http://www.xilinx.com/company/about/programmable.

[18] BROWN, S.; ROSE, J. FPGA and CPLD architectures: a tutorial. *Design & Test of Computers, IEEE,* Toronto, v. 13, n. 2, p. 42-57, 1996.

[19] SOUZA, M. A. (2004). FPGA e suas características. http://www.cefetrn.br/~braulio/G71005-OrgComp/2004.2/avaliacao/fpga_17101.doc

[20] BESERRA, A. F. L. (2004). Field Programmable Gate Array. http://www.cefetrn.br/~braulio/G71005-OrgComp/2004.2/avaliacao/fpga_17102.doc

[21] ALMEIDA, A. R. (2005). Introdução aos Dispositivos Lógicos Programáveis. http://www2.ele.ufes.br/~ailson/digital2/PLD_Intro.pdf.

[22] VACLICA, J.J; GILMOUR, P. S. How To Select a Microcontroller. *Spectrum IEEE*, v. 27, n. 11, Nov. 1990.

[23] WOLF, W. How many system architectures? *Computer IEEE*, v. 36, n. 3, p. 93-95, Mar. 2003.

[24] CONTE, T. M. Choosing The Brain(s) of an embedded system. *Computer IEEE*, v. 35, n. 7, p. 106-107, Jul. 2002.

[25] GUPTA, T.V.K.; SHARMA, P.; BALAKRISHNAN, M.; MALIK, S. Processor Evaluation in an Embedded Systems Design Environment. *VLSI Design 2000*. India, p. 98-103, Jan. 2000.

[26] PATTERSON, D. A.; HENNESSY, J. L. *Organização e Projeto de Computadores: a interface hardware / software.* 2 ed. Rio de Janeiro: LTC Editora, 1998.

[27] JAIN, R. *The Art of Computer Systems Performance Analysis. Techniques for Experimental Design, Measurement, Simulation and Modeling.* New York: John Wiley & Sons Inc, 1994.

[28] GILMOUR, P. S. How To Select tools for microcontroller software. *Spectrum IEEE*, v. 28, n. 2, p. 37-39, Fev. 1991.

[29] PEREIRA, A. M. R.; MASCENA, D. A. A.; GUTIERRE, L. R. Hardware Description Language: SystemC. *CEFET*, Natal, 2002.

[30] EDWARDS, S.; LAVAGNO, L.; LEE, E. Design of embedded systems: formal models, validation and synthesis. *Proceedings of the IEEE*, v. 85, n. 3, p. 366-390, Mar. 1997.

[31] GRAAF, B.; LORMANS, M.; TOETENEL, H. Embedded software engineering: the state of the practice. *IEEE Computer Society*, v. 20, n. 6, p. 61-69, Dez. 2003.

[32] Systems Modeling Language (SysML) Specification. Version 0.9, Janeiro. 2005.

[33] GRADY, B.; RUMBAUGH, J.; JACOBSON, I. *UML, Guia do Usuário.* Tradução Fabio Freitas da Silva, Rio de Janeiro: Campus, 2000.

[34] HAUSE, M.; THOM, F.; MOORE, A. The Systems Modelling Language (SysML). *IEEE Systems and Software*, Jun. 2005.

[35] BOCK, C. Systems engineering in the product lifecycle. *US National Institute of Standards and Technology*, USA, 2005.

[36] GUEDES, G. T. A. *UML 2 - Guia de Consulta Rápida*. São Paulo: NovaTec Editora, 2005.

[37] BOSCH. *Sistemas de Injeção Eletrônica.* São Paulo: Jun. 2001.

[38] FILHO, G. F. *Motor de indução.* São Paulo: Érica, 2000.

[39] CAPELLI, A. Motores elétricos CA. *Saber Eletrônica*, São Paulo, n. 338, p. 4-9, Mar. 2001.

[40] ROCKWELL AUTOMATION. Conceito de CCM Inteligente. *Automation Today*, São Paulo, Out. 1999.

[41] INTERNACIONAL ELECTROTECHNICAL COMMISSION. *Contactors and motor-starters – electromechanical contactors and motor-starters,* IEC 60947-4-1. Geneva, 2000.

[42] UNDERWRITERS LABORATORIES INC. *Industrial Control Equipment,* UL508. Northbrook, 1999.

[43] NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATION. *Motor and Generators,* NEMA MG 1. Rosslyn: 2004.

[44] SILVA, A. J. da; OLIVEIRA, R. S. de. Methods and Guidelines for Embedded System Processor Selection. *Recife: VII Induscon*, Abr. 2006.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo