Universidade Federal de Pernambuco

Centro de Informática

Pós-graduação em Ciência da Computação

ESTIMATIVA DE CONSUMO DE ENERGIA DE CÓDIGO ANSI-C PARA SISTEMAS EMBARCADOS: UMA ABORDAGEM BASEADA EM SIMULAÇÃO ESTOCÁSTICA

Angelo Roncalli de Novaes Pires Ribeiro

DISSERTAÇÃO DE MESTRADO

Recife

10 de setembro de 2007

Universidade Federal de Pernambuco
Centro de Informática

Angelo Roncalli de Novaes Pires Ribeiro

ESTIMATIVA DE CONSUMO DE ENERGIA DE CÓDIGO ANSI-C PARA SISTEMAS EMBARCADOS: UMA ABORDAGEM BASEADA EM SIMULAÇÃO ESTOCÁSTICA

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Prof. Dr. Paulo Romero Martins Maciel*

Recife
10 de setembro de 2007

Dedico a minha esposa, Hévila, e a minha amada família.

AGRADECIMENTOS

Gostaria de registrar meus agradecimentos como um reconhecimento sincero da dedicação das muitas pessoas que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Ao Professor e amigo Paulo Maciel, pela orientação exemplar, pelos conselhos, acompanhamento em todas as fases desse trabalho e, principalmente, pela amizade construída.

Aos Professores Nelson Rosa e Meuse, pela disponibilidade em participarem da banca examinadora, contribuindo com seus comentários para melhorar a qualidade deste trabalho.

Aos colegas do grupo de alunos e amigos do Professor Paulo, em especial ao Professor Ricardo Massa, Raimundo e mais uma vez Meuse. Agradeço bastante o companheirismo de Eduardo Tavares, Gabriel, Adilson, Renato Bibiano, Cezar, Ana Paula, Tiago Barros (Tiaguinho), Euclides e a todos os verdadeiros amigos que conquistei durante todo o mestrado. Uma lembrança especial aos novos alunos do professor Paulo, dentre eles Érica e Julian, que acompanharam os meus passos finais e que estão super empolgados com as suas pesquisas.

Ao amigo Fred Monteiro, pela parceria, apoio e amizade que não me deixou fraquejar diante das dificuldades em um novo país e novos grandes desafios. Me fez sempre olhar para frente com otimismo e perseverança.

Agradeço à PROPESQ, que permitiu à defesa dessa dissertação com os recursos disponíveis, fazendo que a mesma fosse realizada com sucesso.

A todos os amigos que não conseguirei enumerar neste agradecimento mas que sabem que tem o seu lugar especial guardado em meu coração; padrinhos, madrinhas, amigos do CESAR, empresa que me incentivou a ser MESTRE, Samsung, dentre outras. E aos novos amigos de Redmond, que estão fazendo da minha esperiência nos Estados Unidos bastante aconchegante.

Por último, mas não menos importante, à minha família pelo apoio, compreensão, amor e dedicação. Principalmente à minha mãe, Selma Novaes, que muitas vezes levei ao

desespero nos momentos mais difíceis. Ao meu amado pai, Darlan Ribeiro, cujo exemplo de homem me faz ser quem eu sou. À minha irmã pelo apoio amigo e inseparável. À Dadinha, segunda mãe que estará sempre ao meu lado com o seu carinho eterno. Aos meus sogros Jorge Freitas e Sônia Cristina, com a sua torcida constante e amizade inigualável. E à minha amada esposa, fonte de minha inspiração em todos os novos desafios e objetivos da minha vida. AMO TODOS VOCÊS!

RESUMO

Sistema Embarcado é um sistema computacional projetado para uma função dedicada. Geralmente, este sistema executa uma tarefa específica dentre um conjunto maior de tarefas, e possui particularidades tanto de *hardware* quanto de *software*.

Os Sistemas Embarcados estão presentes no cotidiano sob diferentes formas e com diferentes objetivos. Geralmente possuem uma série de restrições, tais como: dimensões das memórias, fonte de energia, baixa velocidade de processamento, dentre outras.

Este trabalho apresenta o desenvolvimento de um modelo em Redes de Petri de desempenho e energias para códigos ANSI-C, considerando um processador de uma plataforma embarcada específica, com o objetivo de estimar o consumo de energia.

A linguagem ANSI-C, como código de Sistemas Embarcados, foi escolhida por ser uma das mais utilizadas no desenvolvimento destes sistemas. Redes de Petri Temporizadas permitem a modelagem e especificação de sistemas paralelos e distribuídos e, ao mesmo tempo, provêem o formalismo matemático necessário para uma avaliação de desempenho. Neste trabalho, o modelo Redes de Petri Temporizada é anotado com informação de consumo de energia, o que originou a Power Petri Net.

Este trabalho contribui também com a implementação de um simulador estocástico para avaliação de desempenho e de um ambiente computacional no qual são realizadas as estimativas. Esse ambiente é formado pelo tradutor de código ANSI-C para Redes de Petri no formato PNML [1], padrão XML para descrição de Redes de Petri, simulador estocástico e extensão do ambiente EzPetri [2]. Como o modelo apresentado, básico, pode ser estendido em outros trabalhos é caracterizada a formação de um *framework*.

Para validação do método proposto, foi utilizado um código de avaliação, *benchmark*, PowerStone [3] desenvolvido para explorar o sistema sob diferentes aspectos de consumo de energia.

Palavras-chave: Redes de Petri, estimativa de consumo de energia, metodologia de modelagem, métodos formais, simulação de Redes de Petri Temporizadas e PNML.

ABSTRACT

Embedded Systems are present in most daily activities with many different purposes. Generally, embedded systems have several design constraints, such as memory size, power source, CPU speed, and so on. Over the last years, power consumption has been receiving particular attention from scientific community since several embedded devices rely on battery as power sources. Therefore, several works on power consumption estimation are available and each one applies a different method in order to tackle this specific problem domain.

This work presents an approach based on Timed Petri Net for estimating power consumption of ANSI-C description, considering the target processor architecture.

ANSI-C language description was chosen since it is the most used language in embedded systems development. On other hands, Timed Petri Net allows modeling parallel and distributed system, using a high-level abstraction of the specification and providing a mathematical formalisms that allows a qualitative analysis as well as a performance and power consumption estimation. The Timed Petri net model adopted is labeled with power consumption information and has been defined as Power Petri Net.

For validating the proposed method, PowerStone *Benchmark* has been adopted [3]. PowerStone is a *benchmark* specifically developed for evaluating power consumption exploring different aspects of the system.

The methodology proposed adopts stochastic Simulation of the Timed Petri Net model in order to execute the performance evaluation and power consumption estimation. The proposed *framework* also consists of a *parser* from an ANSI-C code description to PNML [1], which is a XML pattern of Petri net descriptions.

Keywords: Timed Petri Net, Power Consumption Estimation, Stochastic Simulation, Analysis, Modelling, PowerStone, Performance Analysis

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Sistema Embarcado é um sistema computacional projetado para uma função dedicada. Geralmente, este sistema executa uma tarefa específica dentre um conjunto maior de tarefas, e possui particularidades tanto de *hardware* quanto de *software*.

Este tipo de sistema pode ser encontrado em sistemas aviônicos, sistemas de telemetria, controladores de vôo e outros sistemas de controle de aviões e mísseis; telefonia celular; sistemas de controle de motores; sistemas de anti-travamento em freios; automação de casas inteligentes; calculadoras de mão; video-games; periféricos de computadores como impressoras, mouse, teclados; instrumentos musicais eletrônicos como sintetizadores digitais e pianos digitais.

Os Sistemas Embarcados estão presentes no cotidiano sob diferentes formas e com diferentes objetivos. Geralmente, possuem uma série de restrições, tais como: as dimensões das memórias, fonte de energia, baixa velocidade de processamento, dentre outras.

O barateamento dos sistemas embarcados na década de 80 do século XX resultou no aumento de novas concepções tecnológicas durante a década de 90. Paralelamente, o consumo de energia passou a despertar uma atenção especial da comunidade científica. Essa importância derivou-se de uma maior exigência em relação a sua autonomia, ou seja, o tempo necessário para a troca ou recarga da fonte de energia.

Outro requisito importante é a dimensão. O mercado consumidor tem se tornado cada vez mais exigente em relação ao tamanho destes sistemas, o que impacta diretamente na autonomia, pois, quanto menor a fonte de energia, melhor tem que ser a tecnologia utilizada para garantir uma boa autonomia.

Um grande número de trabalhos pode ser encontrado usando diferentes abordagens com o objetivo de resolver o problema de consumo. Estas abordagens vão desde o estudo de técnicas de compilação, que visam a diminuir o código de máquina para reduzir o

consumo, até mudanças na forma de conceber o projeto, evitando que tarefas que não estão sendo efetuadas interfiram no consumo de energia.

Uma das linguagens de programação mais utilizadas para programação de sistemas embarcados é a ANSI-C. C [4] é uma linguagem de propósito geral que tem como propriedade uma economia na quantidade de expressões (aritmética, lógica, dentre outras), controle de fluxo de programa e estruturas de dados e um variado conjunto de operadores. C não é uma linguagem de alto nem de baixo nível, o que faz com que seja bem aceita entre todos os tipos de programadores.

Como cada vez mais, devido às demandas atuais, busca-se a redução das dimensões dos sistemas embarcados e mais se exige em termos de autonomia, a estimativa de consumo de energia se tornou, então, fator primordial no projeto desses sistemas.

Com o objetivo de reduzir as dimensões e, ao mesmo tempo, aumentar a capacidade de processamento, a tecnologia de semicondutores vem diminuindo cada vez mais a distância entre os seus componentes básicos, causando um aumento na densidade de potência no chip, o que acarreta desafios na construção de encapsulamentos. Estes devem ter menor resistência térmica específica, ou seja, por área de silício.

Durante o ciclo de projeto de Sistemas Embarcados, um dos pontos de maior risco é o projeto do *hardware*. Erros no dimensionamento do *hardware* em etapas iniciais do projeto causam um aumento no custo final do sistema. Este fato motiva o uso de técnicas cada vez mais apuradas e que possibilitem, em tempo de projeto, estimativas de consumo de energia.

Um dos objetivos de uma estimativa mais apurada de consumo de energia é ter o dimensionamento de uma fonte que permita o funcionamento do sistemas embarcado sem grandes folgas, o que evita um aumento no custo final.

1.2 OBJETIVO

O objetivo principal deste trabalho é, principalmente, propor um modelo de desempenho e energia para aplicações C (considerando plataformas embarcadas). Além disso, consiste na implementação de ferramentas que possibilitam a simulação de tal modelo e procedimentos para realização da estimativa de consumo de potência em tempo de projeto.

Este trabalho apresenta uma metodologia baseada em Redes de Petri Temporizada que representa descrições em código ANSI-C e considera um processador de uma arquitetura específica.

A linguagem ANSI-C foi escolhida por ser uma das linguagens mais utilizadas [5] para o projeto desse tipo de sistema. O modelo de Redes de Petri foi escolhido por ser um modelo matemático para descrição de sistemas de comportamento distribuído e paralelo que provê o formalismo matemático necessário para avaliação quantitativa e qualitativa do sistema[6]. O modelo de Rede de Petri Temporizado foi anotado com informações de consumo de energia e originou a rede denominada Power Petri Net. Para que a avaliação do consumo de energia fosse realizada, adotou-se simulação estocástica como metodologia. Essa simulação é a execução do comportamento descrito pelo modelo Power Petri Net expresso em PNML[1].

Utiliza o resultado de um *Parser* que transforma o código ANSI-C em Redes de Petri no formato XML. Este formato, chamado PNML [1], consiste em uma especificação utilizando XML para padrões de Redes de Petri.

Para validação do método proposto foi utilizado um código de avaliação chamado PowerStone [3]. PowerStone é um *benchmark* especificamente desenvolvido para avaliação de consumo de energia de sistemas embarcados que exercita diferentes aspectos do sistema, tais quais manipulação de área de armazenamento de memória, utilização de expressões aritméticas para cálculos matemáticos, utilização de referência à memória (ponteiros), dentre outros.

A maioria dos trabalhos atuais [7, 8, 9, 10, 11, 12] apresenta uma abordagem baseada em baixo nível de abstração, *hardware*, ou um nível muito alto, baseado em especificações puramente de projeto como, por exemplo, UML [13].

As abordagens de baixo nível são baseadas em informações dos circuitos e detalhes de instruções da máquina. Isto ocasiona uma sobrecarga de dados necessários para a estimativa de consumo de energia, tornando o processo de simulação uma tarefa dispendiosa e demorada. Por outro lado, os trabalhos de mais alto nível exigem um esforço de projeto e informações que, muitas vezes, não correspondem a detalhes do sistema final, não levando a um nível necessário de exatidão.

1.3 TRABALHOS RELACIONADOS

O problema com o consumo de energia vem sendo extensivamente estudado e reportado nas literaturas técnico-científicas. Esta seção apresenta sucintamente um conjunto de trabalhos sobre estimativa e consumo de energia, citando os pontos que foram abordados e técnicas utilizadas.

Em geral, existem dois tipos gerais de tratamento quando se procura estimar o consumo de energia. No primeiro, o foco são os dispositivos de *hardware*; outra visão é mais voltada para projetos, onde o foco é estimar o consumo de energia com base no comportamento geral do sistema.

[7] utiliza medição para construção de uma biblioteca de consumo de energia e utiliza simulação estocástica de Redes de Petri colorida para estimar o consumo associado a cenários de aplicações. A Figura 1.1 apresenta o *framework* proposto para a solução. Nesse trabalho, o código binário do programa do qual deve ser avaliado, é transformado em uma rede de Petri Colorida que é simulada para se estimar o consumo da aplicação. Uma biblioteca de consumo das instruções de máquina fornece os valores de consumo de energia que são anotados na rede.

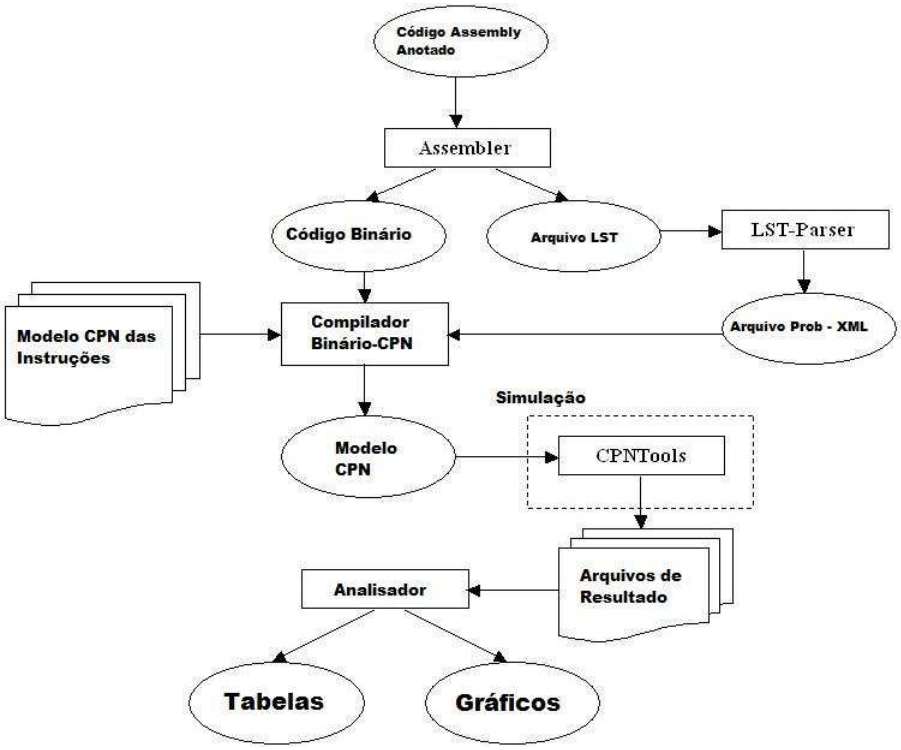


Figura 1.1 *Framework* Proposto pelo trabalho [7]

Este trabalho utiliza uma ferramenta para simulação de Redes de Petri Colorida de propósito geral, o que torna esse processo de simulação lenta. Após a simulação, os dados resultantes são lidos no *framework* EzPetri para exposição dos Resultados de estimativa.

[30] procura comprovar a eficácia do uso da técnica *Dynamic Voltage Scaling* DVS para redução do consumo de energia. DVS é uma técnica que aplica redução de tensão

para reduzir o consumo de energia. Para utilizar essa técnica, é necessário alterar o código gerado, de forma que o compilador automaticamente considere os mecanismos disponíveis associados à redução da tensão de alimentação do processador, de maneira que a aplicação passe a consumir menos energia. O algoritmo proposto no trabalho identifica regiões no programa onde se pode reduzir a tensão do processador ao custo da penalização do desempenho.

A estimativa de consumo de energia é baseada em medições em *hardware* e numa comparação do consumo do código avaliado e da capacidade total da bateria. O *hardware* utilizado nesse sistema pode ser visto na Figura 1.2. Esse Hardware consiste em *Laptop* sem a bateria, um circuito de alimentação ligado à um medidor digital de consumo de energia. A razão de retirar a bateria do LapTop é para se ter acesso à corrente de alimentação do LapTop e assim medir o consumo de Energia. Esse medidor digital também recebe informações de configuração do computador, para assim gerar informações mais precisas do consumo de energia em relação à capacidade total da bateria.

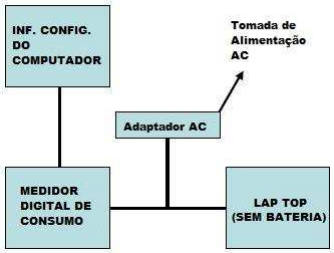


Figura 1.2 *Hardware* para medição do consumo de energia em [30]

Este estudo baseia-se em medição realizada em um computador genérico, no qual vários outros processos estão sendo executados, o uso de formalismo matemático na análise de consumo de energia traz ganhos em relação aos resultados obtidos, ao invés de fazer uma comparação com a capacidade total da bateria.

Um *framework* para estimativa de consumo de energia é apresentado em [31]. Esse *framework*, chamado de *SimplePower*, consiste de uma ferramenta de compilação de código que extrai informações sobre a compilação e estrutura do código, e de um simulador que estima o consumo do *hardware* do sistema.

Para realizar a estimativa, é necessário ter o código fonte, detalhes sobre otimização de compilação, informações relativas ao consumo de energia do barramento de dados, informações sobre dissipação de potência de componentes eletrônicos sobre o consumo de energia de utilização de memória.

É muito difícil, no começo de um projeto de *hardware*, como a maioria dos projetos de sistemas embarcados são formados, ter todos os detalhes sobre o hardware e ferramentas de compilação específicas. Quanto mais dependência se tem dos recursos de hardware do projeto, maior a chance de gastos com o projeto final.

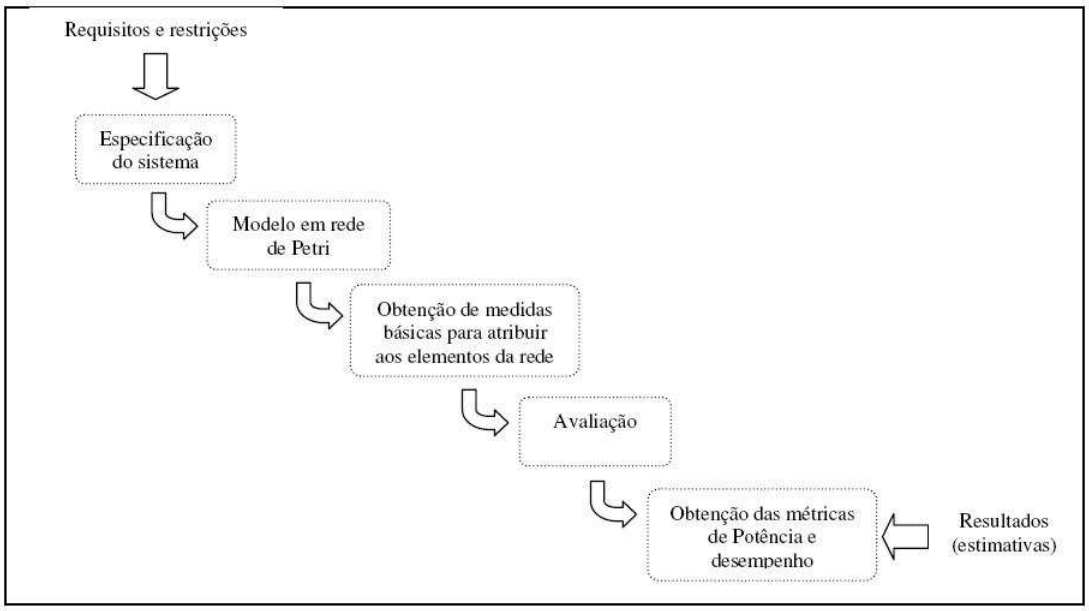


Figura 1.3 Atividades que compõem a metodologia proposta por [8]

[8] utiliza *Generalized Stochastic Petri Net*, GSPN, para estimar consumo de energia de sistemas embarcados. O ponto forte do trabalho, características dos trabalhos em alto nível, é que a estimativa pode ser feita em tempo de projeto, ou seja, sem o *hardware* ser concebido. Como biblioteca de entrada para se obter as informações de consumo de energia, o autor se baseou em medições prévias ou em especificação de consumo de manuais do fabricante.

Um aspecto a ser observado nesse trabalho é a construção manual do modelo final. Uma característica desse projeto é a utilização de aproximações de fase [32] para representar distribuições não-exponenciais. A Figura 1.3 apresenta o esquema e as etapas executadas do processo de estimativa.

1.4 CONTRIBUIÇÕES

Como contribuição deste trabalho, pode-se destacar a implementação de um *Framework* para estimativa de consumo de energia associado ao código C de uma aplicação, ou seja,

definição de um modelo básico e ferramentas para estimativa de consumo de energia que podem ser estendidos em outros trabalhos. Este *Framework* consiste dos seguintes blocos, representados na Figura 1.4.

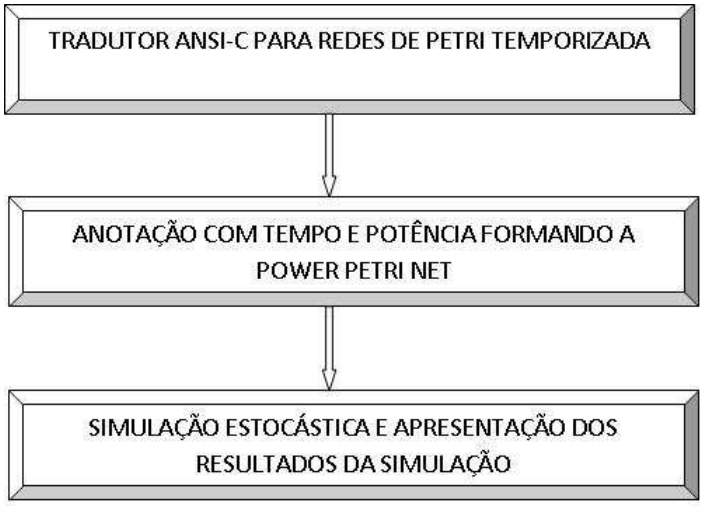


Figura 1.4 Ambiente Computacional de Simulação - Power Petri Net

O primeiro bloco, chamado *PARSER* ANSI-C, é composto por um tradutor que identifica as estruturas da linguagem C. Estas estruturas foram modeladas em Redes de Petri e, após a execução do *Parser*, são traduzidas em PNML que possilitam a simulação da Rede e a estimativa do consumo de energia do sistema.

O segundo bloco constrói o modelo final chamado Power Petri Net. Neste modelo, anotam-se nas Rede de Petri informações de consumo de energia capturadas através de um *hardware* de medição. Este processo gera uma biblioteca de consumo, fazendo com que, para as estimativas seguintes, não seja necessária a utilização de um protótipo. Biblioteca de consumo é um arquivo que contém o valor de consumo e tempo de execução das estruturas adicionadas a este, formando a Rede Power Petri Net. Este modelo, portanto, auxilia o projetista antes mesmo do projeto de *hardware* ser concebido.

O último bloco é um simulador estocástico que permite ao projetista montar cenários, através da atribuição de probabilidades e, assim, conseguir fazer um estudo amplo das suas possiblidades de consumo de energia, ajudando no processo de decisão.

1.5 ORGANIZAÇÃO

O Capítulo 2 apresenta os principais conceitos utilizados nesta dissertação, como Sistemas Embarcados, Modelos Temporais, Processos Estocásticos, Redes de Petri Temporizadas e Introdução à Simulação de Sistemas de Eventos Discretos. O Capítulo 3 apresenta a arquitetura geral do trabalho com o fluxo de atividades e propostas de solução para o problema, enquanto o Capítulo 4 elenca os trabalhos relacionados. Após isto, o Capítulo 5 descreve o método utilizado para modelagem da linguagem ANSI-C em Redes de Petri e faz a devida anotação com tempo e potência. O Capítulo 6 explica o método de extração das informações de tempo e consumo de energia do processo chamado de caracterização do processador. No Capítulo 7 é especificado o simulador desenvolvido; no Capítulo 8, os resultados dos experimentos são apresentados e, por fim, no Capítulo 9, é feita a conclusão do trabalho, a avaliação e a indicação de projetos futuros.

CAPÍTULO 2

FUNDAMENTOS

Neste capítulo serão apresentados os fundamentos necessários à compreensão da metodologia proposta nesta dissertação.

A primeira parte é composta por uma introdução aos Sistemas e Modelos em geral, seguida por apresentação de Modelos Temporais. Também é apresentada uma seção sobre modelos estocásticos usados no trabalho sob a forma de Redes de Petri Temporizadas de Zuberek[14], que é introduzida logo em seguida.

Por fim, é feita uma explanação sobre Simulação de Eventos Discretos.

2.1 SISTEMAS E MODELOS

Um sistema é uma coleção de entidades, que podem ser pessoas, máquinas, módulos de *software* ou de *hardware* que interagem entre si e das quais podemos determinar relações e comportamentos [15].

A análise do comportamento dos sistemas pode ser feita de duas maneiras: realização de experimentos com o sistema real; ou através de experimentos virtuais com modelos. A utilização dos sistemas reais, em geral, possui alguns fatores limitantes:

- o sistema pode ainda não estar disponível por estar sendo projetado;
- o sistema não está acessível; e
- não é possível atuar no sistema em uma escala de tempo e espaço viáveis.

Um modelo é uma representação de um ou mais pontos de vista de um sistema em um determinado nível de abstração. Em um modelo formal, um sistema é representado através de um conjunto de variáveis, funções e relações matemáticas que representam uma ou mais perspectivas do sistema.

O estudo de sistemas envolve uma série de parâmetros de entrada e saída, e, a esses parâmetros podem ser associadas variáveis. Essas podem ser classificadas como Contínuas

ou Discretas. As variáveis contínuas podem assumir qualquer valor no conjunto dos números reais, enquanto as variáveis discretas assumem um determinado subconjunto de valores e suas variações são sempre em quantidades pré-definidas, como por exemplo, no conjunto de números inteiros.

Enquanto um sistema é "algo real", o modelo é uma "abstração", como por exemplo, um conjunto de equações matemáticas. A Figura 2.1 mostra a representação de um sistema através de um modelo.

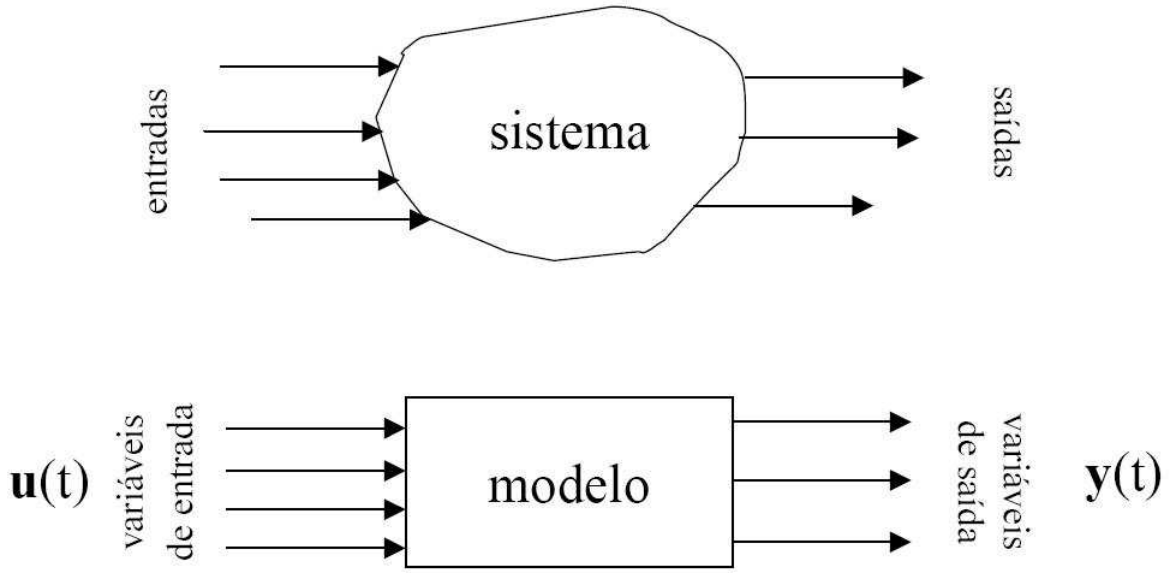


Figura 2.1 Processo de Modelagem Simples

Espera-se do modelo que ele estabeleça relações entre estas variáveis. Esta relação é representada pela função g formando uma relação entre $y(t)$ e $u(t)$ como mostrado a seguir:

$$y(t) = g(u(t))$$

Os modelos de sistemas podem ser classificados como:

- Físicos ou Matemáticos;
- Estáticos ou Dinâmicos;
- Lineares ou Não-lineares;
- Variantes ou Invariantes no tempo;

- Analíticos ou Numéricos.

Modelos invariantes no tempo podem ser representados por $y(t) = g(u(t))$, enquanto variantes no tempo são representados por $y(t) = g(u(t), t)$.

Um modelo é dinâmico se os valores das saídas dependem de valores passados das entradas. Esta propriedade de um sistema linear invariante no tempo é o princípio da superposição. Este princípio diz que a saída de um sinal formado pela combinação linear de diferentes sinais é igual à mesma combinação aplicada aos sinais de saída gerados por cada sinal original separadamente. Este princípio está representado na Tabela 2.1:

Tabela 2.1 Princípio da Superposição

entrada	saída
u_1	y_1
u_2	y_2
$\alpha\ u_1 + \beta\ u_2$	$\alpha\ y_1 + \beta\ y_2$

O exemplo da Figura 2.2 é de um modelo matemático, estático, linear, invariante no tempo e analítico de um circuito elétrico. Este circuito elétrico representa a aplicação prática das leis das tensões da 2^a Lei de Kirchhoff.

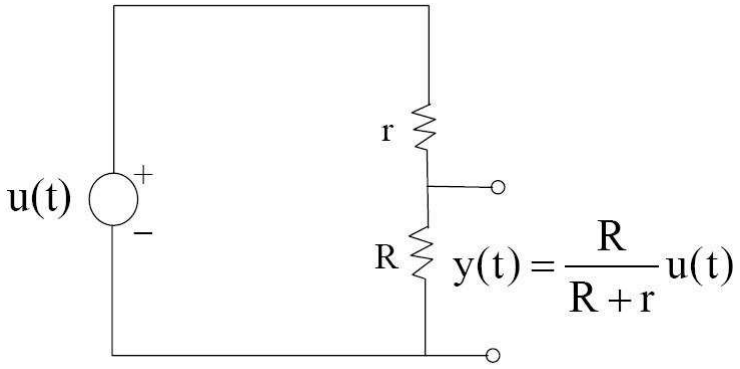


Figura 2.2 Modelo de um Circuito Elétrico

A classificação apresentada não é exclusiva e tem a finalidade de descrever o escopo dos diferentes aspectos da teoria de sistemas, assim como ajuda a identificar suas principais características. A Figura 2.3 fornece uma referência rápida para as classificações de sistemas [16].

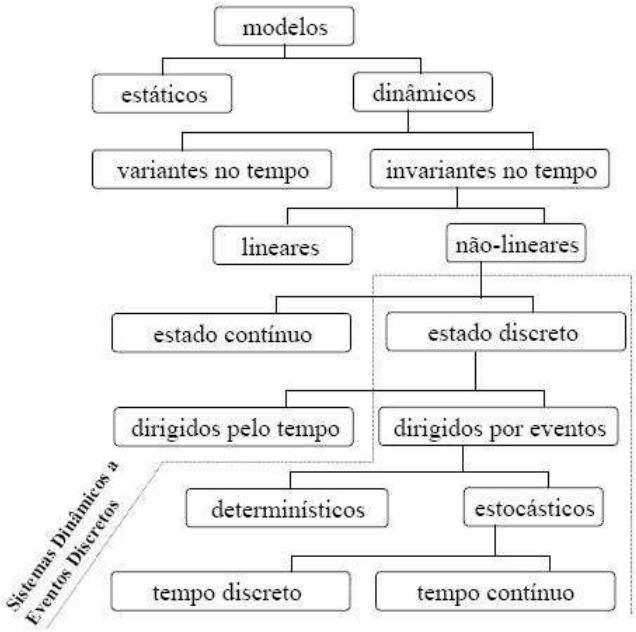


Figura 2.3 Principais Classificações de Sistemas

Diversos modelos de representação têm sido utilizados para representar sistemas computacionais, dentre eles, redes de filas, álgebras de processos, autômatos e Redes de Petri são os mais utilizados. Descrições detalhadas desses modelos são encontradas em [16].

A não linearidade dos Sistemas de Eventos Discretos (SED) é inerente às descontinuidades (saltos) das transições entre os estados resultantes da ocorrência de um evento. Dentro da classe de SED podem-se considerar modelos de sistemas determinísticos ou estocásticos, de estados discretos ou contínuo, e de tempo discreto ou tempo contínuo.

Para explicar o que é um SED, é importante a compreensão do que vem a ser um estado. Estado é a informação necessária num instante de tempo para se prever a saída futura do sistema, caso os valores futuros das variáveis de entrada sejam conhecidos. Os modelos podem ser dirigidos pelo tempo ou dirigidos pela ocorrência de eventos.

Neste último caso, o estado só muda quando ocorre um evento. Por definição, este não possui uma duração específica e altera o estado do sistema.

Um modelo com espaço de estados contínuo, cujas trajetórias sejam contínuas, tem sua dinâmica naturalmente dirigida pelo tempo.

Um modelo com espaço de estados discreto pode apresentar:

- Dinâmica dirigida pelo tempo → eventos ocorrem sincronizados por um relógio.

- Dinâmica dirigida pela ocorrência de eventos → eventos ocorrem de maneira assíncrona e concorrentemente.

Os Sistemas Dinâmicos a Eventos Discretos possuem algumas características, tais como: espaços de estado discretos, dinâmica dirigida pela ocorrência de eventos, sincronismo e concorrência. No que se refere às interpretações, de forma geral, podem-se classificar como: não-temporizado ou lógico, temporizado.

2.2 MODELOS TEMPORAIS

Os Sistemas Dinâmicos a Eventos Discretos podem ser subdivididos em Modelos de Tempo Determinístico e Modelos de Tempo Estocástico. São exemplos de Modelos de Tempo Determinístico: automatos temporizados, Máquina de Estado Finita Extendida, Redes de Petri Temporizada e Álgebra de Processos em Tempo Real. São exemplos de Modelos de Tempo Estocástico: Modelos de Tempo Estocástico & Cadeias de Markov, Redes de Fila, Redes de Petri Estocásticas, Álgebra de Processos Estocásticos. Neste trabalho, utiliza-se o Modelo de Tempo Determinístico denominado Redes de Petri Temporizada.

Dentre as áreas de forte aplicação dos modelos temporais, podem-se ressaltar os sistemas com restrições de tempo real. Em sistemas em tempo real, a preocupação com o tempo é fundamental, pois a resposta não deve apenas ser correta, mas deve ser provida dentro das restrições temporais impostas pelo sistema. Nos Sistemas em Tempo Real Crítico, não respeitar esses requisitos pode levar a resultados catastróficos. Portanto, além da capacidade de representar mecanismos de comunicação, sincronização, compartilhamento de recursos e concorrência, o modelo formal deve prover também métodos para capturar informações de tempo. A FSM, Máquinas de Estado Finito (*Finite State Machine*), Álgebra de processo e Redes de Petri são exemplos desses formalismos que possuem suas extensões temporizadas para tratar da modelagem das restrições de tempo.

2.3 PROCESSOS ESTOCÁSTICOS

A avaliação de desempenho e confiabilidade de sistemas através de métodos analíticos tem sido objeto de inúmeros estudos nas áreas de Ciência da Computação e Matemática Aplicada. Processos estocásticos são modelos matemáticos úteis para a descrição de um fenômeno de natureza probabilística, como uma função de um parâmetro, que, usual-

mente, tem o significado de tempo.

Um processo estocástico $X(t), t \in T$ é uma família de variáveis aleatórias definidas sobre o mesmo espaço de probabilidades, indexadas ou que são funções de um parâmetro t e assumem valores no espaço de estado S. Geralmente, t tem o significado de tempo, e T, intervalo de tempo. O espaço de estado S compreende o conjunto dos possíveis valores de $X(t)$ [17].

Classificam-se os processos estocásticos de acordo com o tipo da variável aleatória associada ao processo. O processo estocástico de tempo contínuo tem associada uma variável aleatória de tempo contínuo, em que o espaço de estados pode ser discreto ou contínuo. Enquanto que o processo estocástico de tempo discreto tem associada uma variável aleatória de tempo discreto, cujo espaço de estado pode ser discreto ou contínuo.

A Figura 2.4 mostra a representação gráfica de dois processos estocásticos, o de tempo contínuo (a) e o de tempo discreto (b). Ambos têm espaço de estado discreto.

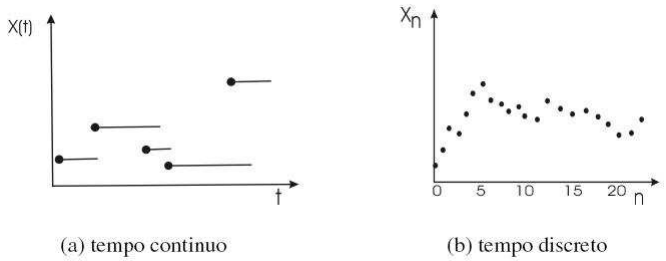


Figura 2.4 Amostras de Processos Estocásticos

Em geral, a descrição probabilística completa de um processo aleatório genérico não é factível. Processos Markovianos são uma classe especial de processo estocástico para o qual a descrição probabilística é simples e de relevância particular.

2.3.1 Processos Markovianos

Definem-se Processos Markovianos como os processos estocásticos que satisfazem as propriedades Markovianas:

- M1: a evolução futura dos estados não depende dos estados anteriores;
- M2: a evolução futura dos estados não depende do tempo de permanência no estado atual;

As propriedades Markovianas também são conhecidas por "ausência de memória". Denominam-se os processos Markovianos de tempo contínuo e espaço de estado discreto de Cadeias de Markov de Tempo Contínuo (CTMC). Os Processos Markovianos com espaço de estado discreto e tempo discreto são denominados de Cadeias de Markov de Tempo Discreto (DTMC).

2.4 REDES DE PETRI

As Redes de Petri (RP) são uma família de técnicas formais muito bem estabelecida, que consegue modelar concorrência, sistemas assíncronos, sistemas não-determinísticos, etc. Vários autores têm mostrado um grande número de usos para RP em diferentes áreas tais como, Ciência da Computação, Engenharia Eletrônica, Química, Gerenciamento de Negócios, sSistemas de Produção, etc.

Algumas classes de Redes de Petri serão comentadas, mas apenas a extensão temporizada das redes de interesse deste trabalho será tratada com mais detalhes. Juntamente com a extensão temporizada será apresentado o modelo de Zuberek [14], que inclui a adoção de probabilidades nas transições em conflito, dentre outras diferenças apresentadas.

2.4.1 Componentes de uma Rede de Petri

O grafo de uma Rede de Petri direcionado e bipartidte, possui pesos e dois tipos de componentes, chamados lugares (*places*) e transições (*transitions*) com arcos que vão de um lugar a uma transição ou de uma transição para um lugar.

Graficamente, representam-se lugares com círculos e transição com barras ou retângulos (ver Figura 2.5). Os arcos são rotulados com seus pesos (exceto se o peso for igual a um), e um arco de peso k pode ser interpretado como k arcos paralelos.

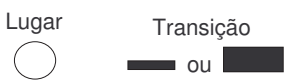


Figura 2.5 Elementos Básicos da Rede de Petri

Quando o conceito de condição-evento estiver sendo usado na modelagem de um sistema, lugares e transições representam condições e eventos, respectivamente. Os lugares são ditos componentes passivos e as transições são componentes ativos. Uma transição

Tabela 2.2 Algumas interpretações típicas para lugares e transições

Lugares de Entrada	Transição	Lugares de Saída
Pré-condições	Evento	Pós-condições
Dados de entrada	Passo computacional	Dados de saída
Sinais de entrada	Processador de sinais	Sinais de saída
Recursos Requeridos	Tarefa	Recursos liberados
Condições	Cláusula Lógica	Conclusões
Buffers	Processador	Buffers

possui um certo número de lugares de entrada e saída, o que representa as pré e pós condições do evento observado. A realização de uma ação, portanto, está associada a algumas pré-condições, ou seja, existe uma relação entre os lugares e as transições, o que possibilita a realização de uma ação. De forma semelhante, após a realização de uma ação, alguns lugares terão suas informações alteradas (pós-condições). Algumas interpretações típicas de transições e seus lugares de entrada e saída são mostradas na Tabela 2.2.

Dados os conceitos informalmente apresentados sobre Redes de Petri, ilustramos a seguir um pequeno exemplo para entendimento: o ciclo repetitivo dos turnos (períodos) de um dia. Dividamos o dia em três períodos: manhã, tarde e noite, ou seja, há três condições. A transição de uma dessas condições para uma outra, por exemplo- amanhecer (noite $\rightarrow$ manhã), são os eventos. O modelo que representa o ciclo operacional desse sistema é formado pelas três condições, representadas por três variáveis de estado (lugares), e por três eventos (transições): amanhecer, entardecer e anoitecer. Para representar a situação atual, ou seja, em que condição encontra-se o sistema modelado, usa-se uma marca grafada (um ponto) no lugar que corresponde a essa situação, por exemplo: a condição atual é manhã. Na Figura 2.6.a tem-se o modelo que representa este sistema e a sua situação atual.

Nesse modelo, temos a condição atual representada pela marca no lugar manhã. Estando nesta condição, o único evento que poderá ocorrer é entardecer, que é representado pela transição de mesmo nome. Na ocorrência desse evento, teremos uma nova situação atual, ou seja, tarde, que é representada graficamente, na Figura 2.6.b por uma marca no lugar tarde.

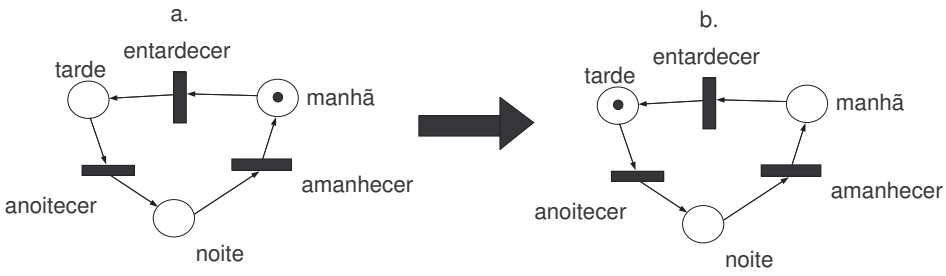


Figura 2.6 Períodos do dia representados com Redes de Petri.

2.4.2 Redes Place-Transition

Podem-se definir mais formalmente as Redes de Petri de diversas maneiras distintas, entre estas, as que mais se notabilizam são as representações matriciais [20], através da teoria bag e por meio de relações entre estados locais e eventos [21]. A definição sob o ponto de vista da teoria bag [22] apresenta mapeamentos das transições para bags de lugares. Neste trabalho será apresentada apenas a definição segundo a notação da teoria bag.

A estrutura das Redes de Petri, segundo a teoria bag [22] é composta por cinco partes: o conjunto de lugares P, o conjunto de transições T, o bag de entrada I, o bag de saída O e a capacidade associada a cada lugar. Para cada transição existe uma função de entrada que é um mapeamento de uma transição t_j em um bag de lugares $\mathrm{I}t_j$. De forma semelhante, as funções de saída mapeiam uma transição t_j em um bag de lugares $\mathrm{O}t_j$. Para denotarmos conjuntos, utilizamos - { } e para os *bags* - [ ].

Definição 2.4.1. *Estrutura das Redes de Petri em bag: define-se a estrutura de uma Rede de Petri R, como uma quíntupla* $R = (P, T, I, O, K)$*, onde* $P = \{p_1, p_2, ..., p_n\}$ *é um conjunto finito não-vazio de lugares,* $T = \{t_1, t_2, ..., t_m\}$ *é um conjunto não-vazio de transições.* $I : T \rightarrow P^{\infty}$ *é um conjunto de bags que representa o mapeamento de transições para lugares de entrada.* $O : T \rightarrow P^{\infty}$ *é um conjunto de bags que representa o mapeamento de transições para lugares de saída.* $K : P \rightarrow \aleph \bigcup \{\omega\}$*; é o conjunto das capacidades associadas a cada lugar, podendo assumir um valor infinito.*

Uma marca, que pode ser também denominada ficha ou *token*, é um conceito primitivo em Redes de Petri. As marcas são informações atribuídas aos lugares. Uma marcação, ou estado, associa um inteiro k, não-negativo, a cada lugar da rede. A seguir serão vistas as seguintes definições: a definição formal de marcação; a definição formal de marcação na forma vetorial; e a definição de Rede de Petri marcada.

Definição 2.4.2. *(Marcação) Seja P o conjunto de lugares de uma rede R. Define-se formalmente marcação como uma função que mapeia o conjunto de lugares P a inteiros não-negativos $M : P \rightarrow \aleph$.*

Definição 2.4.3. *(Vetor Marcação) Seja P o conjunto de lugares de uma rede R. A marcação pode ser definida como um vetor $M = (M(p_1), ..., M(p_n))$, onde $n = \sharp P$, para todo $p_i \epsilon P$ tal que $M(p_i) \epsilon \aleph$.*

Definição 2.4.4. *(Rede Marcada) Define-se uma Rede de Petri marcada pela tupla $RM = (R, M_0)$, onde R é a estrutura da rede e M_0 é a marcação inicial.*

O comportamento dos sistemas pode ser descrito em função dos seus estados e suas alterações. Para simular o comportamento dos sistemas, a marcação da Rede de Petri é modificada a cada ação realizada (transição disparada), segundo regras de execução apresentadas a seguir.

O disparo de transições (execução das ações) é controlado pelo número e distribuição de marcas nos lugares. A transição está habilitada se cada um dos seus lugares de entrada possuir um número de *tokens* pelo menos igual ao peso do arco que os liga. Denota-se a habilitação de uma transição t para uma marcação M_k por $M_k[t > M_i$.

Definição 2.4.5. *(Regra de Habilitação) Seja $R = (P, T, I, O, K, M_0)$ uma rede, $t \in$ T uma transição e M_k uma marcação. Se $M_k[t >, M_k(p_i) \geq I(p_i, t), \forall p_i \in$ P.*

Formalmente, apresentamos as regras de execução das Redes de Petri a seguir:

Definição 2.4.6. *(Regra de Disparo) Seja $R = (P, T, I, O)$ uma rede, $t \in$ T uma transição e M_k uma marcação. A transição t pode disparar quando está habilitada. Disparando uma transição habilitada, a marcação resultante é $M_i = M_k - I(p_j, t) + O(p_j, t), \forall p_j \in$ P. Se uma marcação M_i é alcançada por M_k pelo disparo de uma transição t, ela é denotada por $M_k[t > M_i$.*

2.4.3 Redes de Petri e suas Classes

Duas características de um modelo de Redes de Petri que tornam-as úteis são níveis de abstração e as diferentes interpretações [23, 24]. Estas características especificam um espaço de formalismos apropriados para diferentes propósitos e são apresentados na Figura 2.7 .

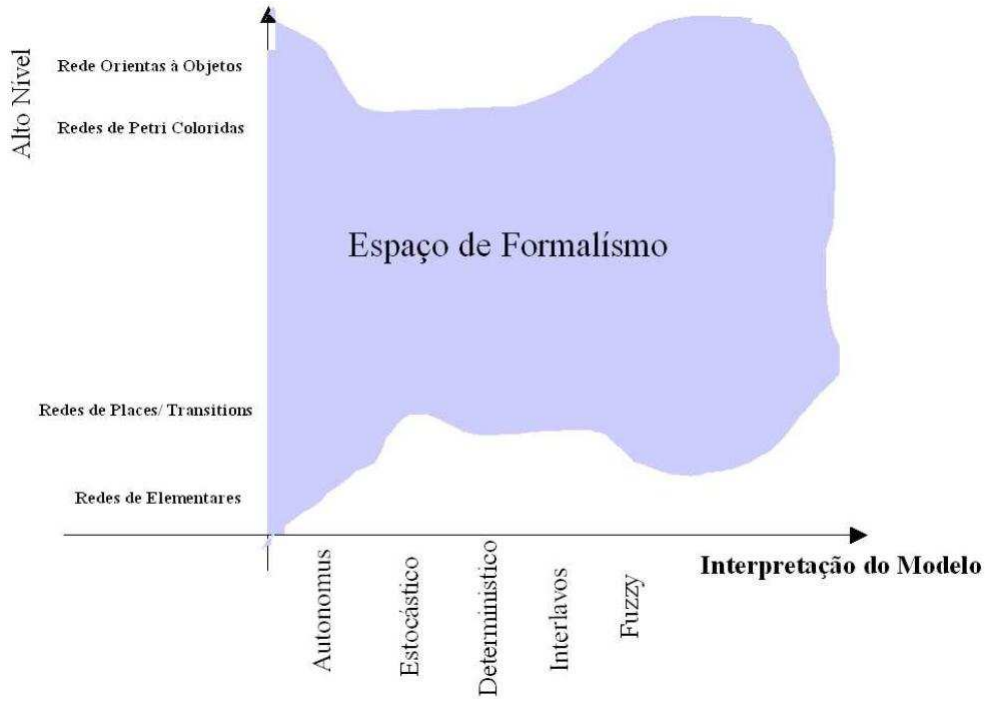


Figura 2.7 Abstração X Interpretação.

A primeira classe no eixo do nível de abstração são as redes elementares, onde lugares representam condições booleanas. A segunda classe são as redes *Place-Transition*, onde lugares são contadores. Adicionalmente algumas extensões podem ser consideradas tais como arco inibidores e redes com prioridades. Tais extensões afetam não somente a concisão, mas também o poder de modelagem dos modelos adotados.

As subclasses de redes são definidas pela introdução de restrições para a estrutura de redes *Place-Transition*. Pelo fato de restringir a generalidade de um determinado modelo, torna-se fácil estudar seus comportamentos [29].

As Redes de Petri denominadas *Free-Choice* permitem representar o conflito e como é o mecanismo para a resolução desse conflito. Conflito consiste, informalmente, quando tem-se duas ou mais transições habilitadas podendo qualquer uma delas ser escolhida para ser disparada. Na verdade, esta subclasse de rede permite modelar concorrência, sincronização e modelos de conflitos, no entanto, de uma maneira mais restrita do que a rede *Place-Transition*.

Definição 2.4.7. *(Free-Choice Nets) Seja* $R = (P, T, I, O, M_0)$ *uma rede classificada como uma free-choice se* $p_i \in I(t_j)$*, então* $I(t_j) = p_i$ *ou* $O(p_i) = t_j$*.*

Esta classe de rede permite ter conflitos controlados, pois quando um lugar é entrada

de diversas transições, este lugar é a única entrada destas transições. Desta forma, todas as transições estarão habilitadas ou nenhuma estará, possibilitando a escolha da realização do evento livremente.

A Rede de Petri temporizada de Zuberek [14], também chamada de *Timed Petri Net*, é uma extensão da Rede de Petri Place-Transition com informações de tempo e com probabilidades de disparo nas transições em conflito. A Rede de Petri Place-Transition não possibilita avaliação de desempenho de modelos temporizados, visto que a mesma não possui informação da duração das atividades. Zuberek propôs uma Rede de Petri que tivesse tempo associado às transições e, nos disparos, os *tokens* são retirados dos lugares de entrada das transições e permanecem no "interior" da transição durante o período de disparo.

Abaixo segue a definição formal da Timed Petri Net:

Definição 2.4.8 (Timed Petri Net)**.** *Seja uma tupla* $Nt = (N, D, C)$ *uma Timed Petri Net, onde* $N = (P, T, I, O, M_0)$ *é uma Rede de Petri,* $D : T \rightarrow \Re^+ \cup \{0\}$ *é uma função que associa a cada transição* t_i *uma duração de disparo* d_i *e* $C : T \mapsto \Re$ *uma função de probabilidade que mapeia transições em conflito da rede às suas probabilidades de disparo.* C *é definido de forma que*

$$\forall T_i \in Free(T) : \sum_{t \in T_i} C(t) = 1$$

. Onde $Free(T)$ *é um conjunto de transições sem confusão (*Confusion Free*).*

P é o conjunto de lugares (estados locais), T é o conjunto de transições (eventos e ações), I é o multi-conjunto de lugares de entrada das transições, O é o multi-conjunto de lugares de saída das transições, e $M_0 : P \rightarrow N$ é o mapeamento que define a marcação inicial da rede.

Um estado em uma *Timed Petri Net* é definido como:

Definição 2.4.9 (Estado de uma Timed Petri Net)**.** *Seja* Nt= (N, D, C) *uma Timed Petri Net, um estado* S *de* Nt *é definito como uma tupla* $S = (M, MT, RT)$*, onde* $M : P \rightarrow \aleph$ *é a marcação,* $MT : T \rightarrow \aleph$ *é a distribuição de* tokens *nas transições em disparo e* $RT : K \rightarrow \Re_T^+$ *é a função do tempo restante de disparo para cada disparo independente, ou seja, para cada* token *interno para cada transição* $mt(t) \neq 0$*.* K *é o número de* tokens *dentro de uma transição* t_i $(mt(t_i) = K)$*.* RT *é indefinido para as transições* t_i *em que* $mt(t_i) = 0$*.*

A regra de disparo de uma Timed Petri Net é definido como:

Definição 2.4.10 (Regra de Disparo)**.** *Seja $Nt = (N, D, C)$ uma Timed Petri Net, e $S = (M, MT, RT)$ o estado da Nt. Diz-se que o conjunto de transições BT é habilitado no instante x se somente se, $M(p) \geq \sum_{\forall t_i \in BT} \sharp BT(t_i) \times I(p, t_i)$, $\forall p \in P$, onde $BT(t_i) \subseteq BT$ e $\sharp BT(t_i) \in \aleph$.*

BT é o conjunto de todas as transições habilitadas no instante x e M(p) representa a marcação dos lugares de entrada dessas transições, satisfazendo a condição necessária para o disparo de cada transição. Nesse momento, o *token* é retirado do lugar de entrada e consumido pelas transições do conjunto BT, é incrementado o conjunto de *tokens* e o tempo de disparo começa a ser contado.

2.5 SIMULAÇÃO DE EVENTOS DISCRETOS

Simulação é a execução de um modelo de forma a permitir previsões sobre o comportamento do sistema que ele representa.

Uma simulação pode ser efetuada através de um modelo de comportamento análogo ao sistema em estudo. Esse modelo pode ser construído por computação numérica usada em conjunção com um modelo matemático dinâmico.

As simulações permitem fazer inferências sobre os sistemas sem precisar construí-los, sem perturbá-los e sem destruí-los. Elas são usadas no estudo de sistemas como ferramenta explanatória para definição de um sistema ou problema, como ferramenta de análise para a detecção de elementos críticos, como ferramenta de síntese e avaliação de soluções propostas e como ferramenta de planejamento para desenvolvimentos futuros.

A simulação pode ser estacionária ou transiente. Simulação estacionária, também conhecida como *non-terminating simulation*, é realizada sem uma duração finita, não possui um evento para determinar sem o final. Geralmente nesse tipo de si-mulação se ignora o início, tempo de *warm-up*, para não influenciar o resultado final. Na transiente temos uma caraterística de finalização da simulação, ou seja, o resultado da simulação converge para um valor que pode ser considerado final de acordo com alguma pré-condição.

Um autômato temporizado estocástico, assim como as Redes de Petri Temporizada de Zuberek[14], pode ser representado por um esquema de escalonamento de eventos. Na Figura 2.8 tem-se um diagrama com as principais funções de um simulador estocástico.

Os componentes de um programa de simulação são:

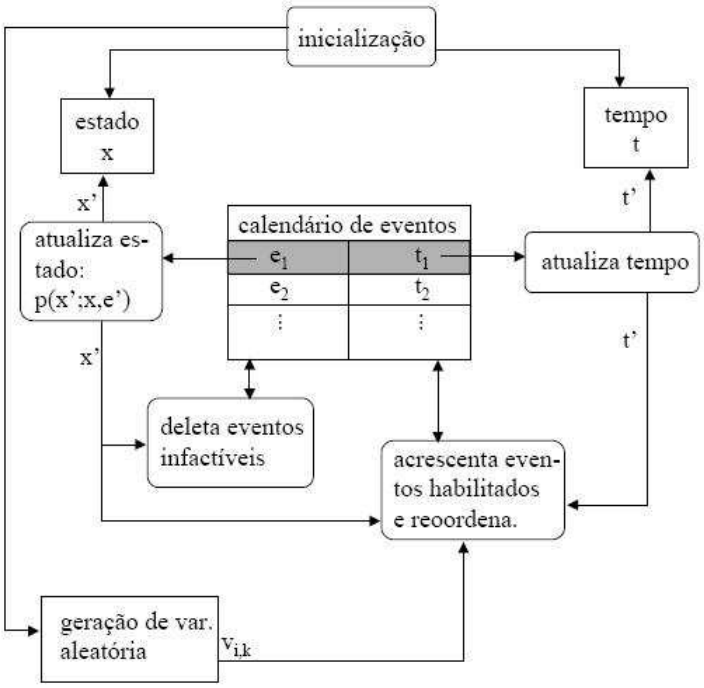


Figura 2.8 Exemplo de um Sistema de Escalonamento de Eventos

1. Estado: uma lista onde as variáveis do estado estão armazenadas;

2. Tempo: uma variável onde o tempo de simulação é armazenado;

3. Calendário de Eventos: uma lista onde todos os eventos escalonados são armazenados juntamente com seu instante de ocorrência;

4. Rotina de Inicialização: inicializa todas as estruturas de dados descritas, ao início da simulação;

5. Rotina de Atualização do Tempo: identifica o próximo evento a ocorrer e avança o tempo de simulação para o tempo de ocorrência daquele evento;

6. Rotina de Atualização de Estado: atualiza o estado através de sorteio com base no próximo evento;

7. Rotinas de Geração de Variáveis Aleatórias: geram, a partir de números aleatórios gerados pelo computador, amostra das variáveis aleatórias associadas aos tempos de vida dos eventos e escolhas entre eventos, de acordo com as distribuições definidas a priori;

8. Rotina de Geração de Relatórios: calcula estimativas das grandezas de interesse a partir dos dados coletados na execução de uma simulação;

9. Programa Principal: responsável pela coordenação geral de todos os componentes; chama inicialmente a rotina de inicialização; durante a execução chama repetidamente as rotinas de atualização e atualiza o calendário; termina o programa.

Para melhor visualização da prática de simulação de Sistemas de Eventos Discretos encontra-se a seguir um exemplo prático da simulação de uma fila, que se encontra na Figura 2.9.

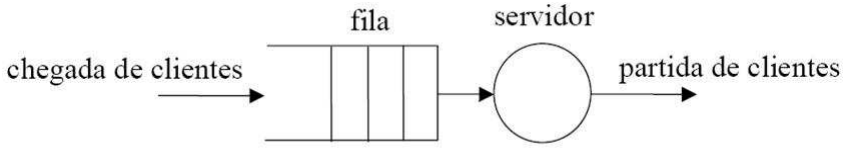


Figura 2.9 Simulação de um esquema de fila

Os ítens de execução de uma simulação do esquema de fila é formado por:

P1 Remover a primeira linha do calendário;

P2 Atualizar o tempo com o valor t_1;

P3 Atualizar o estado de acordo com p(x‘;X, e_1);

P4 Apagar do calendário as linhas correspondentes a eventos infactíveis no novo estado;

P5 Acrescentar ao calendário os eventos factíveis que ainda não estejam escalonados; o tempo associado ao novo evento é obtido adicionando-se ao tempo corrente um valor sorteado da estrutura de relógio;

P6 Reordenar o calendário por ordem crescente do valor do tempo em que os eventos foram escalonados.

Como já foi definido, o objetivo da simulação é obter alguns valores do sistema. Então, antes de qualquer simulação, os dados a serem extraídos têm que ser definidos.

No caso do exemplo do esquema de fila os dados desejados correspondem a: Tempo do Sistema Médio, ou seja, o tempo que um cliente fica na fila e a Probabilidade de exceder o “*deadline*”.

Desta forma temos dois tipos de informações necessárias: 1-Informações Relevantes para a simulação, representados na Figura 2.10 e 2-Informações relevantes para a estimativa de parâmetros, representados na Figura 2.11.

tempo	estado
0,0	0

lista de eventos escalonados	
tipo	instante

Figura 2.10 Informações Necessárias para a Simulação

n_N	0

duração-comp.-fila	
T(0)	
T(1)	
T(2)	
T(3)	

tempos de chegada	
1	
2	
3	
4	
5	

tempos de sistema	
1	
2	
3	
4	
5	

Figura 2.11 Informações Necessárias para as Estimativas

Como entrada do exemplo de simulação temos uma série de eventos gerados por computador, ou seja, os dados de entrada da simulação. Esses dados definem o tempo de chegada e de saída de clientes na fila para simular a estimativa dos parâmetros citados acima. O diagrama de chegada dos eventos pode ser encontrado na Figura 2.12.

Com esses dados pode-se executar a simulação. As Figuras a seguir mostram alguns passos durante a simulação completa do diagrama de chegada e partida de eventos.

A Figura 2.13 representa o inicio da simulação, onde a única informação disponível é o escalonamento do primeiro evento, ou seja, o momento em que o primeiro usuário chegará a fila.

A Figura 2.14 representa o primeiro estado da simulação, que representa o tempo 0.4. Neste momento o primeiro usuário chegou à fila e o simulador já calculou o instante em que o segundo usuário chegará, que é no segundo 0.7. Outra informação representada é o momento em que o primeiro usuário terá seu atendimento finalizado, 2.0.

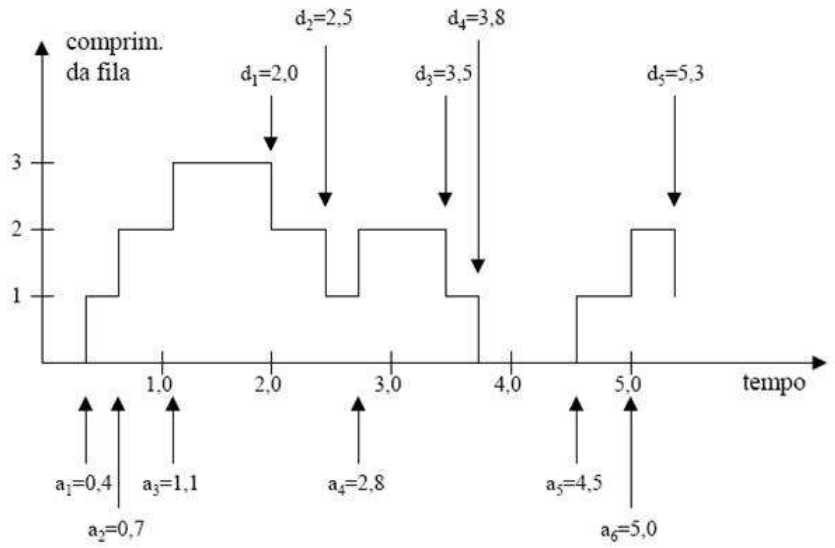


Figura 2.12 Diagrama dos eventos chegarão e partirão da fila como dados de entrada da simulação

t=0 → inicialização.

TEMPO	ESTADO
0.0	0

LISTA DE EVENTOS ESCALONADOS	
CHEGADA	0.4

nN	0

TEMPO DE CHEGADA	
1	
2	
3	
4	
5	

TEMPO DO SISTEMA	
1	
2	
3	
4	
5	

TAMANHO DA FILA	
T(0)	0.0
T(1)	0.0
T(2)	0.0
T(3)	0.0
T(4)	0.0

Figura 2.13 Passo 1 da Simulação

A Figura 2.15 representa o segundo estado da simulação, que é a chegada do segundo usuário na fila. Podem-se observar nas variáveis da simulação que já tem dois usuários na fila e verifica-se, através da lista de eventos escalonados, que o terceiro usuário chegará antes que algum parta, no caso no instante 1.1.

A Figura 2.16 representa o terceiro estado, passo número 4 da simulação. Pode-se observar a presença agora de 3 usuários na fila e que o próximo evento habilitado é o evento de partida de um usuário, que acontecerá no instante 2.0.

Figura 2.14 Passo 2 da Simulação

Figura 2.15 Passo 3 da Simulação

A Figura 2.17 representa o quarto estado, que descreve o instante 2.0 que indica a saída do primeiro usuário da fila. Neste momento as fórmulas de estimativa da simulação já são utilizadas e na coluna de tempo do sistema já é armazenada a primeira informação do tempo de espera do primeiro usuário na fila 1.1.

Assim segue a simulação até o estado representado na Figura 2.18 que representa a saída do último usuário até a quantidade de 5 usuários entrando e saindo do sistema, que é o critério de parada da simulação, ou seja, verificar os resultados até a saída de 5 usuários do sistema.

No final da simulação as estimativas são extraídas das fórmulas:

Tempo de Sistema Médio:

Figura 2.16 Passo 4 da Simulação

Figura 2.17 Passo 5 da Simulação

$$\hat{S_n} = \frac{1}{N} \sum_{k=1}^{N} S_k$$

$$\text{Resultado:} S_5 = (1.6 + 1.8 + 2.4 + 1.0 + 0.8)/5 = 1.52$$

Probabilidade de exceder o "deadline":

$$\hat{P_n^D} = \frac{n_N}{N}$$

$$\text{Resultado:} P_5^D = 1/5 = 0.2$$

O processo de simulação inclui o critério de parada, que podem ser vários. No exemplo demonstrado o critério de parada era quantidade de clientes que chegam e saem da fila. Quando a quantidade de clientes que chegaram e saíram chegar a 5, a simulação atinge o

Figura 2.18 Passo 12 da Simulação

seu critério de parada. Esse critério pode variar, dependendo do tipo de simulação e do resultado esperado.

2.6 CONSIDERAÇÕES FINAIS

Este capítulo resumiu a fundamentação necessária para o entendimento dos capítulos seguintes, tratou sobre modelagem e avaliação de desempenho, modelos temporais, processos estocásticos, Redes de Petri e, por fim, uma breve introdução sobre simulação de eventos discretos.

CAPÍTULO 3

ARQUITETURA GERAL DO TRABALHO

Com o objetivo de se ter um processo automático de estimativa de consumo de energia, este trabalho propõe um *framework* para análise de código fonte de sistemas. O *framework* proposto é uma extensão do EZPetri [2], que é um ambiente para desenvolvimento de ferramentas baseadas em Redes de Petri.

A Figura 3.1 mostra a arquitetura proposta. Pode-se observar que as entradas para a realização da estimativa de consumo são o código do software escrito em ANSI-C e a biblioteca de consumo de energia das estruturas básicas do ANSI-C que serão apresentados no capítulo 4. O primeiro bloco consiste em um *parser* que traduz a descrição de código em um arquivo XML que será a entrada do simulador. Esse arquivo XML contém as informações de consumo de energia, o que forma a *Power Timed Petri Net* apresentada no capítulo 4. O último bloco descreve o simulador em si que executa todo o processo de execução da simulação e captura dos dados de estimativa, apresentando os resultados após a parada da simulação. O critério de parada da simulação será descrito no capítulo 5.

A plataforma de *hardware* adotada para aplicações dos conceitos desenvolvidos é baseada na família de microcontroladores 8051. Adotou-se particularmente o microcontrolador AT89S8252 da ATMEL. Este microcontrolador foi escolhido devido a sua vasta utilização no mercado de sistemas embarcados de baixo custo.

Nas próximas seções segue uma descrição de cada estágio que compõe o *framework*.

3.1 PLATAFORMA DE HARDWARE

As informações de consumo de energia e tempo das estruturas básicas são capturadas em um ambiente de medição que consiste em protótipos do sistema do qual se quer obter a estimativa.

No caso deste trabalho, a arquitetura escolhida foi AT89S8252. A Figura 3.2 mostra a descrição do protótipo para medição de consumo de energia.

Nesse protótipo, podem-se observar circuitos específicos como:

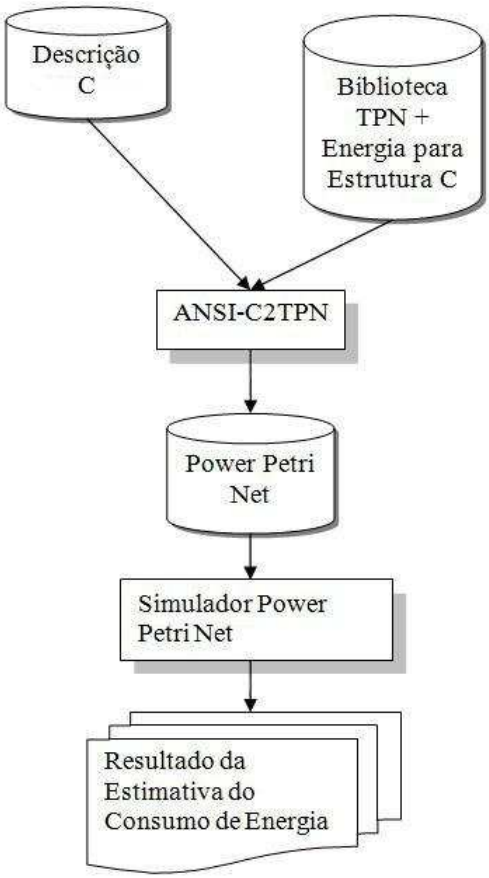


Figura 3.1 Visão Geral da Arquitetura da Power Petri Net

- Fonte de Energia: essa é a fonte de referência para o processo de medição e é através dela que se extrai a tensão de alimentação para detecção do consumo das estruturas básicas;
- Programador Serial: circuito responsável por prover as funcionalidades de programação do microcontrolador;
- Circuito de Medição: circuito responsável por prover os pontos para que o osciloscópio possa identificar o início e o término de uma determinada tarefa e assim prover o tempo e o consumo da estrutura básica;
- Circuito Oscilador: esse circuito é responsável por prover o relógio para funcionamento do microcontrolador;
- Microcontrolador: circuito integrado onde os programas estão sendo executados.

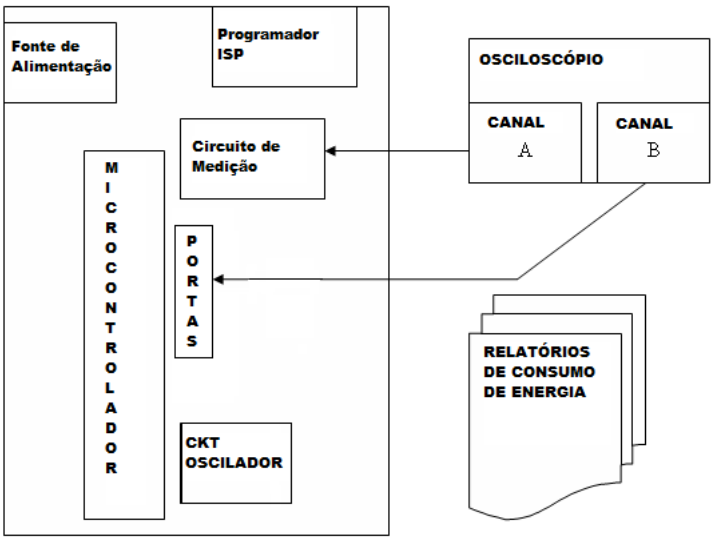


Figura 3.2 *Hardware* usado na Caracterização do Processador

Com a construção da plataforma de *hardware* obtêm-se uma biblioteca de consumo de energia para as estruturas do código ANSI-C para essa determinada arquitetura, apresentada logo a seguir.

3.2 BIBLIOTECA DE CONSUMO DE ENERGIA DOS MODELOS

Após a medição dos tempos de execução e do consumo de cada esturutra da linguagem ANSI-C, onde temos a biblioteca de consumo de energia pronta, segue o processo de construção de biblioteca de modelos de Power Petri Net. Essa biblioteca serve de entrada para o *parser* montar a Power Petri Net e consiste de um arquivo em XML com os valores de consumo das estruturas básicas do C.

Para a construção da biblioteca o compilador do código utilizado foi o C Keil-C51v7, por ser amplamente usado na comunidade de *softwares* embutidos. O compilador *Keil-C51v7* possui dez níveis de otimização em duas ênfases: extensão de código e tempo de execução, totalizando vinte opções de implementação de código binário. Para a construção da biblioteca e da realização da estimativa a otimização escolhida foi *Constant Folding (Favor Speed)*.

3.3 PARSER ANSI-C PARA TIMED PETRI NET

Este módulo consiste de um programa de identificação das estruturas do ANSI-C e construção das entidades JAVA que representará a Power Petri NET. O *parser* reconhece as estruturas para possibilitar a construção da rede final, que é o primeiro passo do compilador formado.

Como artefato de entrada, tem-se a descrição ANSI-C objeto da estimativa e a Biblioteca de modelos.

O artefato resultado dessa atividade é um arquivo que consiste na Rede de Petri Temporizada em formato PNML [33]. PNML é um formato padrão para ferramentas de Redes de Petri, para troca de informações.

Esse *parser* foi construído baseado em JavaCC (Java Compiler Compiler)[34] que é um gerador de *parsers*, código aberto, de Linguagem de Programação JAVA.

3.4 SIMULADOR

Com a Power Petri Net construída, o próximo passo é a simulação. Para a simulação ser realizada, é necessária a adição de mais uma informação, que diz respeito aos cenários de execução a serem avaliados. O cenário é definido através da definição no modelo das probabilidades de execução associadas às estruturas de decisão contidas no código. No modelo, estas probabilidades são anotadas às transições em conflito.

Após a definição do cenário, a simulação pode ser iniciada.

A simulação é finalizada, quando o critério de parada (intervalo de confiança) associado à métrica de interesse é alcançado. Esse critério de parada é detalhado no capítulo 5.

3.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma visão geral de todo o *framework* desenvolvido, bem como definiu todas as atividades contidas no desenvolvimento de todos os artefatos.

Neste ponto foram apresentadas todas os artefatos gerados pelo *framework* e atividades realizadas após cada etapa. No restante da dissertação serão apresentadas as formalizações e detalhes sobre o trabalho.

CAPÍTULO 4

MODELO POWER PETRI NET

Neste capítulo serão apresentados os modelos Power Petri Net para as estruturas básicas do ANSI-C.

4.1 A LINGUAGEM ANSI-C

Dado que a linguagem ANSI-C é uma linguagem amplamente utilizada para implementação de sistemas embarcados, foi escolhida como linguagem deste estudo. Segundo Kernighan, C não é uma linguagem de alto nível como também não é uma linguagem de baixo nível. C permite o controle preciso das entradas e saídas dos sistemas e essa é uma das principais razões de ser tão difundida em desenvolvimento de Sistemas Embarcados.

O projeto de desenvolvimento de Sistemas embarcados não é muito diferente de projetos para desenvolvimento de computadores de mesa *desktops* [5]. A diferença mais significativa é que um programa para sistemas embarcados, geralmente, precisa ser estável para manipulação direta de dispositivos de *hardware* e não tem o seu ambiente dividido com outros programas. Então, a presença de variáveis globais e rotinas de inicialização de ambiente é bem comum.

É comum também a presença de um um laço principal de controle para que o programa continue executando sem finalizar. Outro aspecto importante a ser mencionado é que o projetista, normalmente, deve possuir conhecimento em *hardware* e dispositivos mecânicos.

O Instituto Nacional Americano de Padrões (*American National Standards Institute*) definiu um padrão para linguagens C. Esta linguagem foi denominada ANSI C e se tornou a versão padrão da linguagem C. Além das funcionalidades da versão antiga, ANSI C incorporou características diferenciadas, tais como a forma de declarar funções. O objetivo original da formação do padrão ANSI foi portabilidade.

A primeira especificação ANSI-C foi lançada em 1990, em um documento formal chamado ANSI $X3.159 - 1989$. Esse padrão também foi adotado pela Organização Internacional de Padrões (*International Standards Organization - ISO*) como ISO/TEC

9899 – 1990.

O código ANSI-C é formado de estruturas básicas, como Atribuições, Operações Aritméticas e Operações Lógicas que foram modeladas com o objetivo de simular e, conseqüentemente, estimar o consumo de energia. Nas próximas seções, todas essas estruturas básicas serão descritas e as Redes de Petri temporizadas apresentadas.

Dentro do escopo do trabalho, as estruturas básicas modeladas foram: (i) Atribuição, Operações Aritméticas e Operações Lógicas; (ii) Condicionais; (iii) Switch; (iv) Interação com FOR; (v) Interação com *While*; e (vi) Chamada de Função.

4.2 O MODELO POWER PETRI NET PARA ANSI-C

Antes de iniciar a descrição do modelo para as estruturas da linguagem, é necessário apresentar a definição da rede Power Petri Net.

Seja $Nt = (N, D, C)$ uma rede de Petri temporizada (ver Seção 2.4), $PTPN = (Nt, Pw)$ é definida como uma rede temporizada com anotações de energia, $Pw : T \rightarrow \mathbb{R}^+ \cup \{0\}$ é uma função que associa às transições um número real que representa o consumo de energia relativo à execução da ações representada pela transição.

Na definição acima, a Rede de Petri temporizada é estendida com informação de energia representada por Pw. A obtenção dos dados temporais e de consumo de energia de cada estrutura básica será demonstrada no capítulo sobre caracterização do processador, enquanto que a atribuição dos valores de $C : T \mapsto \Re$ uma função de probabilidade que mapeia transições em conflito da rede às suas probabilidades de disparo será atribuída no capítulo referente à simulação.

4.2.1 Atribuição, Operação Aritmética e Lógica

A estrutura denominada atribuição representa a cópia de valores, que podem ser constantes ou variáveis, em memória. Expressões aritméticas e Lógicas representam operações matemáticas cujo resultado é armazenado em memória. Essas três estruturas são as mais simples em ANSI-C. São representadas pela mesma estrutura, ou seja, uma transição, sua pré-condição e sua pós-condição.

A Figura 4.1 mostra a Rede de Petri referente ao padrão seqüencial. O Padrão formado por essas estruturas é denominado Padrão Seqüencial.

Seja a rede $N' = (P', T', I', O', D', C', Pw')$ que representa a estrutura seqüencial tal que:

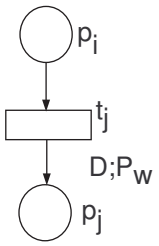


Figura 4.1 Padrão Seqüencial

- $P' = \{p_j, p_i\}$, onde p_j é o lugar de saída da transição que representa a instrução j e p_i é o lugar de entrada
- $T' = \{t_j\}$,
- $I' = \{(t_j, p_i)\}$,
- $O' = \{(t_j, p_j)\}$,
- $D'(t_j) = n, n \in \mathbb{R}^+ \cup \{0\}$,
- $C'(t_j) = 1$,
- $Pw'(t_j) = n, n \in \mathbb{R}^+ \cup \{0\}$.

4.2.2 Comando Condicional

O comando condicional é formado por uma expressão lógica denominada expressão de teste. Com isso, dois caminhos são possíveis, o primeiro se o resultado da expressão for positivo e o segundo se o resultado da expressão for negativo.

Um exemplo de código ANSI C de um comando condicional segue abaixo:

```
If(Condition) {
   True Body
} else {
   False Body
}
```

A Figura 4.2 mostra a Rede de Petri desse comando. Esse comando pode ser representado por um padrão, chamado Padrão Condicional, dividido em duas partes. A

primeira parte, o início, é representado por um lugar de entrada conectado a uma ou mais expressões de teste. Essa primeira parte é chamada Ponto de Entrada da Escolha. A segunda parte é o padrão usado para juntar os dois caminhos, no final do comando, também chamado de Ponto de Saída de uma escolha. O Padrão de Saída é formado pela ligação das duas últimas transições de cada caminho em um lugar de saída.

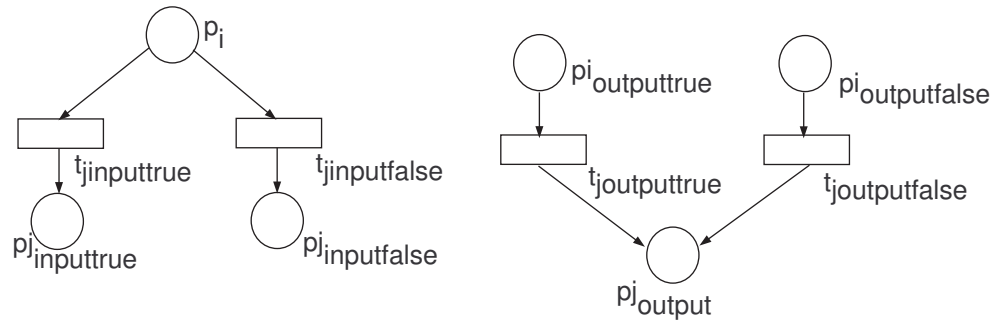


Figura 4.2 Representação em Rede de Petri do Padrão Condicional.

Seja a rede $N' = (P', T', I', O', D', C', Pw')$ que representa o ponto de entrada estrutura condicional tal que:

- $P' = \{p'j, p_jinputtrue, p_jinputfalse\}$, onde $p_jinputtrue$ e $p_jinputfalse$ são os lugares de saída do ponto de entrada da escolha e $p'j$ o lugar de entrada
- $T' = \{t_jinputtrue, t_jinputfalse\}$,
- $I' = \{(t_jinputtrue, p_i), (t_jinputfalse, p_i)\}$,
- $O' = \{(t_jinputtrue, p_jinputtrue), (t_jinputfalse, p_jinputfalse)\}$,
- $D'(t_jinputtrue) = n$, $D'(t_jinputfalse) = n, n \in \mathbb{R}^+ \cup \{0\}$,
- $C'(t_jinputtrue) + C'(t_jinputfalse) = 1$,
- $Pw'(t_jinputtrue) = n$, $Pw'(t_jinputfalse) = n, n \in \mathbb{R}^+ \cup \{0\}$.

Para o ponto de saída do comando condicional tem-se a rede $N' = (P', T', I', O', D', C', Pw')$ que representa o ponto de saída estrutura condicional tal que:

- $P' = \{p_i, p_j, p_k\}$, onde p_i e p_j são lugares de entrada e pk o lugar de saída
- $T' = \{t_k outputtrue, t_k outputfalse\}$,
- $I' = I \cup \{(t_k outputtrue, p_i), (t_k outputfalse, p_j)\}$,
- $O' = O \cup \{(t_k outputtrue, p_k), (t_k outputfalse, p_k)\}$,
- $D'(t_k outputtrue) = n$, $D'(t_k outputfalse) = n, n \in \mathbb{R}^+ \cup \{0\}$,
- $C'(t_k outputtrue) = 1$, $C'(t_k outputtrue) = 1$,
- $Pw'(t_k outputtrue) = n$, $Pw'(t_k outputfalse) = n, n \in \mathbb{R}^+ \cup \{0\}$.

4.2.3 Switch

Switch é um dos comandos de escolha formado por uma expressão de teste e várias condições, chamadas de *case*. A montagem da rede dessa estrutura é bem semelhante à montagem do condicional, porém, tem uma particularidade que é a ocorrência, ou não, de um *break*. Cada condição é formada por um lugar de entrada, o código interno, como o corpo do condicional, e o lugar de saída. Se o *break* ocorrer, esse lugar de saída é conectado à saída final do condicional, no caso, o Padrão Ponto de Saída. Se ele não ocorrer, o lugar de saída é conectado ao corpo do próximo condicional. Esse comportamento pode ser melhor observado na Figura 4.3 e no código a seguir. Nota-se que, no caso, não tem um *break*, Case2, usa-se uma transição auxiliar para ligar o seu corpo ao corpo do próximo case, no caso o *default*.

```
   Switch (expression):
{
      Case1 const-expr:
          BodyStatements;
             Break;
      Case2 const-expr:
          BodyStatements;
      Default:
        DefaultBody;
             Break;
}
```

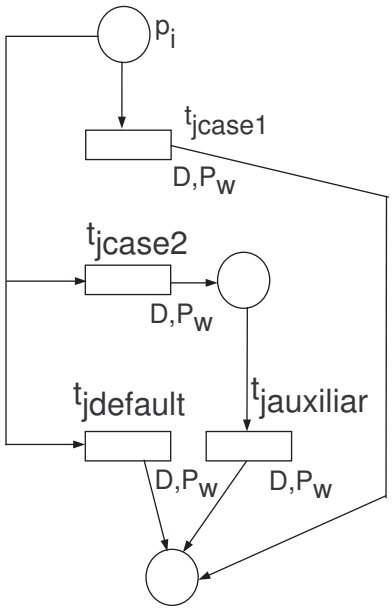


Figura 4.3 Exemplo da Rede de Petri do Switch

Sua formação é toda baseada na utilização do Padrão condicional, com a utilização da transição auxiliar e de uma transição para cada condição.

Para melhor entendimento tomamos a primeira parte do *Switch* como um padrão condicional e cada *case* como um ponto de saída da Rede de Petri, representado por uma transição.

Sendo assim, são definidos o ponto de entrada da escolha e o ponto de saída.

Seja a rede $N' = (P', T', I', O', D', C', Pw')$ que representa a estrutura do switch:

- $P' = \{p_j inputswitch, p_j input1, ..., p_j inputN, p_j inputdefault\}$, onde $p_j inputswitch$ é o lugar de saída do cálculo da expressão da escolha, $p_j input1, ..., p_j inputN$ são os lugares de saída do ponto de entrada da escolha que serão utilizados como lugares de entrada das estruturas que estão dentro de cada ramo do construtor

- $T' = \{t_j inputswitch, t_j input1, ..., t_j inputN, t_j inputdefault\}$, onde $t_j inputswitch$ representa o cálculo da expressão de escolha

- $I' = \{(t_j inputswitch, p_i), (t_j input1, p_j inputswitch), ..., (t_j inputN, p_j inputswitch), (t_j inputdefault, p_j inputswitch)\}$,

- $O' = \{(t_j inputswitch, p_j inputswitch), (t_j inputswitch, p_j input1),$

$..., (t_j inputswitch, p_j inputN), (t_j inputswitch, p_j inputdefault)\}$,

- $D'(t_j inputswitch) = n$, $D'(t_j input1) = n, ..., D'(t_j inputN) = n$, $D'(t_j inputdefault) = n, n \in \mathbb{R}^+ \cup \{0\}$,
- $C'(t_j inputswitch) = n1, C'(t_j input1) = n2$, $..., C'(t_j inputN) = nn, C'(t_j inputdefault) = nn + 1\}$, $onde \sum ni = 1$,
- $Pw'(t_j inputswitch) = n, Pw'(t_j input1) = n$, $..., Pw'(t_j inputN) = nePw'(t_j inputdefault) = n, n \in \mathbb{R}^+ \cup \{0\}$.

4.2.4 For

A próxima estrutura ANSI-C apresentada é a FOR. FOR é um comando de iteração dividido nas seguintes partes básicas: Comandos de Inicialização, Expressão Condicional de Teste, Comandos de Pós Execução e Corpo de Execução. A estrutura do código ANSI-C pode ser visualizado abaixo:

```
for(comandos de inicialização;
    Expressão condicional de testes;
    comandos de Pós Execução)
{
  Corpo de Execução.
}
```

A estrutura da iteração FOR é formada pela rede da Figura 4.4. Seja a rede $N' = (P', T', I', O', D', C', Pw')$ que representa a estrutura do FOR:

- $P' = \{p_i initializationCode, p_i nputConditionalTest, p_j condinput, p_j true, p_j false\}$, onde $p_i initializationCode$ é o lugar de entrada para o corpo da inicialização do For e $p_i nputConditionalTest$ é o lugar de entrada para o corpo da for. $p_j postexecution$ é o lugar de saída do for para o início de uma nova iteração.
- $T' = \{t_j initializationCode, t_j conditionaltest, t_j inputtrue, t_j inputfalse, t_j postexecution\}$, onde $t_j initializationCode$ representa a transição após o código de inicialização e $t_j conditionaltest$ representa a transição de entrada no corpo do FOR.

- $I' = \{(t_j initializationCode, p_i nitializationCode), (t_j conditionaltest, p_i nputConditionalTest), (t_j inputtrue, p_j condinput), (t_j inputfalse, p_j condinput), (t_j postexecution, p_j true)\}$,
- $O' = \{(t_j initializationCode, p_i nputConditionalTest), (t_j conditionaltest, p_j condinput), (t_j inputtrue, p_j true), (t_j inputfalse, p_j false), (t_j postexecution, p_i nputConditionalTest)\}$,
- $D'(t_j initializationCode) = n$, $D'(t_j conditionaltest) = n$, $D'(t_j inputtrue) = n$, $D'(t_j inputfalse) = n, D'(t_j postexecution) = n, n \in \mathbb{R}^+ \cup \{0\}$,
- $C'(t_j initializationCode) = 1$, $C'(t_j conditionaltest) = 1$, $C'(t_j postexecution) = 1$, $C'(t_j inputtrue) = n1$, $C'(t_j inputfalse) = n2$, $onde \sum ni = 1$,
- $Pw'(t_j initializationCode) = n$, $Pw'(t_j conditionaltest) = n$, $Pw'(t_j inputtrue) = n$, $Pw'(t_j inputfalse) = n$, $Pw'(t_j postexecution) = n$, $n \in \mathbb{R}^+ \cup \{0\}$.

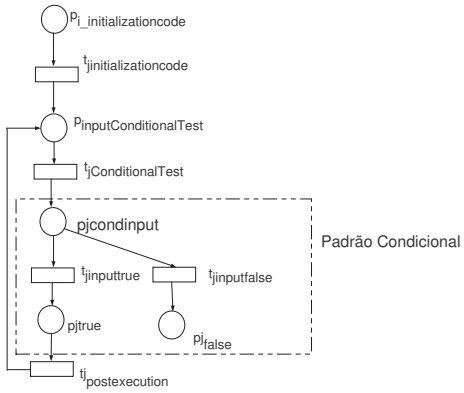


Figura 4.4 Exemplo da estrutura de Iteração FOR, composta pelos padrões Seqüencial e Condicional

4.2.5 While e Do-While

Outro tipo de estrutura de iteração é o *While* e o *Do-While*. Essa estrutura é formada por um código de testes (Padrão Condicional), um corpo de execução para quando o resultado do teste for positivo, e um lugar de saída para quando o resultado do teste for negativo. A diferença do *While* para o *Do-While* é que, no *Do-While*, o corpo de execução

é executado pelo menos uma vez antes do primeiro teste e, no caso do *While*, o teste é executado antes de qualquer execução do corpo.

A estrutura ANSI-C do *Do-While* pode ser observada a seguir:

```
Do-While: Do {
  Corpo do While.
} While(condição);
```

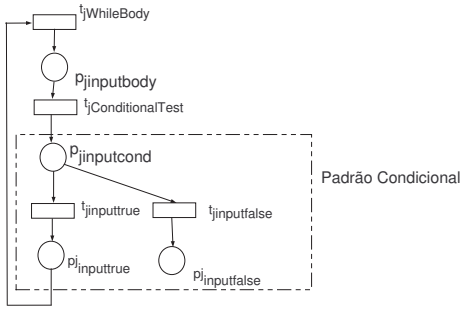


Figura 4.5 Rede de Petri da Iteração Do-While

Seja a rede $N' = (P', T', I', O', D', C', Pw')$ que representa a estrutura do *Do-While*:

- $P' = \{p_jinputbody,\ p_jinputcond,\ p_jinputtrue,\ p_jinputfalse\}$,
- $T' = \{t_jwhilebody,\ t_jconditionaltest,\ t_jinputtrue,\ t_jinputfalse\}$,
- $I' = \{(t_jwhilebody,\ p_jinputtrue),\ (t_jconditionaltest,\ p_jinputbody),\ (t_jinputtrue,\ p_jinputcond),\ (t_jinputfalse,\ p_jinputcond)\}$,
- $O' = \{(t_jwhilebody,\ p_jinputbody),\ (t_jconditionaltest,\ p_jinputcond),\ (t_jinputtrue,\ p_jinputtrue),\ (t_jinputfalse,\ p_jinputfalse)\}$,
- $D'(t_jwhilebody) = n,\ D'(t_jconditionaltest) = n,\ D'(t_jinputtrue) = n,\ D'(t_jinputfalse) = n,\ n \in \mathbb{R}^+ \cup \{0\}$,
- $C'(t_jwhilebody) = 1,\ C'(t_jconditionaltest) = 1,\ C'(t_jinputtrue) = n1,\ C'(t_jinputfalse) = n2,\ onde \sum ni = 1$,
- $Pw'(t_jwhilebody) = n,\ Pw'(t_jconditionaltest) = n,\ Pw'(t_jinputtrue) = n,\ Pw'(t_jinputfalse) = n,\ n \in \mathbb{R}^+ \cup \{0\}$.

A estrutura ANSI-C do *While-Do* pode ser observada abaixo:

```
While-Do: While(condição) {
   Corpo do while.
}
```

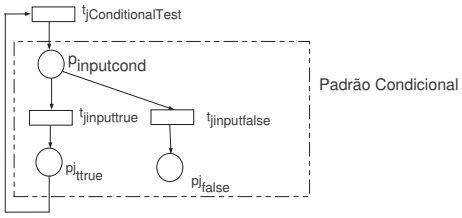


Figura 4.6 Rede de Petri da Iteração *While*

Seja a rede $N' = (P', T', I', O', D', C', Pw')$ que representa a estrutura do *While-Do*:

- $P' = \{p_j inputcond, p_j inputtrue, p_j inputfalse\}$,
- $T' = \{t_j conditionaltest, t_j inputtrue, t_j inputfalse\}$,
- $I' = \{(t_j conditionaltest, p_j true), (t_j inputtrue, p_j inputcond), (t_j inputfalse, p_j inputcond)\}$,
- $O' = \{(t_j conditionaltest, p_j inputcond), (t_j inputtrue, p_j inputtrue), (t_j inputfalse, p_j inputfalse)\}$,
- $D'(t_j conditionaltest) = n$, $D'(t_j inputtrue) = n$, $D'(t_j inputfalse) = n$, $n \in \mathbb{R}^+ \cup \{0\}$,
- $C'(t_j conditionaltest) = 1$, $C'(t_j inputtrue) = n1$, $C'(t_j inputfalse) = n2$, $onde \sum ni = 1$,
- $Pw'(t_j conditionaltest) = n$, $Pw'(t_j inputtrue) = n$, $Pw'(t_j inputfalse) = n$, $n \in \mathbb{R}^+ \cup \{0\}$.

4.2.6 Chamada de Função

A última estrutura ANSI-C apresentada neste trabalho é a chamada de função. Essa representação mostra o desvio que ocorre quando uma função é chamada através de

um novo padrão denominado Padrão de Chamada de Função. Semelhante ao Padrão Condicional, o de chamada de função também é formado por duas partes, um chamado Ponto de Entrada da chamada de função e o outro Ponto de Saída da chamada de função.

A estrutura ANSI-C da chamada de função é apresentada abaixo:

```
void main () {
  int x,y;
  y=3;
  functionExample();
  x=y;
}

void functionExample() {
    int testVariable;
    testVariable=2;
}
```

A Rede de Petri equivalente apresenta o desvio do fluxo principal do programa utilizando a divisão dos *Tokens*. Um *token* é gerado no lugar de entrada da função e no lugar posterior à estrutura chamada antes da função. Com isso, no ponto de retorno, temos como entrada da transição dois lugares e, assim, o fluxo só volta a ser executado quando o *token* chega no final da função. Essa representação pode ser observada na Figura 4.7.

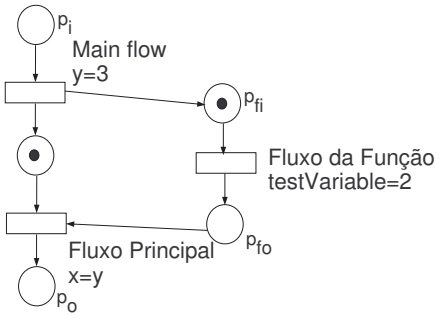


Figura 4.7 Rede de Petri para uma chamada de função

Como mencionado anteriormente, a chamada de função cria mais dois novos padrões, o Ponto de Entrada da função e o Ponto de Saída da função.

Seja a rede $N' = (P', T', I', O', D', C', Pw')$ que representa o Padrão de Chamada de Função - Ponto de Entrada de função:

- $P' = \{p_i mainflow, p_j inputfunction\}$,
- $T' = \{t_i beforefunctioncall\}$,
- $I' = \emptyset$,
- $O' = \{(t_i beforefunctioncall, p_i mainflow), (t_i beforefunctioncall, p_j inputfunction)\}$,
- $D'(t_i beforefunctioncall) = n$, $n \in \mathbb{R}^+ \cup \{0\}$,
- $C'(t_i beforefunctioncall) = 1$,
- $Pw'(t_i beforefunctioncall) = n$, $n \in \mathbb{R}^+ \cup \{0\}$.

Seja a rede $N' = (P', T', I', O', D', C', Pw')$ que representa o Padrão de Chamada de Função - Ponto de Saída de função:

- $P' = \{p_i mainflow, p_j outputfunction\}$,
- $T' = \{t_t afterfunctioncall\}$,
- $I' = \{(t_t afterfunctioncall, p_i mainflow), (t_t afterfunctioncall, p_j outputfunction)\}$,
- $O' = \emptyset$,
- $D'(t_t afterfunctioncall) = n$, $n \in \mathbb{R}^+ \cup \{0\}$,
- $C'(t_t afterfunctioncall) = 1$,
- $Pw'(t_t afterfunctioncall) = n$, $n \in \mathbb{R}^+ \cup \{0\}$,

Como conseqüência do Ponto de Entrada, tem-se um *token* gerado no lugar posterior à transição antes do ponto de entrada e um *token* gerado na rede que representa o interior da função.

Através da composição dos dois padrões apresentados anteriormente observa-se que o *token* gerado após o Ponto de Entrada da Função fica bloqueado pelo último lugar do Ponto de Saída da Função.

A Figura 4.8 representa o padrão de chamada de Função.

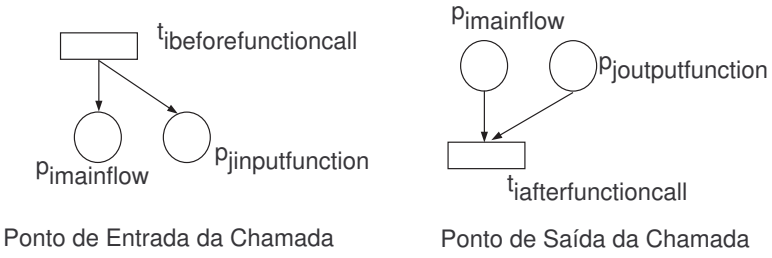


Figura 4.8 Rede de Petri com o Padrão de Chamada de Função

4.3 COMPOSIÇÃO DAS ESTRUTURAS

A seguir tem-se um exemplo prático da composição das estruturas básicas e da formação da Power Petri Net. Esse exemplo é formado por um laço principal, comandos de atribuição, expressões lógicas e estruturas condicionais.

verbatim

```
void main() {
   int a,b;
   while(true)
   {
      b = 5;
      a = random();
      if(a < b)
      {
          a=0;
      }
      else
      {
          a=1;
      }
   }
}
```

A Power Petri Net Formada a partir do exemplo acima é apresentada na Figura 4.9 .

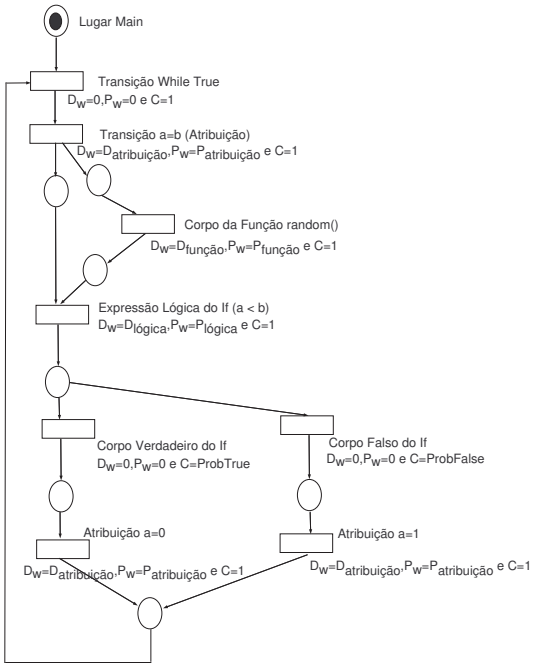


Figura 4.9 Exemplo da Composição das Estruturas Básicas

4.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os modelos Power Petri Net para estruturas ANSI-C. Estes modelos são as estruturas básicas para composição e geração do modelos das aplicações. São, portanto, a base para o processo de simulação. Antes da simulação, contudo, o modelo da aplicação (gerado dos modelos básicos apresentados) passa por um processo de inserção de informação de tempo de execução de cada estrutura, consumo de energia e probabilidades.

CAPÍTULO 5

SIMULAÇÃO

Este capítulo apresenta o ambiente de simulação da Power Petri Net. Primeiramente é mostrada a introdução dos valores de informação de probabilidade usados na simulação e depois é mostrado o passo a passo da implementação do simulador.

5.1 ATRIBUINDO VALORES DE PROBABILIDADE ÀS TRANSIÇÕES EM CONFLITO

A primeira etapa no processo de avaliação (simulação) do modelo consiste na identificação e definição dos cenários relevantes considerados durante o processo. Estes cenários são definidos pela atribuição de intervalos de probabilidades associados aos pontos de decisão da aplicação (representados no modelo por transições conflitantes), ou seja, probabilidades das estruturas de decisão.

Como uma regra geral, toda transição na Power Petri Net tem uma probabilidade de disparo igual a 1, a exceção das transições em conflito. Neste caso, é necessária a atribuição de probabilidades à elas e o somatório total das probabilidades das transições envolvidas tem que ser igual a 1. Na abordagem adotada, essas probabilidades são atribuídas pelo projetista.

5.2 PROCESSO DE SIMULAÇÃO

O processo de simulação pode ser dividido em passos. O 1º passo é a definição de uma marcação inicial da rede. O lugar que é marcado representa o início da execução do sistema representado em ANSI-C, ou seja, a função principal chamada de função *main*. Cada *token* colocada no lugar inicial representa uma unidade de processamento.

Seja $N = (P, T, I, O, D, Pw, C)$ uma Power Petri Net. Seja ainda K igual ao número de processadores existentes na plataforma. A marcação inicial da rede $M_0 : P \rightarrow \mathbb{N}$ é tal que $M_0(p_0) = K$ e $\forall p \in P, p \neq p_0, M(p) = 0$. O número de processadores, neste trabalho, só representa a capacidade de simulação de várias redes ao mesmo tempo, ou seja, vários

microcontroladores em um mesmo sistema embarcado, porém, sem comunicação entre eles.

Para que o processo de simulação possa ser executado, algumas informações adicionais são necessárias. As informações necessárias para o funcionamento do engenho de simulação consistem, basicamente, de três atributos: número mínimo, o máximo e o número atual de disparo de cada transição. Eles são adicionados como extensão à Power Petri Net.

Essas informações são utilizadas para o critério de parada da simulação que inclui o intervalo de confiança e o máximo permitido, além de serem utilizadas no *warm-up* do simulador. Antes de iniciar a simulação, essas informações são atribuídas às transições, bem como a todos os contadores utilizados na simulação têm seu valores iniciais igualados a zero.

Contadores globais, que armazenam o tempo total, e o consumo de energia total também são zerados. As transições inicialmente habilitadas são definidas através de uma função que calcula, pela marcação inicial, as próximas transições a serem disparadas. Essas transições, se não estiverem em conflito, consomem o *token* e assim o processo de simulação se inicia. Os contadores globais são incrementados a cada passo da simulação, como vai ser formalizado posteriormente.

Dada $N = (P, T, I, O, D, C, Pw)$ uma Power Petri Net, a estrutura $AS = (n_m, n_M, n_t, n_T, e_D, e_{Pw})$, onde $n_m, n_M, n_T \in \mathbb{N}$, $n_t : T \to \mathbb{N}$ e $e_D, e_{Pw} \in \mathbb{R}$ contém os atributos de simulação para a SPTPN. Os valores $0 < n_m \leq n_M$ são os números mínimo e máximo globais de disparos de transições, respectivamente. A variável n_T registra o número total de disparos de transições, de forma global. A função n_t mapeia cada transição no número de seus disparos executados durante a simulação. As variáveis e_D e e_{Pw} acumulam o tempo e energia totais, respectivamente.

Sejam $N = (P, T, I, O, D, C, Pw)$ uma Power Petri Net e $M_0 : P \to \mathbb{N}$ uma marcação inicial, $M = M_0$ uma marcação e $AS = (n_m, n_M, n_t, n_T)$ uma estrutura de atributos de simulação, onde n_m e n_M são valores informados pelo projetista, $n_T = 0$, $\forall t \in T, n_T(t) = 0$ e $e_D = e_{Pw} = 0$. Portanto, a estrutura necessária para simulação é definida pela tupla $N_S = (N; M; AS)$.

A implementação atual é tal que $n_m = 10\% n_M$, de forma que o projetista só precisa fornecer o valor de n_M. Esses 10% foi definido empiricamente, considerando que esse valor fosse suficiente para que o erro mínimo fosse atingido pela simulação.

Considerando a estrutura $N_S = (N; M; AS)$ uma rede de simulação, seja ainda T_d o

conjunto de transições disparadas de N para uma dada marcação M que levaram a rede a uma marcação M', o novo estado AS' da estrutura de informações de simulação é tal que:

- $n'_m = n_m$
- $n'_M = n_M$
- $n_t(t)' = n_t(t) + 1, \forall i \in T_d$
- $n'_T = n_t + \sharp T_d$, onde $\sharp$ denota a cardinalidade do conjunto
- $e'_D = e_d + \sum t \in T_d D(t)$
- $e'_{Pw} = e_{Pw} + \sum t \in T_d e_{Pw}(t)$

Os componentes M' e AS' representam o novo estado da estrutura N' após um passo na simulação.

Depois que os contadores de disparo mínimo atingirem o seu valor, as transições que são parte do conjunto dos conflitos, ou seja, transições que têm a mesma pré-condição, recebem uma atenção especial. Este tempo inicial é definido pelo critério de *warmup* da simulação estocástica. A proporção dos disparos é calculada tendo o número de disparos contados com a probabilidade atribuída a cada transição em conflito.

No processo atualmente implementado, o período de *warm-up* perdura enquanto $n_T < n_m$.

Os principais critérios de parada para a simulação são o erro e o intervalo de confiança que se deseja para estimativa. A freqüência relativa de disparo das transições, a longo prazo (dado que o modelo é estacionário), converge para as probabilidades atribuídas às transições em conflito. Quando todas as freqüências relativas associadas aos grupos de transições em conflito convergirem para os valores atribuídos, a simulação é finalizada.

Após definir um passo de simulação, faz-se necessária a formalização do critério de parada da simulação. Este critério de parada é dado a partir de um erro e intervalo de confiança pré-estabelecido para todos os conjuntos de transições em conflito.

Considerando que a estrutura $N_S = (N; M; AS)$ não mais esteja no período de *warm-up*, isto é, $n_T \geq n_m$, e sendo ϵ definido como o erro máximo admitido para a simulação, o critério de parada para a simulação da rede é definido por:

$E = t^* \frac{S}{\sqrt{n}}$,

n é o número de amostras,

s é o desvio padrão da média (métrica de interesse),

t^* é o valor crítico de t que leva em consideração o nível de confiança $(1 - \alpha)$ e o número de amostras (n), e

E é o erro.

n é considerado o número de rodadas. Uma rodada consiste em executar a aplicação do seu estado inicial até que se atinja o estado final. Neste estado final, o desvio padrão (s) é calculado, dado que se tem o t^* e o n (até aquele ponto) verifica-se se o erro é menor que o máximo permitido. Caso não seja, o processo prossegue. Quando o critério for alcançado, pára-se a simulação.

Este é um processo simplificado de avaliação do critério de parada de uma simulação estacionária que leva em consideração o erro e o intervalo de confiança (clássico - paramétrico).

Ao ser interrompida por algum dos critérios exibidos anteriormente, os dados de tempo de execução e energia estão armazenados em e_D e e_{Pw}

Outro critério de parada é quando a quantidade máxima de disparo das transições é atingida. Isso significa que a simulação foi executada até uma quantidade máxima pré-estabelecida e, até aquele momento, todos os conjuntos de transições em conflito não convergiram. Neste caso, as métricas fornecidas não são confiáveis, pois o nível de significância não foi alcançado ou o erro médio associado à estatística é maior que o erro máximo permitido.

Finalmente, após a conclusão da simulação, a potência total é calculada e o consumo de energia é estimado. Essa estimativa final é calculada considerando a energia total calculada na simulação e o tempo de duração total da execução que foi armazenado em e_D. Desta maneira, a potência estimada do sistema é calculada e, por conseqüência, o consumo de energia do período determinado.

5.3 VISÃO GERAL DA IMPLEMENTAÇÃO

O ambiente computacional implementado está dividido em duas partes: *Parser* (*ANSI-C*) para estruturas PNML [1] e Simulador para Power Petri Net.

O *parser* foi construído para traduzir as estruturas básicas do ANSI-C em Power Petri Net(formato PNML [1]). A implementação seguiu o padrão de desenvolvimento apresentado na Figura 5.1.

Como pode ser observado, a implementação segue o padrão de desenvolvimento MVC - *Model view Control*. A parte de visualização (*view*) é representada pelas estruturas básicas do ambiente java (Popup Menus, Editores, Visualizadores, Ferramentas). O Controle está representado pela implementação da funcionalidade do eclipse de ação, por exemplo, o clicar do mouse em um item do menu, ou em cima de um arquivo, comanda-se a inicialização da execução do código do *parser*. E por fim, a camada do modelo é implementada pela utilização do JavaCC, Java Compiler Compiler, que é um *software* de geração de *parsers* através de bibliotecas de formação de linguagens e de objetos Java. Toda essa infraestrutura foi desenvolvida usando o *framework* de geração de PNML, XML usado para interação entre ferramentas de Redes de Petri.

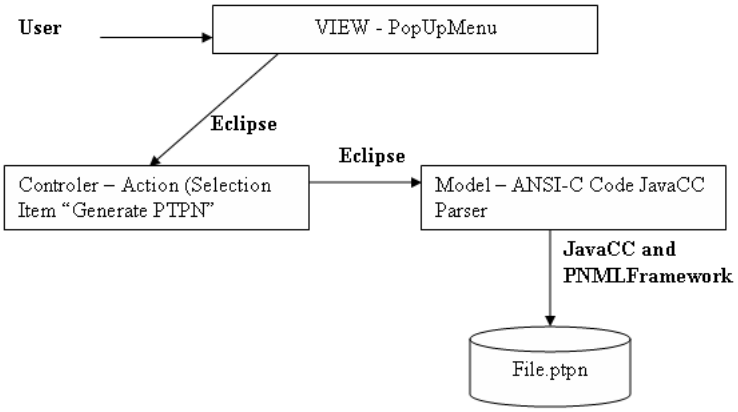


Figura 5.1 Arquitetura do *Parser* de ANSI-C para PTPN

É importante ressaltar a biblioteca usada pelo JavaCC com as regras de formação da estruturas ANSI-C. Neste arquivo que define as regras de formação, tem-se a estrutura em Java que faz a montagem do arquivo PNML. Essas camadas do *parser* podem ser visualizados na Figura 5.2.

O PNML *Framework*, construído pela Université Pierre et Marie Curie, Move - Laboratoire d' informatique de Paris 6 (LIP6), foi desenvolvido para a construção de Redes de Petri através de uma especificação XML padrão, chamada PNML. Este *framework* serve para traduzir especificações PNML em entidade JAVA e assim servir tanto para salvar especificações Java em arquivos XML quanto para para ler esses arquivos. O objetivo principal da criação desse padrão é permitir uma interação entre ferramentas que manipulam Redes de Petri. Com a utilização desse *framework*, que é *open source* para pesquisas,

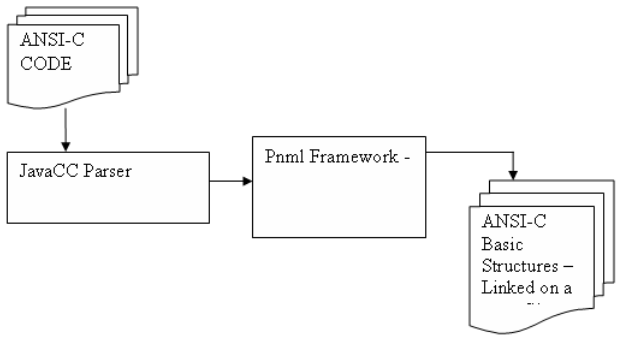


Figura 5.2 Modelo do Tradutor ANSI-C

foi possível salvar o modelo gerado pelo tradutor em PNML para posterior simulação. Desta forma, foi possível desenvolver duas ferramentas totalmente desconectadas e que podem ser reutilizadas futuramente.

A segunda parte do ambiente consiste na construção de um simulador da Power Petri Net, construído como um *Plug-in* para um ambiente de desenvolvimento Java, de acordo com o projeto demonstrado na Figura 5.3. Esse projeto também adota o padrão de desenvolvimento MVC e, como também foi desenvolvido baseando-se no mesmo ambiente, as camadas View e Control têm a mesma base de funcionamento explicada no tradutor. A camada de modelo foi construída através da leitura do PNML [1] e com a extensão de um engenho de simulação para Power Petri Net[35].

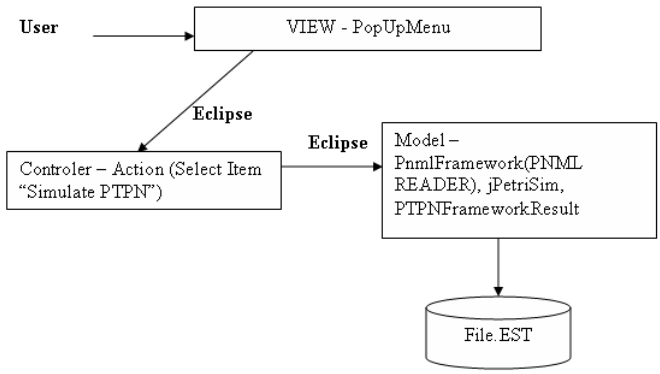


Figura 5.3 Simulador da Power Petri Net Estocástica - Plugin Schema

Através da extensão desse simulador de Redes de Petri Temporizadas, chamada jPetriSim [35], foi possível adaptá-lo para o modelo proposto por Zuberek [14] com as

devidas anotações de potência. Essa extensão foi denominada jPetriSimPTPN.

No simulador, assim como no *parser*, o PNML *framework* foi usado para abstrair a construção dos arquivos PNML, então, uma entidade do *framework*, chamada leitor de PNML foi usada para a geração de entidades Java a fim de referenciar Lugares, transições e arcos, e todas as informações em relação a tempo, consumo de energia, e probabilidades de execução das estruturas básicas ANSI-C.

Um módulo de geração de números aleatórios com distribuição uniforme também foi utilizado para o processo de decisão relativo ao disparo das transições que estão em conflito.

Sendo assim, tem-se a informação da probabilidade de execução em cada transição atribuída às transições em conflito. Essa probabilidade é comparada com a quantidade de vezes que cada transição foi disparada.

Uma vez que o critério de parada foi alcançado durante a simulação, seria muito caro, do ponto de vista de simulação, fazer comparação desde o seu início. Então, um critério de inicialização, *warmup*, foi definido para que as comparações só começassem a ser realizadas a partir do momento em que os dados começassem a ser significativos. Esse *warmup*, na versão inicial, foi definido empiricamente com o valor de 10% da quantidade máxima de simulação.

A Figura 5.4 apresenta detalhes sobre a implementação do simulador. A especificação em Redes de Petri, entidades java, são formadas a partir de tabelas com as informações para simulação, como lugares de entrada, lugares de saída, quantidade de vezes que a transição foi disparada, tempo de execução, probabilidade para transições em conflito, entre outras definidas anteriormente. Essas tabelas são atualizadas todas as vezes que o engenho de simulação é chamado a partir do disparo das transições, após o tempo do próximo evento. Para a geração dessas tabelas com informações para simulação, uma entidade foi criada, denominada Gerador de Estruturas Internas. Essa entidade é responsável pela atualização das tabelas.

O engenho de Redes de Petri temporizada analisa os *tokens* de entrada e transições habilitadas para gerar o resultados do passo de execução, que definirão os pré-requisitos para o próximo passo.

Após estes passos serem executados, um objeto de observação chamado Extrator de Dados de Desempenho, que implementa o critério de parada da simulação, verifica a necessidade de continuá-la e extrair dela todas as informações necessárias para a saída de estimativa. Exemplos desses dados são exatamento o tempo total que a simulação

levou e o consumo de energia total, dados esses que serão usados para o cálculo total da potência estimada do sistema, como foi apresentado anteriormente.

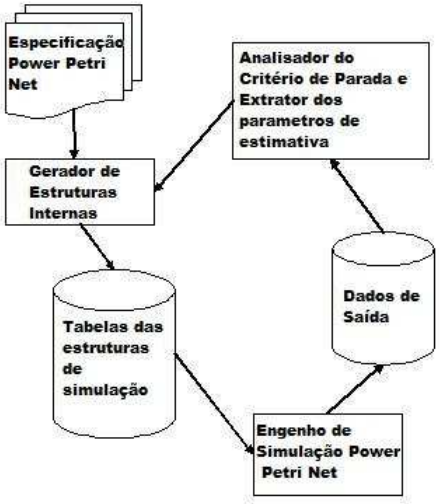


Figura 5.4 Arquitetura Interna do Simulador da Power Petri Net

Para a implementação do engenho foi utilizado o jPetriSim, detalhado no trabalho [35].

5.4 COMENTÁRIOS FINAIS

Este capítulo apresentou, inicialmente, os fundamentos necessários para o entendimento do engenho de simulação. Posteriormente, uma visão dos componentes do simulador da Power Petri Net foram apresentados.

CAPÍTULO 6

ESTUDOS DE CASO

Foram realizados experimentos para validar o processo de modelagem e os ambientes de avaliação propostos e os ambientes de avaliação foram usados para explorar os padrões comportamentais do código. Também foram realizados experimentos que ilustram casos típicos de códigos presentes em sistemas embutidos, e, em todos eles, a infra-estrutura de análise foi utilizada para a estimativa de consumo de energia, potência e tempo de execução em função dos cenários. Os resultados são apresentados através dos valores médios para cada cenário. Os algoritmos utilizados foram os desenvolvidos pela Motorola, e são chamados de *Power Stone Benchmark* [3].

Esses algoritmos têm uma particularidade de terem sido desenvolvidos especificamente com o objetivo de exercitar diferentes aspectos de consumo de energia de sistemas embarcados especificados em ANSI-C.

Os algoritmos utilizados foram os qurt, bcnt, crc e fir. Para um melhor entendimento, as próximas seções explicam cada experimento.

6.1 QURT

Qurt é um tipo de código para embaralhar dados em um *array* e, para isso, utiliza a função de raiz quadrada para calcular variações nesse *array*. Usando este código, os dados de entrada podem ser variados para se atingir diferentes valores de saída, pois isso funciona como um codificador de dados, usado em ambientes de transição de dados. Esse algoritmo utiliza vários *arrays* e operações aritméticas com apenas duas expressões condicionais. Devido à simplicidade do código, este exemplo foi utilizado para fazer a comparação com o valor simulado, usando o ambiente desenvolvido, e o valor medido na placa de prototipação. A Tabela 6.1 mostra o consumo de energia estimado variando a probabilidade das estruturas condicionais. Esse resultado pode ser observado também na Figura 6.1.

A Figura 6.2 mostra o corpo principal do código de chamada do qurt. Com este código pode-se observar que o experimento foi chamado três vezes durante o processo de

Figura 6.1 Gráfico da Estimativa de Consumo de Energia do Qurt

Tabela 6.1 Estimativa de Consumo do Qurt variando as duas expressões condicionais

Variação	Consumo Simulado(nJ)
1	5300.00 nJ
2	5299.99 nJ
3	5299.98 nJ
4	5300.01 nJ

medição.

6.2 BCNT

BCNT é um código que utiliza uma série de operações entre dois *arrays*, com operações de manipulação de espaço de memória, como operação de deslocamento. Como BCNT é outro código simples, em relação à quantidade de pontos de decisão, este exemplo foi utilizado para validação dos resultados de medição e da utilização de biblioteca de valores de tempo de execução de estruturas básicas e consumo de energia.

Como esse experimento não possui ponto de decisão, ou seja, não foi preciso definir nenhum valor de probabilidade, não precisou realizar nenhum experimento de variação de probabilidade, portanto, o valor de consumo estimado de energia foi de 578 nJ e o valor de consumo medido foi de 556 nJ.

```
void main( void )
{
        double a[3], x1[2], x2[2];
    int result;

    P1 = 255; /*Comeca P1 com o valor 1*/
    while(1)
        {
                //Abaixa o pino para Leitura
        P1 ^= 0x01;         /* Toggle P1.0*/

                //Colocar aqui o referido codigo

                  a[0] = 1.75;
                  a[1] = -3.2;
                  a[2] = 2.45;
                  qurt(a, x1, x2);
                  result = *(int *) &x1[0];
                  a[0] = 1.5;
                  a[1] = -2.5;
                  a[2] = 1.5;
                  qurt(a, x1, x2);
                  result += *(int *) &x1[1];
                  a[0] = 1.8;
                  a[1] = -4.275;
                  a[2] = 8.31;
                  qurt(a, x1, x2);
                  result -= *(int *) &x1[1];
                  if(result != -1776094907)
                    {
                      puts("qurt: fail\n");
                    }
                  else
                    {
                      puts("qurt: success\n");
                    }

                P1 ^= 0x01;
                //Temporizacao. Ajustar para melhor medir o codigo todo.
        variavelGeral=0;
        while(variavelGeral<10000000)
        {
            variavelGeral++;
                }
        }

}
```

Figura 6.2 Código Principal do Exemplo Qurt

6.3 CRC

CRC é um algoritmo para calcular *Cycle Redundant Checking*, ou seja, verificação do conteúdo de uma determinada série de dados. Algoritmos CRC são muito utilizados na verificação da recepção de dados em barramentos de comunicação influenciados por ruídos.

Este experimento foi utilizado variando os pontos de decisão para poder observar que, com esta técnica, pode-se chegar a diferentes tipos de cenários, variando a probabilidade das estruturas de decisão. Esse resultado pode ser visto na Tabela 6.2.

Tabela 6.2 Estimativa de Consumo do CRC com diferentes configurações

Configuração	Consumo Simulado(nJ)
1	15.3 nJ
2	12 nJ
3	13.7 nJ
4	19.7 nJ

Esse resultado pode ser observado também na Figura 6.3.

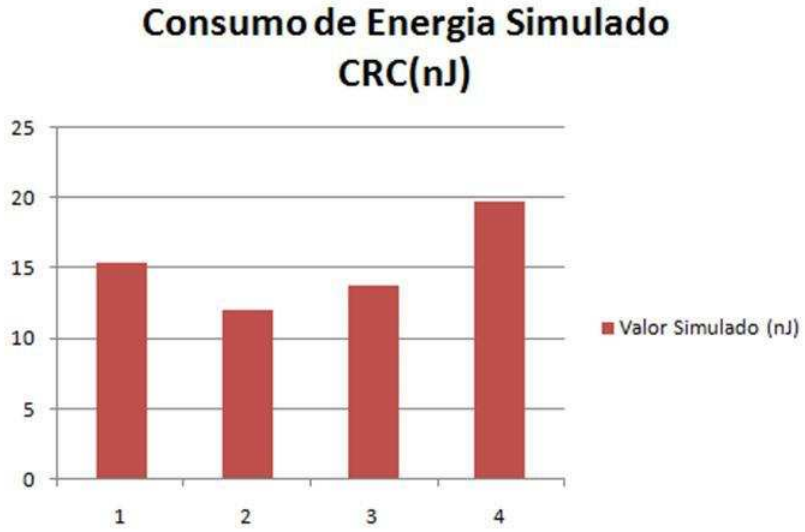


Figura 6.3 Gráfico da Estimativa de Consumo de Energia do CRC

6.4 FIR FILTER

O último estudo de caso utilizado foi um filtro, que é um algoritmo mais complexo do ponto de vista de quantidade de estruturas de processamento e também de quantidades de pontos de decisão. Com isso, pode-se validar a capacidade de variação de estimativa de consumo de energia em algoritmos mais complexos.

Este algoritmo utiliza geração de números aleatórios para execução e aplicação em um algoritmo de filtro. Esse filtro é um filtro usando o algoritmo de Gauss, muito usado em técnicas de processamento digital de sinais.

O resultado desse experimento pode ser observado na Tabela: 6.3.

Tabela 6.3 Estimativa de consumo do FIR Filter com diferentes configurações

Configuração	Consumo Simulado(nJ)
1	17.8 nJ
2	13.4 nJ
3	15.6 nJ
4	17.5 nJ

Esse resultado pode ser observado também na Figura 6.4.

Figura 6.4 Gráfico da Estimativa de Consumo de Energia do FIR

6.5 COMPARAÇÃO ENTRE OS DADOS ESTIMADOS E OS DADOS MEDIDOS

A Tabela 6.4 resume os resultados dos experimentos. Nesta Tabela, o campo Medido, representa o resultado da energia medida a partir dos experimentos para validação apresentado no processo de medição de consumo de energia e caracterização do processador. O campo Estimado consiste no resultado apresentado pelo simulador em relação ao consumo de energia simulado. Apresenta-se, também, o tamanho do código de máquina gerado pelo compilador de cada exemplo apresentado.

Tabela 6.4 Resumo do Resultado dos Experimentos

Experimento	Medido (nJ)	Estimado(nJ)
Qurt	5307,08	5300,02
BCNT	556	578
CRC	19,70	19,10
FIR	13,20	13,40

A comparação dos Resultados também pode ser observada na Figura 6.5.

Podemos afirmar que os dados não se distanciam de uma distribuição normal, dado que a repetição do processo de medição para cada uma dos experimentos provê resultados praticamente iguais aos apresentados. Pequenas variações são possíveis desde que se aumente significativamente a resolução dos mecanismos de medição, contudo os dados se

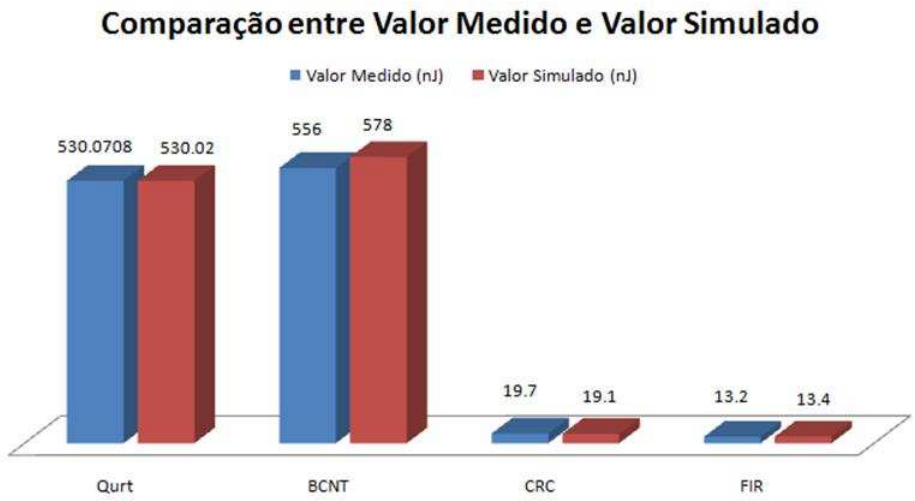


Figura 6.5 Gráfico da Comparação dos Resultados

distribuem de forma aproximadamente normal em torno dos valores médios apresentados na Tabela 6.4.

Desta forma, é possível aplicar o Teste T emparelhado e calcular o intervalo de confiança para diferença média de forma a que sejam encontradas evidências que refutem a igualdade entre modelo e sistema.

Tabela 6.5 Teste-T emparelhado e Intervalo de Confiança: Medido e Estimado

	N	Média	Desvio Padrão	SE Mean
Medido	4	1474,00	2568,00	1284,01
Estimado	4	1477,63	2561,98	1280,99
Diferença	4	-3,63500	12,66742	6,33371

95% Intervalo de Confiança para diferença média: $(-23,79169; 16,52169)$ Teste T para diferença media = 0 (vs não = 0): Valor T = $-0,57$ Valor P = $0,606$.

Dado que o valor 0 está incluso no intervalo de confiança calculado (-23,79169; 16,52169) não se têm evidências estatísticas para refutar a hipótese de que o modelo e o sistema sejam equivalentes.

Dado que a análise foi realizada para um nível de confiança de 95% e que o valor P = $0,606$ é maior que $(1 - 0,95)$=$0,05$, fica claro que a hipótese de igualdade entre modelo e sistema não pode ser refutada.

6.6 CONSIDERAÇÕES FINAIS

Os experimentos mostraram a aplicação do modelo e ambiente de análise na captura de informações cruciais às estratégias de otimização de sistemas embutidos. O modelo mostrou-se adequado a seu propósito possibilitou realizar estimativas precisas a partir de aplicações descritas em ANSI-C, possibilitando, em tempo de projeto, a tomada de decisão em relação à fonte de energia a ser utilizada no projeto que rodará um determinado código ANSI-C.

Todos os experimentos foram realizados no microcontrolador AT89S8252, o AT89S8252, que é um dispositivo CMOS, de 8 bits, de baixo consumo de energia com 8K bytes de memória *Flash* programável e 2K byte de memória EEPROM.

CAPÍTULO 7

CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo traz um resumo do trabalho apresentado, mostrando como os objetivos propostos foram atingidos, exibindo também quais os problemas encontrados durante o desenvolvimento do mesmo. A seguir, são apontadas as contribuições para a área de conhecimento Redes de Petri e Estimativa de Consumo de Energia. Finalmente, são apontadas possibilidades de melhorias no trabalho e novas linhas de investigação para trabalhos futuros dentro do mesmo tema.

O objetivo principal deste trabalho é apresentar um mecanismo estocástico (modelo e simulador estocástico) para análise do consumo de energia de aplicações, implementadas em linguagem C, de Sistemas Embarcados, de forma a prover ao projetista mecanismos para estimativa de potência em tempo de projeto.

Este trabalho apresentou um método baseado em Redes de Petri Temporizada que representa descrições ANSI-C e considera um processador de uma arquitetura específica.

O principal problema encontrado durante a realização do trabalho foi em relação ao processo de medição e caracterização do processador, pois, devido à baixa precisão dos equipamentos de medição utilizados, foi necessária a utilização de medição através de um resistor *shunt* que, como foi apresentado no anexo sobre caracterização do processador, não é a melhor abordagem de medição.

Outro problema encontrado foi em relação às ferramentas de análise de Redes de Petri Temporizada. As ferramentas existentes, por serem muito genéricas, não possibilitavam anotar as Redes de Petri com informações que possibilitassem a estimativa de consumo de energia. Com isso, foi necessário, durante todo a fase de pesquisa, um desenvolvimento de várias ferramentas, desde *parsers* a simuladores, o que demandou parte significativa do tempo de pesquisa.

7.1 CONTRIBUIÇÕES

Este trabalho apresentou uma abordagem para a fase de planejamento de projetos de sistemas embarcados, possibilitando a análise de aspectos de consumo de energia, mesmo

antes da disponibilidade do *hardware* final do projeto, desde que se tenha disponível dados de consumo das estruturas básicas do ANSI-C para a plataforma em que se deseja realizar a estimativa.

Para isso, foi desenvolvido um ferramental para análise e estimativa de consumo de energia que consistiu em um *parser* de estruturas de código ANSI-C em Redes de Petri Temporizada.

O modelo desenvolvido representa uma importante contribuição, já que abordou uma das linguagens de programação mais utilizadas em sistemas embarcados e permitiu a análise utlizando um método formal que permitirá extensões em trabalhos futuros.

Outro ponto de contribuição foi que o ferramental construído sob a forma de um *framework* de desenvolvimento, possibilitará sua extensão em trabalhos futuros, pois é possível incluir outros detalhes para estimativa e outros tipos de arquitetura.

7.2 TRABALHOS FUTUROS

Em relação à caracterização do processador, ou seja, ao processo formal de medição e obtenção de valores de consumo de tempo e energia das estruturas básicas, um desenvolvimento de um ambiente de medição mais adequado ajudaria na obtenção de valores mais precisos para validação dos modelos e estimativas.

A modelagem de outros tipos de microcontroladores, como por exemplo o ARM7 TDMI, e arquiteturas abrangeria a utilização da pesquisa realizada.

Outro ponto a ser abordado é a comunicação (passagem de mensagens e compartilhamento de memória) entre sistemas embarcados multiprocessados.

Por fim, em relação ao Modelo, poderá ser feito um melhor estudo com arquitetura com mais de um processador, pois, apesar de se apontar direções neste sentido, nenhum estudo de caso com múltiplos processadores foi realizado no presente trabalho.

APÊNDICE A

PROCESSO DE MEDIÇÃO DE CONSUMO E TEMPOS

Este apêndice descreve o processo de medição do consumo de energia e tempo de execução de cada estrutura básica do C. Primeiramente, são descritas as bases do processo de medição e o processo de descrição de atividades, logo após, a arquitetura de medição que foi usada é apresentada, bem como uma explanação sobre o formato da biblioteca de consumo construída.

A.1 FUNDAMENTAÇÃO TEÓRICA DO PROCESSO DE MEDIÇÃO

Antes da execução do processo de medição é necessário que se defina o processo de medição a ser adotado.

O produto da fase de planejamento da medição, primeiro bloco da Figura A.1, é um documento que descreve o protocolo de medição. Este protocolo descreve "o que, onde, como e a freqüência" do processo de coleta de dados, assim como a forma de armazená-los, o plano de análise e quem deverá executar cada tarefa.

Segundo [36], as tarefas necessárias para a realização dos experimentos podem ser observadas na Figura A.1.

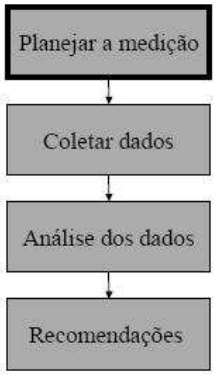


Figura A.1 Procedimentos para o Processo de Execução de Medição

No planejamento da medição, é importante ter um bom conhecimento do sistema

alvo, identificar os pontos em que os dados devem ser executados, assim como definir, de forma precisa, a forma com que os dados serão coletados para que o processo não seja tendencioso e possa ser reproduzido de forma confiável.

As próximas seções descrevem o *hardware* em que as medições foram executadas, o processo de medição e os pontos de dificuldades encontrados em relação ao equipamento, e a construção da biblioteca resultante da medição.

A.2 HARDWARE DE MEDIÇÃO

Para executar a medição do consumo de energia, é necessária a construção de um protótipo e definir uma estratégia de medição.

O processo de caracterização, ou seja, a definição dos valores de energia e tempo de acordo com o processador alvo, consiste na construção de um *Hardware* de medição, no código para medição de estruturas básicas, na utilização do código de *Benchmark* para validação dos resultados e na construção da biblioteca de consumo de energia.

O *hardware* de medição consiste em uma placa com o processador alvo com apenas algumas estruturas básicas para permitir a medição. Este *hardware* consiste de um Microcontrolador, Fonte de Alimentação, Circuito de Medição, Portas para comandar o início e o fim da medição, circuito de oscilação e porta serial para programar o microcontrolador. A Figura A.2 mostra o esquema de todo o protótipo.

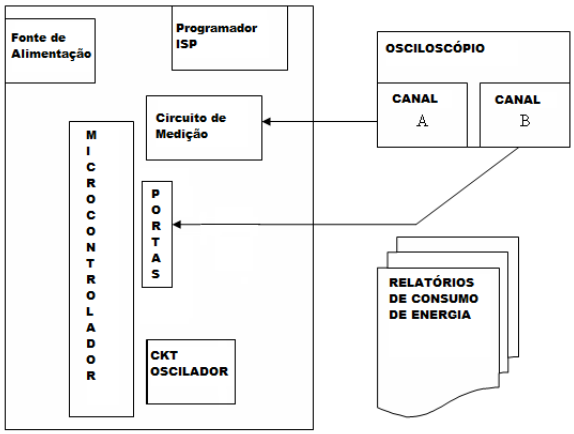


Figura A.2 *Hardware* usado para Caracterização do Processador - Medição Tempo e Potência(Energia)

O microcontrolador usado para a primeira validação do modelo foi o AT89S8252. Esse microcontrolador foi escolhido por ser um dos mais utilizados em projetos de sistemas

embarcados de propósitos gerais. Por ser um microcontrolador bastante utilizado no meio acadêmico, os resultados da pesquisa podem ser comparados com outras abordagens.

O uso de sistemas reguladores de fonte, como fontes chaveadas e circuitos mais precisos de medição, podem levar a resultados de estimativa mais precisos, pois os valores coletados na medição seriam mais próximos do valor real.

O circuito de medição pode ser construído de diversas formas. A escolhida para este trabalho, por ser de mais fácil implementação, foi a utilização de um resistor ligado à fonte de alimentação, chamado resistor *shunt* de forma em que um osciloscópio é utilizado para monitorar a diferença de pontencial no resistor considerando um determinado intervalo de tempo. Este intervalo é observado através de sinais emitidos pelo processador (através de terminais previamente definidos) que indicam o início e fim das medições.

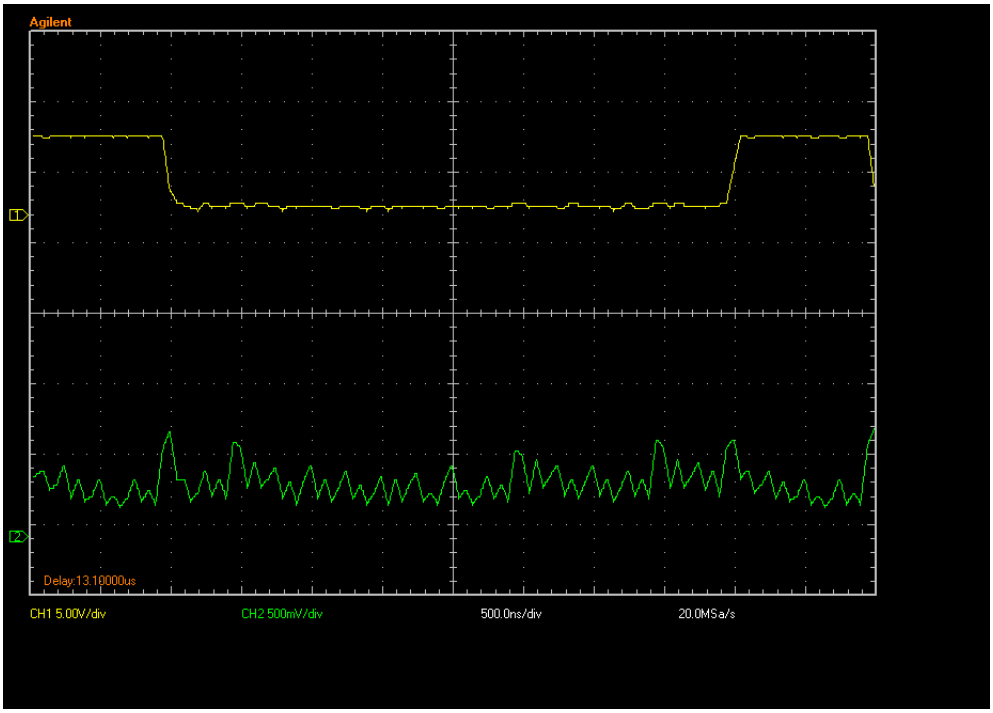
Adicionalmente, é necessário um circuito de programação, para execução dos aplicativos de medição das estruturas básicas e os códigos de avaliação, também chamados de *benchmarks*. Juntamente com o circuito de programação, é necessário o circuito de oscilação para definição da base de tempo do controlador. É importante que a freqüência de oscilação seja a mesma utilizada no sistema no qual se quer estimar o consumo de energia.

Um exemplo da saída do osciloscópio ligado às portas do microcontrolador pode ser observado na Figura A.2. São utilizados dois canais do osciloscópio, um para a medição da tensão no resistor *shunt* e o outro como *trigger* nas portas de comando de início e fim da medição. O resultado é extraído de uma planilha resultante e os valores médios são armazenados na biblioteca de consumo de energia.

Para validação do processo proposto, foi utilizado um *benchmark* PowerStone [3], criado para a avaliação de consumo de energia de produtos da Motorola, grande fabricante de sistemas embarcados. Esse código foi portado para o microcontrolador ATS8959 e, com o resultado, foi montada a biblioteca de consumo de energia mostrada em A.4.

A.3 PROCESSO DE MEDIÇÃO

Com relação à técnica de medição, o sistema caracterizador teve de lidar com duas questões básicas: a definição do resistor utilizado como mecanismo a ser monitorado e a adequação da taxa de amostragem do osciloscópio para medidas específicas. Com relação ao resistor, sua definição foi realizada empiricamente por meio da avaliação de diversas opções, pois a questão básica para a escolha é a determinação do seu valor e da



sua precisão. O resistor não deve interferir de forma significativa na tensão e corrente de alimentação do dispositivo, ao mesmo tempo em que não deve ter valor excessivamente baixo, de forma que a energia dissipada no próprio resistor seja demasiadamente alta e que possibilite a interferência significativa devida a sinais espúrios do ambiente externo.

Resistores espiralados em corpo cerâmico possuem reatância indutiva que faz com que o fator de qualidade do circuito RL resultante impacte na definição do sinal adquirido. Em resistores espiralados, o fator qualidade tende a piorar em baixas resistências, dada a redução da relação $\frac{R}{\omega L}$. Após a análise de diversos dispositivos, utilizou-se um resistor de filme metálico com 51Ω. O resistor com este valor proporcionou uma diferença de pontencial média na ordem 500mV, o que proporcionou excelente relação sinal/ruído para a medida. A queda de 500mV na alimentação do dispositivo está dentro da faixa de operações práticas do dispositivo que permite tensões na faixa de 4,5V a 6,0V, sendo 5,0V a tensão nominal recomendada.

O osciloscópio utilizado para medição foi o da marca *Agilent*, série 3000, cujas principais características são: 60 a 200MHz de largura de banda, 1 GSa/s de taxa de amostragem máxima, memória de sinal de 4kpts e conexão usb para extração de dados.

Com relação à configuração do osciloscópio, todas as medidas foram realizadas em

modo tempo real[1], sem que os modos de cálculo de valor médio fossem ativados. As medidas foram feitas dessa forma visando a observar flutuações estatísticas no consumo enquanto estivesse caracterizando uma instrução.

Como resultado, observou-se que o consumo é estável, não apresentando alterações significativas entre as diversas execuções de uma mesma instrução. Embora o osciloscópio utilizado opere em tempo real com taxa de amostragem nominal de 1GSa/s, sua baixa capacidade de armazenamento, 4000 pontos, força a redução gradativa da taxa de amostragem efetiva quando se utilizam bases de tempo superiores a 200ns/div.

A Figura A.3 ilustra esse efeito comparando a curva de decaimento da taxa de amostragem de um osciloscópio de 2500 pontos de memória com um sistema de aquisição com capacidade de armazenamento superior (PS-3206[37]), e com um sistema de operação em tempo equivalente (PS-3206-ETS[37]). Todas as medidas foram realizadas com o osciloscópio operando com base de tempo ajustada para 500ns/div, de maneira a manter a mesma precisão para todo o experimento de caracterização. Isso implicou medidas a uma taxa de amostragem de 500MSa/s.

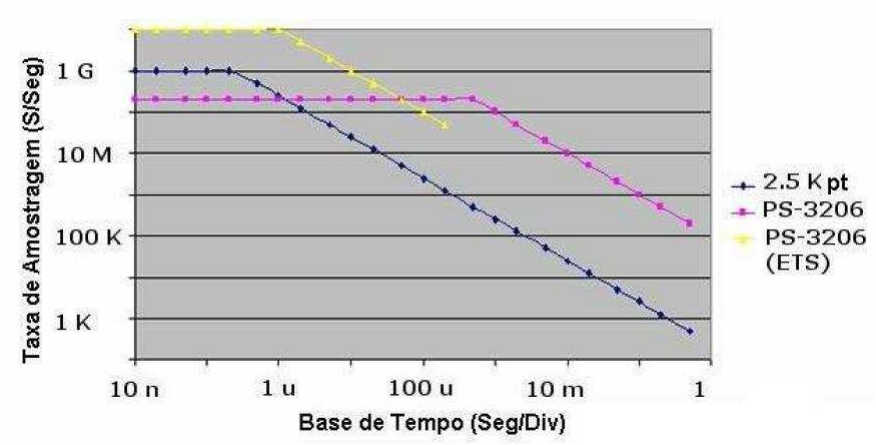


Figura A.3 Efeito da limitação de memória na redução da taxa de amostragem.

A.4 BIBLIOTECA DE CONSUMO DE ENERGIA

A biblioteca de consumo construída representa as características de consumo e temporais das estruturas básicas do ANSI-C. Tais características dependem do tipo de compilador

[1]Tempo Real no contexto representa o tempo exato de medição, ou seja, não é feito nenhum tratamento sobre o tempo calculado em frações de segundos

utilizado, que deve ser um compilador similar ao que será utilizado no projeto, e do *hardware* utilizado. Para este experimento, o compilador utilizado foi o da Keil [38] para o 8051. Outro ponto importante a ser observado é o tipo de otimização configurado no simulador, que pode alterar o resultado final da estimativa.

A biblioteca de consumo de energia foi construída a partir do resultado da medição, e armazenada em arquivo XML com os valores de consumo de energia e o tempo de execução das tarefas básicas definidas no modelo ANSI-C.

Essas atividades básicas foram: (i) Atribuição, (ii) Operação Aritmética e (iii) Operação Lógica. Com os resultados coletados dessas três estruturas básicas, foi possível a anotação de todos os estudos de caso e assim aconteceu a validação da estimativa, que vai ser apresentada no capítulo 6.

```
<pnml xmlns="http://www.informatik.hu-berlin.de/top/pnml/">
 <net xmlns="http://www.informatik.hu-berlin.de/top/pntd/ptNe
  <name>
   <text>Biblioteca Consumo 8051\bcnt\bcnt.c</text>
  </name>
   <resultadoMedição>
    <name>
     <text>Atribuição</text>
    </name>
    <toolspecific tool="Tempo=0.0003315"/>
    <toolspecific tool="Potencia=0.007098094"/>
   </resultadoMedição>
   <resultadoMedição>
    <name>
     <text>Expressão Aritmética</text>
    </name>
    <toolspecific tool="Tempo=0.0000663"/>
    <toolspecific tool="Potencia=0.002891294"/>
   </resultadoMedição>
   <resultadoMedição>
    <name>
     <text>Expressão Lógica</text>
    </name>
    <toolspecific tool="Tempo=0.00003315"/>
    <toolspecific tool="Potencia=0.003074824"/>
   </resultadoMedição>

 </net>
</pnml>
```

Figura A.4 Arquivo XML com o resultado das medições de Consumo de Energia e Tempo

A Figura A.4 mostra o arquivo XML formado a partir do processo de caracterização. O valor mostrado para cada estrutura representa a média calculada das medidas obtidas através do procedimento explicado anteriormente.

A biblioteca é formada por um rótulo que representa cada estrutura básica medida. O rótulo principal é denominado “resultadoMedição”. Como sub rótulos tem-se “Nome”,

identificador de cada tipo de estrutura, que pode ser Atribuição, Expressão Aritmética ou Expressão Lógica, “Tempo”e “Potência”.

A.5 CONSIDERAÇÕES FINAIS

Neste capítulo, foi mostrado o processo de realização da caracterização do processador usado na estimativa de consumo de energia. Inicialmente, descreveram-se o processo de medição, as respectivas atividades, e o ambiente utilizado para medição (*hardware* e execução da medição) e construção da biblioteca consumo de energia.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] M. Jungel and E. Kindler and M. Weber. The Petri Net Markup Language. Humboldt-Universit “at zu Berlin – Institut f”ur Informatik, Germany, 1999.

[2] Adilson ARCOVERDE JR. EZPetri, Um Ambiente para integração de linguagens de descrição de Redes de Petri. 2004.

[3] A. MALIK and B. MOYER. A low power unified cache architecture providing power and performance flexibility. In Proceedings of the International Symposium on Low Power, 2000.

[4] Brian W. Kernighan and Dennis M. Ritchie. The C Programming Language. Prentice Hall Software Series, Prentice Hall, Second Edition.

[5] Kirk Zurell. C Programming for Embedded Systems. R & D Books.

[6] Paulo Romero Martins Maciel, Rafael Dueire Lins, Paulo Roberto Freire Cunha. Introdução às Redes de Petri e Aplicações.

[7] Meuse N. O. Junior et al. Analysing Software Performance and Energy Consumption of Embedded Systems by Probabilistic Modeling: An Approach Based on Couloured Petri Nets. ICATPN, 2006.

[8] Fernando Ferreira Carvalho. Avaliação Estocástica de consumo de energia no projeto de sistemas embarcados. UFPE, Master Thesis, March 2004.

[9] Johann Laurent and Nathalie Julien et al. Functional Level Power Analysis: An Efficient Approach for Modeling the Power Consumption of Complex Processors.

[10] Gang Qu and Naoyuki Kawabe et al. Function-Level Power Estimation Methodology for Microprocessors.

[11] Chandra Krintz and Ye Men et al. Application-level Prediction of Battery Dissipation.

[12] Paul Landman. High-Level Power Estimation.

[13] Object Management Group. Unified Modeling Language. Version 2.1.1. Site: http://www.omg.org/technology/documents/formal/uml.htm.

[14] W. M. Zuberek. Timed Petri Nets, Definitions, Properties, and Applications. Microelectronics and Reliability, volume 31, number 4, pages 627–644, 1991.

[15] Modelagem e Simulação de Sistemas Computacionais. Prof. Graça Bressan. LARCS-PCS/EPUSP 2002

[16] Introduction to Discrete Event Systems, Christos G.Cassandras, Stéphane Lafortune. Kluwer Academic Publishers (Boston, Dordrecht, London)

[17] Holger Hermanns, Process Algebra and Markov Chains,Formal Methods and Tools Group, Faculty of Computer ScienceUniversity of Twente, AE Enschede, The Netherlands

[18] Tadao Murata. Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, pages 541–580 volume 77 number 4, apr, 1989.

[19] W. Brauer. Net Theory and Applications. Springer-Verlag, Hanburg, 1979. Lecture Notes in Computer Science.

[20] G. W. Brams. Réseaux de Petri: Théorie et Pratique, tome 1. Masson Editions, 1983.

[21] W. Reisig. Petri Nets: An Introduction. Springer-Verlag, 1982.

[22] J. L. Peterson. Petri Nets an Introduction. Prentice-Hall Inc, 1981.

[23] Teruel E. Silva, M. Petri nets for the design and operation of manufacturing systems. CIMAT'96, 1996.

[24] Teruel E. Vallete R. Pingaud H. Silva, M. Petri nets and production systems. Lecture Notes in Computer Science - Advances in Petri Nets, 1998. Edited by W. Riesig and G. Rozenberg.

[25] Balbo G. Conte G. Bobbio. A. Chiola G. Cumani A. Marsan, A. The effect of execution policies on the semantic and analysis of stochastic petri nets. IEEE TSE, 1989.

[26] M. K. Molloy. On the Integration of Delay and Throughput MEasures in Distributed Processing Models. PhD thesis, Los Angeles, CA, 1981. PhD. Thesis.

[27] P. H. Starke. Remarks on timed petri nets. Proc. 9th European Workshop on Application and Theory of Petri Nets, 1988.

[28] F. Bowden. Modelling time in petri nets. 2nd Australia-Japan Workshop on Stochastic Models, July 1996.

[29] Harhalakis G. Proth J. M. Silva M. Vernadat F. B. Dicesare, F. Practice of Petri Nets in Manufacturing. Chapman & Hall, London, 1993.

[30] Chung-Hsing Hsu and Ulrich Kremer. The Design, implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction.

[31] N. Vijaykrishnan and M. Kandemir et al. Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower.

[32] Gunter Bolch and Stefan Greiner and Hermann de Meer and Kishor Shridharbhai Trivedi. Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications.

[33] Jonathan Billington and Soren Christensen et al. The Petri Net Markup Language: Concepts, Technology, and Tools,

[34] Java.net Community. Javacc Community Page. Jan, 2007. Site:https://javacc.dev.java.net.

[35] Cesar A. L. Oliveira. Simulação de Redes de Petri em Ambiente Java. Universidade de Pernambuco, 2006.

[36] David J. Lilja, Measuring Computer Performance: A Practitioner's Guide. Cambridge University Press,2000.

[37] Osciloscópio Agilent. http://www.home.agilent.com/

[38] Embedded Development Tool, Keil an Arm Company. http://www.keil.com/ Agosto-2007.

[39] Cpntools Group. version 1.4.0. Site:http://wiki.daimi.au.dk/cpntools/cpntools.wiki.

[40] Marsan and Balbo et al. Modeling with Generalized Stochastic Petri Net.

[41] Enrico Macii and Massoud Pedram and Fabio Somenzi. High-Level Power Modeling, Estimation, and Optimization.

[42] R. David and H. Alla. Petri Nets and Grafcet – Tools for Modeling Discrete Event Systems. Prentice Hall, 1992.

[43] J. L. Peterson. Petri Net Theory and the Modeling of Systems. Prentice Hall, 1981.

[44] M. Zhou. Petri Nets in Flexible and Agile Automation. Boston, MA: Kluwer Academic Publishers, 1995.

[45] M. Zhou and F. DiCesare. Petri Net Synthesis For Discrete Event Control of Manufacturing Systems. Boston, MA: Kluwer Academic Publishers, 1993.

[46] Joel Coburn, Srivaths Ravi, and Anand Raghunathan. Power Emulation: A New Paradigm for Power Estimation. DAC 2005.

[47] Steve Holmes. C Programming. University of Strathclyde Computer Centre Curran Building 100 Cathedral Street Glasgow

[48] Byte Craft Limited. Proof that C can match or beat Assembly.

[49] McDermid J. A. Barroca, L. M. Formal methods: Use and relevance for the development of safety-critical systems. The Computer Journal, 1992.

[50] J. M. Spivey. Z Notation - A Reference Manual. Prentice Hall International, 1989.

[51] R. Milner. Communication and Concurrency. Prentice Hall, 1989.

[52] C. A. R. Hoare. Communicating Sequential Processes. Prentice Hall International, 1985.

[53] A formal description technique based on the temporal ordering of observational behaviour. Technical report, Information Processing System - Open Systems Interconnection, 1987.

[54] R. Goldblatt. Axiomatizing the Logic of Computer Programming. LNCS. Springer-Verlag, 1982.

[55] R. Alur and D. Dill. Automata for modeling real-time systems. In Proceedings of the seventeenth international colloquium on Automata, languages and program- ming, 1990.

[56] R. Alur and D. Dill. A theory of timed automata. Theor. Comput. Sci., 1994.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).