UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ESTIMADOR E CARACTERIZADOR DE CONSUMO DE ENERGIA PARA SOFTWARE EMBARCADO

FRANCISCO COELHO DA SILVA

MANAUS
2011

UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FRANCISCO COELHO DA SILVA

ESTIMADOR E CARACTERIZADOR DE CONSUMO DE ENERGIA PARA SOFTWARE EMBARCADO

Dissertação apresentada ao Programa de Pós-Graduação *Stricto Sensu* em Engenharia Elétrica da Universidade Federal do Amazonas, como requisito parcial para obtenção de título de Mestre em Engenharia Elétrica, área de concentração Controle e Automação de Sistemas.

Orientadora: Prof. Dr. João Edgar Chaves Filho
Co-Orientador: Prof. Dr. Raimundo da Silva Barreto

MANAUS
2011

FRANCISCO COELHO DA SILVA

ESTIMADOR E CARACTERIZADOR DE CONSUMO DE ENERGIA PARA SOFTWARE EMBARCADO

Dissertação apresentada ao Programa de Pós-Graduação *Stricto Sensu* em Engenharia Elétrica da Universidade Federal do Amazonas, como requisito parcial para obtenção de título de Mestre em Engenharia Elétrica, área de concentração Controle e Automação de Sistemas.

Aprovado em:

BANCA EXAMINADORA

Prof. Dr. João Edgar Chaves Filho, Presidente
Universidade Federal do Amazonas

Prof. Dr. Raimundo da Silva Barreto, Membro
Universidade Federal do Amazonas

Prof. Dr. José Reginaldo Hughes de Carvalho
Universidade Federal do Amazonas

MANAUS

2011

DEDICATÓRIA

À minha amada esposa Mayara e ao meu filho Maxwell, pelos sacrifícios e compreensão nas horas em que estive ausente, à minha avó e mãe pelo amor incondicional, por serem exemplos em todos os sentidos da minha vida, ao meu pai e aos meus irmãos.

AGRADECIMENTO

Deixo aqui registrado meus agradecimentos como um reconhecimento sincero da dedicação das pessoas que, durante estes dois anos contribuíram de forma direta ou indireta para a realização deste trabalho.

Agradeço em primeiro lugar a DEUS, pela minha vida e por todos os caminhos que até o momento tive que cruzar. Agradeço também a toda minha família, pois eles são a pilar mestre que suportam minhas edificações. Sem esse apoio o caminho percorrido nesses dois anos teria sido muito mais sinuoso e difícil de cruzar.

Agradeço sinceramente ao meu orientador Prof. Dr. João Edgar Chaves Filho e meu co-orientador Prof. Dr. Raimundo da Silva Barreto pela grande oportunidade dada para trabalhar em seus grupos de pesquisa e sendo orientado pelos próprios, além disso, ressalto a paciência, suas disponibilidades, principalmente “compreensão e amizade” oferecidas ao longo desse difícil percurso, o que me ajudou a tornar possível a concretização deste trabalho. Muito obrigado, Prof. João Edgar e Prof. Barreto.

Obrigado também a todos os membros do programa de pós-graduação em engenharia elétrica do Departamento de Ciência da Computação DCC da UFAM pela oportunidade de realizar o mestrado e pelos recursos fornecidos durante o curso.

E Também deixo registrado o reconhecimento a minhas amigas Michele e Maria do Carmo, pelo incentivo nos momentos complicados que passei durante esta caminhada.

Gostaria de agradecer a alguns colegas de curso que me ajudaram nessa caminhada.

Sou grato aos colegas e amigos Almir Kimura e Luciano Pinto pelas horas de esforço e dedicação aos estudos de PPE e SL.

Sou grato também aos amigos Bruna, Allan Roberto e Joel Parente pelas ajudas nas horas mais complicadas, um reconhecimento aos colegas de trabalho:

Da Envision: Francisco Aldenir , Edivaldo, Edson, Arnoldo, Benedito, Rudney, Rainne, Samuel, Aldervan, Caio, Leandro, Ronildo, Marcelo, Ricardo e Aldermir.

Da Nokia: Wesley e Eduardo Assunção.

Do INdT: Germano, Ocielide, Idelcio Cardozo, Iramylson e Abidon.

Da Infosigma: Roadrigo, Dias, Antunes, Fabíola, Elizangela, Marivaldo e Eduardo Lopes.

Que sempre me incentivaram para concluir este trabalho.

E finalmente, agradeço ao colega Carlos Mar pelo valioso trabalho de implementação do mecanismo de simulação de consumo de energia utilizado neste trabalho.

“Todas coisas cooperam para o bem, daqueles que amam a Deus.”

Romanos 8:28

RESUMO

Consumo de energia nos últimos anos tornou-se um aspecto importante em projetos de sistemas embarcados. A produção e utilização em larga escala dos dispositivos móveis tem imposto várias restrições como: peso, tamanho, tempo de vida útil da bateria e funcionalidades complexas. Dispositivos móveis operam sob uma fonte de energia limitada cuja autonomia e tempo de vida útil estão diretamente relacionados ao consumo de energia das aplicações. Estas questões contribuíram para incluir o consumo de energia como métrica de qualidade no projeto de sistemas embarcados.

Este trabalho tem como objetivo propor uma abordagem de medição, estimação e comparação do consumo de energia de código de programas escritos em linguagem ANSI-C, baseados em ensaios de códigos previamente escolhidos com características de consumo de energia e no tempo de execução. Para dar suporte à abordagem, uma ferramenta de estimação chamada PESTI foi estendida para atender múltiplos cenários probabilísticos.

Programas escritos em linguagem ANSI-C são embarcados no processador LPC2148 da família ARM 7. Nesse programa são inseridos *flags* de sinalização para *start* e *stop*, para delimitar o tempo de execução e medirmos o consumo de energia do código. Um *hardware* chamado de caracterizador de consumo de energia auxiliará na medição em tempo real de execução do código.

A ferramenta de estimação chamada PESTI com características probabilísticas e atribuições para múltiplos cenários probabilísticos é usada para estimar o consumo de energia do programa escrito em ANSI-C.

Validamos a abordagem proposta, executando um conjunto de experimentos mostrando a viabilidade da extensão da ferramenta de estimação e o caracterizador que, em conjunto, viabilizarão as estimativas de consumo de energia no processador alvo.

As atividades realizadas para a execução dos experimentos foram:

- Validar a abordagem proposta;
- Comparar os resultados medidos e estimados entre a ferramenta PESTI com o caracterizador para a mesma plataforma de hardware embarcada (ARM7).

Os experimentos foram divididos em três passos:

- Estimação dos códigos na ferramenta PESTI em simples e múltiplos cenários;
- Caracterização do código em questão;
- Comparação da caracterização e ferramenta PESTI.

Onde os resultados obtidos mostram uma diferença entre os valores estimados e simulados e os resultados medidos.

Os experimentos foram conduzidos sobre:

- AMD Turion(tm) II Dual Core Mobile M500, 2.20GHz, 4GB de RAM;
- SO Linux Distribuição Mint kernel 2.6.22;
- SO de 32 bits Windows 7.

Palavras-chave: Sistemas Embarcados, Consumo de Energia, Tempo de Execução, Grafo de Fluxo de Controle, Múltiplos Cenários Probabilísticos, Estimativas, ANSI C, Ferramenta PESTI, Caracterizador de Consumo de energia para software embarcado.

ABSTRACT

The energy consumption in the past years became a very important issue in embedded system projects. The high production and wide application of mobile devices have forced the emergence of various restrictions to this system, such as: weight, size and lifetime of batteries and multiple functionalities. Mobile devices works under limited power source that autonomy and lifetime are directly related to energy consumption of the running applications. These concerns have contributed significantly to include the energy consumption as metric for project quality in embedded systems.

The main goal of this work is to propose metrics, estimative and compare the energy consumption of programs code written in ANSI-C language, based on execution time of embedded systems. In order to support the approach it was improved a tool in algorithm level known as PESTI in multiple scenarios.

It was written a program in ANSI-C language and loaded in processor of the ARM 7 family's. Then, it was added into this program flags to signalize start and stop in order to measure execution time of each track in analysis.

The estimative tool already modified to attribute multiple scenarios, for a program written in ANSI-C and translated into an annotated control flow graph, with tracks assignments of probabilities. This model is probabilistically simulated by using Monte Carlo methodology. The proposed approach was validate carrying out a series of experimental in order to show the viability of the improved tool of estimation and characterization, which together will make the estimates of energy consumption somewhat more feasible.

- Validate the proposed approach added;
- Compare the results between simulation time and the tool for characterization PESTI with the same hardware platform embedded (ARM7).

The experimental were divided in three steps:

- Simulation of the code in the tool PESTI in multiple scenarios;
- Characterization of the query code;
- Comparison of the characterization tool and PESTI.

The experiments were conducted on:

- AMD Turion (tm) II Dual Core Mobile Processor M500, 2.20GHz, 4Gb of RAM;
- OS Linux Mint Distribution kernel 2.6.22 32-bit;

- OS Windows 7 32-bit.

Keywords: Embedded systems, energy consumption, runtime, control flow graph, multi scenarios probabilistic method, Estimative, ANSI C, PESTI tool, featured power consumption for embedded software.

ÍNDICE DE ILUSTRAÇÕES

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS

ARM	*Advanced RISC Machine*
ANSI	*American National Standards Institute*
PWM	*Pulse-Width Modulation*
JTAG	*Joint Test Action Group*
IBIDEM	*É um vocábulo de origem latina, com significado de "no mesmo lugar"*
CMRR	*Common-Mode Rejection Ratio*
DSP	*Digital Signal Processor*
USB	*Universal Serial Bus*
TCP/IP	T*ransmission Control Protocol/Internet Protocol*
PS/2	*Personal System/2*
RAM	*Random Access Memory*
A/D	*Analogic/Digital*
IC	*Integrate Circuit*
RISC	*Reduced instruction set computing*rganização
CFG	*Control Flow Graph*
RTL	*Register Transfer Level*
CSV	*Comma Separated Values*
IRQ	*Interrupt ReQuest*
FIQ	*Fast Interrupt ReQuest*
VLIW	*Very long instruction word*
CRC	*Cyclic Redundancy Code*
IDE	*Integrated Development Environment*

SUMÁRIO

CAPÍTULO I

INTRODUÇÃO

A explosão do uso dos dispositivos móveis levou muitos grupos de pesquisa a buscarem novas formas de diminuir o consumo de energia desses diversos sistemas, além de otimizarem esse gasto com novos hardware e novos algoritmos nos projetos de **Sistemas Embarcados**.

Com a miniaturização de componentes eletrônicos, dispositivos cada vez menores e o aumento com certeza das linhas de códigos embarcados em processadores, percebe-se a necessidade de compreender melhor a variedade de restrições à que esses sistemas são submetidos, como: peso, tamanho, tempo de vida útil da bateria e funcionalidades complexas baseadas em código embarcado. Esses dispositivos portáteis trabalham sob uma fonte de energia limitada cuja autonomia e tempo de vida útil está diretamente relacionado ao **Consumo de Energia** das aplicações. Os fatores mencionados anteriormente contribuem, consideravelmente, para o aumento do consumo de energia nos dispositivos embarcados, tornando esse fato uma métrica de qualidade no projeto de sistemas embarcados.

O termo **Sistema Embarcado** pode ser utilizado para qualquer sistema digital que esteja incorporado a outro com o intuito de acrescentar ou aperfeiçoar funcionalidades. O avanço da tecnologia portátil e embarcada, principalmente da microeletrônica, e o barateamento dos sistemas microprocessados, fez com que ocorresse uma explosão da utilização desse tipo de sistema nas mais diversas áreas da vida humana [3]. Como resultado dessa revolução, veio o desenvolvimento em larga escala de sistemas digitais com funcionalidade específica. Entretanto, as técnicas e paradigmas existentes para o desenvolvimento de aplicações não focavam essa nova classe de sistema. Dessa forma, para desenvolver sistemas competitivos, ou seja, de baixo custo e boa qualidade, métodos de projeto precisaram ser criados ou adaptados. Devido a isso, grande parte das pesquisas, ainda hoje, visa à criação ou à melhoria de técnicas já desenvolvidas.

Em projetos de sistemas embarcados, percebemos que as tarefas não são triviais, pois as mesmas podem estar sujeitas às mais diferentes restrições [1], como: tamanho, peso, custo, consumo de energia, *time-to-market* e confiabilidade. Atualmente, consumo de energia, particularmente em sistemas embarcados, é ponto de discussão em pesquisas em indústrias e em pesquisas acadêmicas. Essas discussões são respaldadas na massificação do uso de

dispositivos móveis e na exigência cada vez maior do mercado para que os mesmos tenham grande potencial computacional e maior grau de autonomia.

Devido a essas exigências, uma das métricas de qualidade passou a ser o consumo de energia no contexto do desenvolvimento de projetos de sistemas embarcados. Além disso, o consumo de energia tem impacto na forma de encapsulamento do sistema, implicando no custo final do produto.

Um sistema embarcado tem como fontes de consumo três componentes: o *software*, o *hardware* e os barramentos. Entretanto, os maiores responsáveis pelo consumo de energia do sistema são o *software* e o *hardware*, pois suas descrições estão diretamente vinculadas ao padrão comportamental do sistema [3]. Nesta pesquisa, o interesse concentra-se em mecanismos de avaliação de consumo de energia estabelecidos pelo *software*, pois o mesmo determina um padrão de execução e, consequentemente, de consumo sob o processador no qual o executa [24]. Nesse sentido o ARM7 é um processador novo e pouco estudado. A Figura 31 mostra um diagrama de idealização do processador ARM7 em diversas aplicações, devido à sua grande gama de dispositivos incorporados a um único chip e suas respectivas aplicações dedicadas.

Neste trabalho foi realizada a extensão do trabalho desenvolvido em [2], a ferramenta PESTI, que é algorítmico de consumo de energia e estimativa de desempenho para sistemas embarcados, a fim de suportar múltiplos cenários probabilísticos, os quais somente eram possíveis realizar se fossem realizadas um a um.

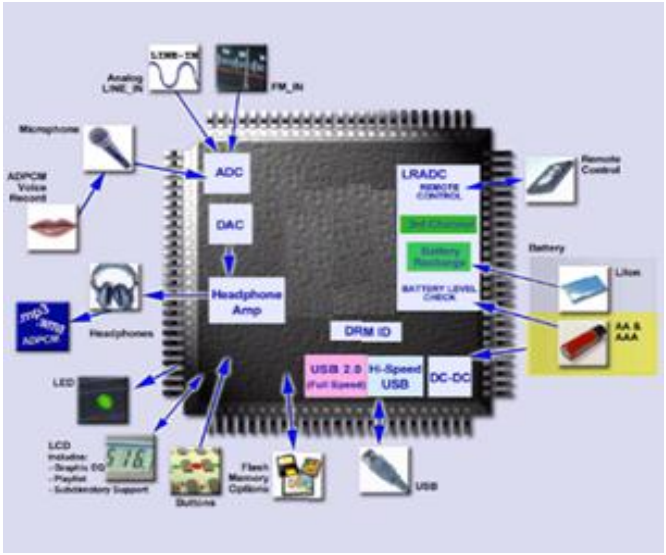


Figura 1 – Aplicações como processador ARM.

Fonte: Autor

1.1 CONTEXTUALIZAÇÃO

Para o desenvolvimento de projetos de sistemas embarcados, é necessário englobar diversos requisitos e restrições, as quais devem ser plenamente atendidas, de forma simples, coerente e eficaz. Quando sistema embarcado é projetado seguindo esses requisitos, isto faz com que o sistema consuma a menor quantidade de energia possível operar, garantindo maior autonomia por longos períodos de tempo sem necessidade de recarga [1]. Buscando atingir esse objetivo, os projetistas devem buscar executar métodos de avaliação rápidas e que lhes forneçam estimativas confiáveis a respeito do consumo de energia do sistema.

Este método de avaliação escolhido e o nível de abstração aplicado tem influência direta nos resultados obtidos, ou seja, quanto mais baixo for o nível de abstração aplicado, mais custosa e complexa torna-se seu desenvolvimento, entretanto, seus resultados tendem a ser mais precisos, pois quanto mais abstrato é o modelo mais impreciso são seus resultados e menor é o esforço para obtê-los [2].

Quando o consumo de energia passou a ser métrica de qualidade para os projetos de sistemas embarcados, foram propostas diversas abordagem de como estimar consumo de energia, sendo essas abordagens baseadas em hardware ou software.

Utilizando métodos baseados em hardware, os projetistas devem conhecer detalhadamente a arquitetura em questão, dessa forma a estimativa é levada ao baixo nível do sistema, o que torna este método pouco flexível e complexo e que não pode ser aplicado a todos os tipos de dispositivos existentes, além de limitar-se em questões de propriedade intelectual.

Uma das formas de se avaliar o sistema é através de modelos baseados em instrução. Esses modelos avaliam o consumo de energia do sistema por meio da simulação do conjunto de instruções do processador, mas não possuem nenhuma especificação formal de seu mecanismo de computação interna. São mais flexíveis que os mecanismos de simulação por hardware, permitem a avaliação do sistema em um nível de abstração mais elevado e não exigem que o projetista seja um especialista no processador sob o qual o sistema irá executar.

Então, é possível o desenvolvimento de uma ferramenta baseada em grafo de fluxo de controle para estimar o consumo de energia em sistemas embarcados que possibilite a aplicação de um nível de abstração como o de um programa escrito em linguagem ANSI C. Adicionalmente, para computar o consumo de energia, a ferramenta pode ser dotada de um modelo de consumo de energia baseado em instrução.

1.2 **DESCRIÇÃO DO PROBLEMA**

Uma das fontes de consumo de energia em um sistema embarcado é o *software*, e quando embarcado em um processador apresenta um padrão de consumo ao ser executado. O problema considerado neste projeto é expresso pela seguinte pergunta: "Dado um programa escrito em linguagem ANSI-C e um processador da família ARM, sob o qual esse programa é executado, e um modelo de consumo de energia baseada em ANSI-C, é possível determinar probabilisticamente os perfis de consumo desse código antes de baixá-lo para o processador e fazer comparações com medições reais através de um caracterizador?”.

Esse questionamento é respondido, porque dentro do contexto do projeto de sistemas embarcados, o consumo de energia, além de ser métrica de qualidade é também uma restrição imposta pela especificação não funcional do sistema.

Sendo resposta afirmativa para aplicações em *assembly* utilizando o microcontrolador da família 8051, especificamente o AT89S8252 [3], surge então, a necessidade de se criar ferramentas para tratar a complexidade dos cenários de execução sobre o código ANSI-C para que sejam comparados resultados com medições reais.

Portanto o problema consiste em estimar e caracterizar o consumo de energia de um programa escrito em ANSI C.

Nesse nível de abstração, as estimativas são menos precisas e as medições de consumo de energia também se tornaram um desafio necessário para projetistas e desenvolvedores, porém o esforço e o custo para gerá-las são menores.

Dependendo da complexidade da função que será desempenhada pelo sistema, da experiência do projetista de *hardware* e dos desenvolvedores de *firmware/software* e também de outras características inerentes ao projeto, a relação custo/benefício entre a obtenção de estimativas precisas e o esforço necessário para alcançá-las podem ser compensadoras quando tais estimativas possam ser tomadas nas fases de mais alto nível do projeto além de serem realizadas medições caracterizadoras para comparações entre simulações e medições reais de consumo de energia.

1.3 **OBJETIVO GERAL**

É desenvolver uma ferramenta para estimar e caracterizar o consumo de energia de software embarcado utilizando programas escritos em linguagem ANSI-C embarcados em um processador ARM7 considerando múltiplos cenários.

1.4 **OBJETIVOS ESPECÍFICOS**

Como objetivos específicos têm-se:

- Construir um caracterizador de consumo de energia para Código ANSI-C de um processador ARM7.
- Implementar um hardware específico para monitoração do Consumo de Energia devido ao Software ANSI-C embarcado.
- Estender o estimador probabilístico de consumo de energia, desenvolvido em [2], alterando as anotações de código e estimando múltiplos cenários.
- Validar o método de estimativa e caracterização de consumo de energia através de experimentação em aplicações de testes existentes (*benchmarkings*).
- Comparar os resultados obtidos das estimativas com medições reais.

1.5 **PROCEDIMENTOS METODOLÓGICOS**

De forma aplicada e quantitativa, este trabalho utiliza métodos e procedimentos científicos com pesquisa bibliográfica e documental, além de um estudo de caso, sendo aplicada a categoria de processadores ARM7.

A pesquisa bibliográfica se limitou a trabalhos relacionados à simulação, estimação e caracterização de consumo de energia em processadores mais simples como os da família 8051 e aos processadores mais robustos como o ARM7.

Na pesquisa documental foram consultados as especificações técnicas tanto dos componentes utilizados ou encontrados na revisão bibliográfica como os possíveis componentes utilizados neste projeto ajudando assim a definir quais os mais viáveis para utilização na implementação prática.

No estudo de caso, utilizamos um cenário definido, onde as simulações, estimativas e medições serão construídas a partir dos métodos propostos. Para a etapa de simulação, aplicaremos o método de Monte Carlo utilizando a ferramenta PESTI desenvolvida em [2]. Na etapa de estimativa, aplicaremos o método probabilístico mencionado anteriormente. Na etapa de medições, a implementação prática do modelo será desenvolvida onde aplicaremos o caracterizador através do qual obteremos as medições reais do cenário utilizado nas simulações e estimações.

O desenvolvimento do circuito de caracterização para a realização das medições proporcionou um passo importante para a comparação de valores simulados, medidos e estimados, sendo esses valores tratados em forma de gráficos e tabelas.

Formalmente, todo o processo se resume em realizar simulações do consumo de energia no processador ARM7, estimar valores simulados, realizar medições no caracterizador e em seguida, tratar os valores simulados, estimados e medidos.

O método proposto para a realização de estimativas e as caracterizações de consumo de energia de software ANSI-C embarcado foi realizado basicamente através de cinco passos, dos quais três serão automatizados pelas ferramentas já desenvolvidas pelos autores consultados. Os passos que compõem o método são:

- Upload do código ANSI C para o processador ARM7;
- Aquisição do modelo de consumo de energia baseado no código ANSI-C;
- Simulação probabilística do trecho de programa na ferramenta PESTI, utilizando o método de Monte Carlo;
- Análise dos resultados obtidos pela simulação;
- Caracterização do consumo de energia durante a execução do trecho de código em um ambiente controlado.
- Realizar um tratamento de dados adquiridos nos ensaios e medições.

A Figura 62 mostra os campos de concentração das linhas de pesquisa sobre o consumo de energia, inclusive a linha proposta neste trabalho.

Estimativa por análise probabilística (área azul) - O modelo probabilístico de circuitos que analisa valores médios de consumo, passando por uma inferência de probabilidades, usando uma ferramenta para fazer as análises necessárias das inferências para que se tenha a estimativa de consumo em potência.

Estimativa por simulação (área cinza) - A simulação de circuitos modelados. Em seguida, são realizadas análises dos padrões de formas de ondas geradas pelos códigos embarcados na plataforma trabalhada e por fim gerando a estimativa de consumo expressado em potência.

Estimativa de consumo por caracterização (área amarela) - é uma linha pouco explorada nos trabalhos relacionados, o que torna este projeto uma contribuição importante a esta area que pesquisa.

Pode-se usar o modelo de consumo de energia no qual a ferramenta é baseada em duas formas. A primeira é por meio de um processo de caracterização energética do conjunto de

instruções do processador que pode ser encontrado em [24] ou [3]. A segunda é através de *datasheets* fornecidos pelo fabricante da plataforma, quando esses disponibilizam tais informações. Esse passo é executado apenas uma vez e está vinculado ao processador escolhido para a caracterização.

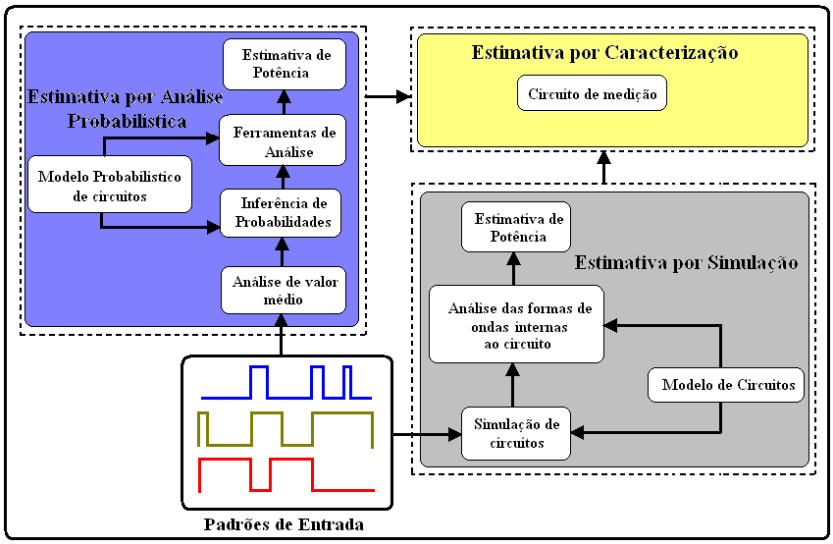


Figura 2 - Estimativas baseadas em Simulação, probabilidade e caracterização.

Fonte: Autor

Com o intuito de garantir a execução da pesquisa com êxito, os resultados esperados e que as metas definidas sejam alcançadas, a seguinte metodologia será utilizada como forma de nortear sua execução.

Para um melhor entendimento, as atividades descritas, as quais fazem parte desta metodologia adotada, mostram os passos a serem utilizados para a contemplação deste trabalho:

Resumo da metodologia:

- Pesquisa Bibliográfica (artigos relacionados ao tema);
- Estudo e definição de Padrão de Anotação;
- Estudo da definição do Formato do CFG;
- Estudo da Ferramenta PESTI;
- Estudo do Simulador de instruções do LINUX utilizando a técnica de Monte Carlo;
- Processador ARM e definição dos componentes de projeto do caracterizador
- Estudar e aplicar modelo de consumo de energia do processador selecionado
- Aplicar estudos de caso
- Projeto e ajuste do caracterizador de consumo de energia

- Elaborar dissertação
- Revisão da dissertação
- Defender dissertação.

1.6 **ORGANIZAÇÃO**

A organização deste trabalho conserva uma relação de interdependência entre os capítulos, de modo que conceitos e definições apresentados nos capítulos iniciais são amplamente utilizados nos capítulos seguintes:

i. Introdução (Capítulo I);
ii. Revisão Bibliográfica (Capítulo II);
iii. Trabalhos Correlatos (Capítulo III);
iv. Modelos de consumo de energia (Capítulo IV);
v. Ferramenta PESTI (Capítulo V);
vi. Caracterizador (Capítulo VI);
vii. Resultados e Experimentos (Capítulo VII)
viii. Conclusão e Trabalhos Futuros (Capítulo VIII)
ix. Referências;
x. Apêndice.

No Capítulo II, são abordados os seguintes tópicos: Histórico dos sistemas embarcados e seus conceitos, além de suas características, microprocessadores e microcontroladores; um breve estudo sobre a plataforma ARM e o seu núcleo de processamento; comentário sobre amplificadores operacionais de instrumentação; conceitos sobre CFG e um embasamento teórico sobre consume de energia devido ao *software*.

No Capítulo III, são realizados detalhados estudos sobre os trabalhos relacionados a esse tema.

No Capítulo IV, é abordado modelos de consumo de energia.

No Capítulo V, apresenta-se a ferramenta de estimação de consumo de energia PESTI, desenvolvida por Carlos Mar [2].

No Capítulo VI, apresenta-se o projeto do caracterizador de consumo de energia.

No Capítulo VII, apresentam-se os resultados e os experimentos utilizando a ferramenta PESTI e o Caracterizador.

No Capítulo VIII, resume-se a conclusão dos resultados e fornece ideias para pesquisas futuras.

Por fim, no Apêndice, apresentam-se os códigos usados neste trabalho.

CAPÍTULO II

FUNDAMENTAÇÃO TEÓRICA E BIBLIOGRÁFICA

2.1 INTRODUÇÃO

Em alguns dados de pesquisas em alta tecnologia, mais de 90% dos microprocessadores fabricados mundialmente são destinados a máquinas que usualmente não são chamadas de computadores. Dentre alguns destes dispositivos estão aparelhos celulares, fornos microondas, automóveis, aparelhos de DVD e PALM´s. O que diferencia este conjunto de dispositivos de um computador "convencional" (PC – *Desktop*, *Notebook*), conhecido por todos, é o seu projeto baseado em um conjunto dedicado e especialista constituído por Hardware, Software e Periféricos – um Sistema Embarcado [18].

Um Sistema Embarcado, *Embedded System*, pela sua natureza especialista, pode ter inúmeras aplicações. Pode-se ter sistema embarcado para controle de freios de um veículo automóvel, em que esse sistema deve gerenciar certos periféricos de controle como um sensor. Em outra aplicação, um sistema embarcado, através de suas funções de aquisição de dados, captura informações dos sensores de temperatura e umidade e envia estes dados a um display ou para um computador via comunicação serial.

Os Sistemas Embarcados encontram-se cada vez mais presentes em nosso dia-a-dia, e com uma utilização e importância crescente, tornam-se necessários estudos nas áreas de projeto em *hardware*, *software* e interfaceamento com base em outros dispositivos que usem esse princípio de desenvolvimento.

2.2 HISTÓRICO DOS SISTEMAS EMBARCADOS

Sistemas embarcados têm sua origem no fim da década de 1960 [17]. Nessa época, o que existia era um pequeno programa de controle funcional de telefones. Em pouco tempo esse pequeno programa escrito em *assembly* estava sendo usado em outros dispositivos de forma customizada com pequenas adaptações nos sinais de entrada e saída do programa.

Com o advento de microprocessadores especialistas, foi possível desenvolver software específico para os variados tipos de processadores. Os programas eram escritos em linguagem de máquina. Na década de 1970 começavam a surgir bibliotecas de códigos direcionados para sistemas embarcados para processadores específicos. Atualmente, os sistemas embarcados

podem ser programados em linguagens de alto nível e podem até possuir sistemas operacionais embarcados.

2.3 CONCEITOS DE SISTEMAS EMBARCADOS

Um sistema é classificado como embarcado quando este é dedicado a uma única tarefa e interage continuamente com o ambiente a sua volta por meio de sensores e atuadores [17]. Por exigir uma interação contínua com o ambiente, esse tipo de sistema requer do projetista um conhecimento em programação, sistemas digitais, noções de controle de processos, sistemas de tempo real, tecnologias de aquisição de dados (conversão analógico/digital e sensores) e de atuadores (conversão digital/analógico, acionamento eletromecânico e PWM), além de cuidados especiais na eficiência de estruturação do projeto e do código produzido.

A denominação "embarcado" (do inglês *Embedded Systems*) vem do fato de que esses sistemas são projetados geralmente para serem independentes de uma fonte de energia fixa, como uma tomada ou gerador. As principais características de classificação desse sistema são a sua capacidade computacional e a sua independência de operação. Outros aspectos relevantes dependem dos tipos de sistemas, modos de funcionamento e itens desejados em aplicações embarcadas.

Figura 3 – Lógica de um Sistema Embarcado usando um microprocessador.

Fonte: Ball *et al.* (2005).

Todo sistema embarcado é composto por uma unidade de processamento, que é um circuito integrado, fixado a uma placa de circuito impresso. Possui uma capacidade de processamento de informações vinda de um software que está sendo processado internamente nessa unidade. Logo o software está embarcado na unidade de processamento. Todo software embarcado é classificado de firmware [17].

2.4 CARACTERITICAS DE UM SISTEMAS EMBARCADO

Outros aspectos relevantes são referentes aos tipos de sistemas, modos de funcionamento e itens desejados em aplicações embarcadas.

2.4.1 Tipos de Aplicações de Sistemas Embarcados

• **Propósito geral:** são as aplicações mais parecidas com os computadores de mesa, mas em embalagens embarcadas. Nelas costuma haver grande interação entre os usuários e o sistema. Como exemplo, tem-se os videogames, os conversores de TV a cabo, caixas de bancos, etc.

• **Sistemas de controle:** controles em malha fechada com realimentação em tempo real. Geralmente são as aplicações mais robustas, com placas dedicadas e múltiplos sensores de entrada e saída. Muitas vezes, fornecem pouca interação com o usuário, mostrando sinalizações através de LED's. Usados nos motores de automóveis, processos químicos, controle de vôo, usinas nucleares, etc.

• **Processamento de sinais:** envolve um grande volume de informação a ser processada em curto espaço de tempo. Os sinais a serem tratados são digitalizados através de AD's, processados, e novamente convertidos em sinais analógicos por DA's. Caso de tratamento de áudio, filtros, modens, compressão de vídeo, radares e sonares, etc.

• **Comunicações e redes:** chaveamento e distribuição de informações. Sistemas de telefonia e telecomunicações e internet.

2.4.2 Modos de Funcionamento de Sistemas Embarcados

Os dois modos de funcionamento dos sistemas embarcados, apresentados abaixo, são determinantes para saber como programar o dispositivo e como será seu funcionamento e comportamento na aplicação para o qual foi desenhado.

• **Reativo:** o funcionamento se dá como resposta a eventos externos que podem ser:

- Periódicos (caso de sistemas rotacionais ou de controles de *loop*)
- Assíncronos (pressionamento de um botão por parte do usuário).

Há, então, uma necessidade de entrada de dados para que aconteçam as ações

de funcionamento.

Geralmente não há limite de tempo para que os sinais de entrada sejam acionados, pois dependem da interação com o usuário ou com o processo ao qual é destinado. Porém, a saída, função do sinal de entrada, deve ser realizada exatamente após os sinais de entrada começarem a atuar.

• **Controle em tempo real:** existem limites de tempo para executar cada tarefa (leitura de sensor, emissão de sinais para um atuador, atualização de display, etc.). Por isso mesmo, nem sempre tempo real é igual ao modo mais rápido de executar uma tarefa. Esses modos de operações, por serem cíclicos, não dependem da entrada de sinais para executar as atividades, sendo capaz de tomar decisões referentes à ausência dos mesmo. Os sistemas de tempos real são classificados em:

- ✓ **Soft Real Time**: As tarefas podem ser executadas em um intervalo de tempo específico, sem consequências graves se este limite de tempo não for cumprido. Um exemplo é um sistema bancário, onde apenas uma mensagem de erro aparecerá se determinada tarefa não for realizada dentro do tempo pré-determinado.
- ✓ **Hard Real Time**: As tarefas devem ser executadas em um tempo específico, com consequências graves se qualquer tarefa falhar. Como exemplo pode-se pensar nos sistemas de controle de um avião (Figura 4), onde uma falha pode resultar em queda e perdas de vidas.

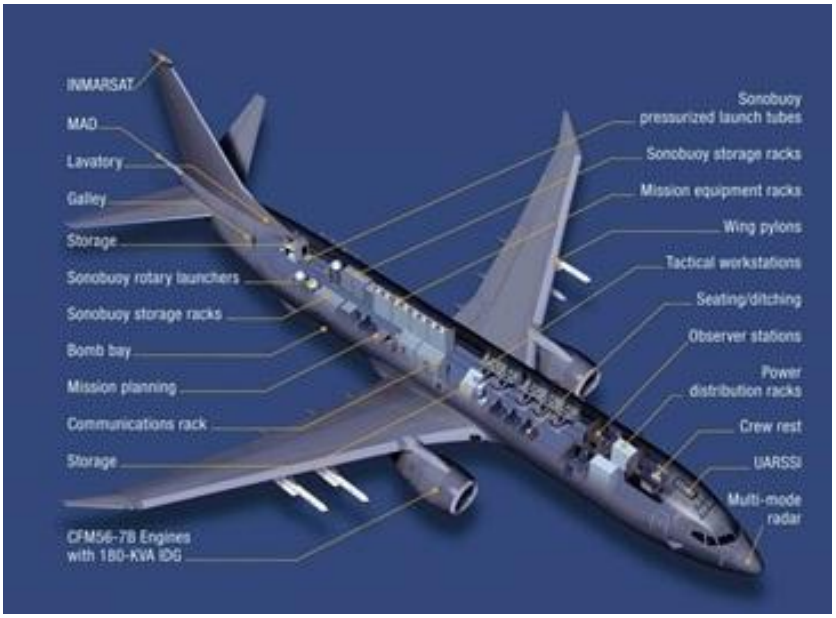


Figura 4 – Aplicações de Sistemas Embarcados na indústria Aeronáutica.

Fonte: Autor

2.5 **MICROPROCESSADORES X MICROCONTROLADORES**

Os microprocessadores são componentes dedicados ao processamento de informações com capacidade de cálculos matemáticos e endereçamento de memória externa. Utilizam barramentos de dados, controle e endereços para fazer acesso aos periféricos de entrada e saída e dependem de circuitos integrados externos como memória para armazenamento de dados e execução do programa, conversor A/D para aquisição de dados analógicos e sensores e outro periférico necessário conforme aplicação do sistema. A vantagem dos microprocessadores é que ainda possuem maior velocidade de processamento e são usados em soluções mais complexas, porém essa vantagem está prestes a igualarem-se e até mesmo serem superados pelos microcontroladores com seus núcleos de 16 e 32 bits.

Os microcontroladores são pequenos sistemas computacionais bastante poderosos que englobam em um único chip: interfaces de entrada/saída digitais e analógicas, periféricos importantes como a memória RAM, memória *flash*, interfaces de comunicação serial, conversores analógicos/digitais e temporizadores/contadores. A vantagem dos microcontroladores é que além de possuirem os periféricos integrados a um único chip, são responsáveis por executar e armazenar os programas escritos para eles (*firmware*), assim como a capacidade de absorver mais funções com o incremento de periféricos, através de CI's, como comunicação USB, pilha do TCP/IP, comunicação RF e porta PS/2. Com o advento dos microcontroladores de 16 e 32 bits (atualmente o padrão é de 8bits) a capacidade de gerenciar soluções mais complexas e maior velocidade de processamento se iguala ao do microprocessador. O crescimento dos sistemas embarcados muito se deve a esse componente.

Figura 5 – Microprocessadores e Microcontroladores.

Fonte: Autor

2.6 **PLATAFORMA ARM**

Os processadores ARM tem como principal característica, um baixo consumo de energia e um alto nível de processamento; essas características são desejáveis na maioria dos equipamentos eletrônicos. A arquitetura dos processadores ARM é RISC, o núcleo ARM7 está disponível em vários microprocessadores de diversos fabricantes como: Philips, Analog Devices, OKI, entre outros [21].

2.7 **HISTÓRIA DO PROCESSADOR ARM**

Em 1983 a *Acorn Computer* Ltd. desenvolveu o primeiro modelo do processador ARM. Já em 1985 incidiu o lançamento do processador ARM1, resultando em produto. O sucesso foi tamanho que no ano adjacente foi lançado o ARM2. Em 1990 foi realizada uma parceria entre *Apple Computers*, VLSI Technology e a *Acorn Computer Ltd*. Dessa parceria passou a existir a *Advanced* RISC *Machine* Ltd [22].

A Advanced RISC Machine Ltd. não perfaz os processadores ARM, ela os projeta e licencia. Várias empresas de semicondutores produzem os processadores ARM adicionando suas próprias funcionalidades e periféricos. Algumas das empresas que produzem processadores ARM são: *Philips, Atmel, Texas Instruments, Cirrus Logic, Intel, IBM, Sharp e Samsung*, entre outras (IBIDEM).

Os processadores ARM podem ser divididos em famílias, sendo elas:

• **Processadores para aplicações**: Processadores projetados para executarem sistemas operacionais para aplicações embarcadas, sendo uma aplicação dessa família os telefones celulares. Podem ser citados os seguintes processadores que fazem parte dessa família: ARM1020E, ARM1022E, ARM1026EJS, ARM11 MPCore, ARM1136J(F)-S, ARM1176JZ(F)-S, ARM720T, ARM920T, ARM922T, ARM926EJ-S, entre outros;

• **Processadores para sistemas embarcados**: Processadores de soluções de tempo real, aplicações industriais e automotivas podem ser um exemplo do emprego dessa família de processadores. Os seguintes processadores que fazem parte dessa família: ARM Cortex-M3,17ARM1026EJS, ARM1156T2(F)-S, ARM7EJ-S, ARM7TDMI, ARM7TDMI-S, ARM946ES, ARM966E-S, ARM968E-S, entre outros;

• **Processadores secure cores**: Destinados a aplicações que necessitem de segurança, o seu núcleo incorpora várias características de segurança e esses processadores são utilizados em

aplicações bancárias, TV paga, redes e biometrias. Alguns dos processadores que fazem parte dessa família: SecurCore SC100, SecurCore SC110, SecurCore SC200, SecurCore SC210.

Ano	Processador	Características principais
1985	ARM1	Protótipo que não chegou ao mercado
1986	ARM2	Possuía um barramento de dados de 32 bits, porém só usava 26 bits, deixando os 6 bits restantes para as flags de estado
1989	ARM3	4KB de cache , melhorou a performance.
-	ARM4	Não foi lançado ao mercado
-	ARM5	Não foi lançado ao mercado
1991	ARM6	Processador 32 bits, tinha somente 35000 transistores.
1994	ARM7	Usa metade da energia usada no ARM6, e tem de 50% a 100% mais performance.
1995	StrongARM	Baseado na Arquitetura ARM da ARM Limited. Ele foi criado pela DEC, que posteriormente o vendeu para a Intel que continuou a sua produção, logo o substituindo pelo XScale
-	ARM7TDMI	Pipeline de 3 estágios.. Processador de 32 bits que combina tamanho reduzido, pouca energia e alta performance.
-	ARM8	Pipeline de 5 estágios, banda de memória duplicada.
-	ARM9TDMI	Pipeline de 5 estágios também, usado em calculadoras HP. Reduziu o espaço do programa executável em 35%.
-	ARM9E	Processador de 32 bits, incluindo instruções da ARM®, Thumb® e Dsp.
-	ARM10E	Pipeline de 6 estágios, cache variando entre 32k e 16k, dependendo do modelo.
-	XScale	Linha produzida pela Intel, baseada na arquitetura StrongARM, usam menos energia porque trabalham em velocidades inferiores do que os CPUs Desktop.
-	ARM11	532-665 MHz, usado em comunicadores como Nokia E90, Apple iPhone.
-	Cortex	Atualmente o processador mais poderoso da ARM, de 600Mhz até mais de 1 GHz. Inclui 13 estágios de pipeline, com estados de espera programáveis.

Tabela 1 – Evolução da família ARM

Fonte: Autor

2.8 **NÚCLEO DO PROCESSADOR ARM**

O LPC21xx utiliza o núcleo ARM7TDMI-S. As principais características desse núcleo são[21]:

• **Modo *Thumb***: segundo conjunto de instruções com 16 bits;

• **Multiplicação longa**: capacidade de realizar multiplicações de 32 bits com resultados de 64 bits;

• **Depuração**: possui com extensão de hardware para depuração através da porta JTAG;

• **Embedded ICE**: extensão do módulo de depuração permite adicionar pontos de parada (breakpoints) e visualização dos registros através da porta JTAG;

• **Alta capacidade de processamento**: 0.9 MIPS/MHz;

O processador ARM possui dois conjuntos de instruções, ARM e Thumb. Quando a intenção do uso do processador ARM for velocidade, deve-se utilizar o conjunto de instruções ARM, pois esse modo garante um desempenho em torno de 40% em relação ao modo Thumb;

porém, quando se almeja economia de memória, o conjunto de instruções *Thumb* nos garante 30% de economia em relação ao conjunto de instruções ARM [21].

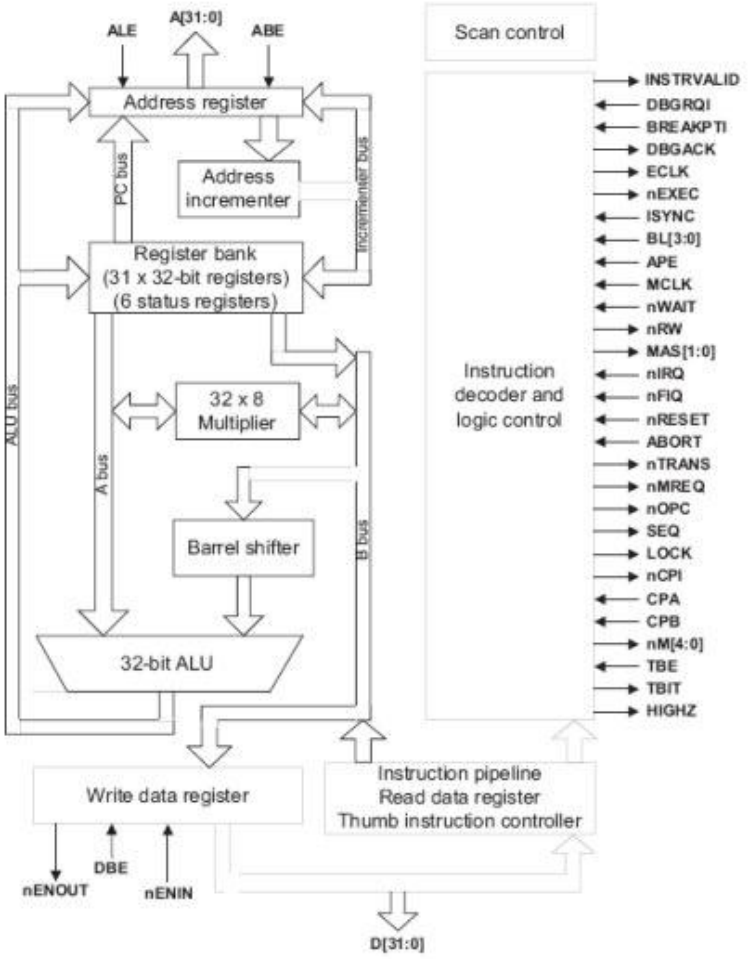


Figura 6 – Diagrama blocos do núcleo ARM7.

Fonte: Manual de referência do processador ARM7.

A Figura 5 apresenta o diagrama de blocos do núcleo do processador ARM7, o qual foi retirado do manual técnico do ARM7. Fazendo uma breve análise sobre o diagrama de blocos apresentado, é possível reparar nos seguintes detalhes: multiplicador 32x8, unidade lógica aritmética, contador de programa, decodificador de instruções, decodificador de instruções *Thumb*, controle de debug, dentre outros.

O acesso à memória nos processadores ARM é estruturalmente Von Neumann, com barramento de 32 bits para acesso à memória do programa e dados ao mesmo tempo. Em arquiteturas RISC, o padrão de acesso à memória seria *Harvard*, que consiste em duas memórias separadas, uma memória para o programa e outra para os dados. Já na arquitetura Von Neumann existe uma única memória tanto para o programa quanto para os dados [17]

A estrutura de *pipelin*e dos processadores ARM7 consiste em três estágios: busca decodificação e execução, salientando que cada estágio é independente. Com o uso do pipeline, a maioria das instruções é executada em um ciclo de máquina. Na Figura 7 é possível visualizar o ciclo das tarefas do *pipeline* (IBIDEM).

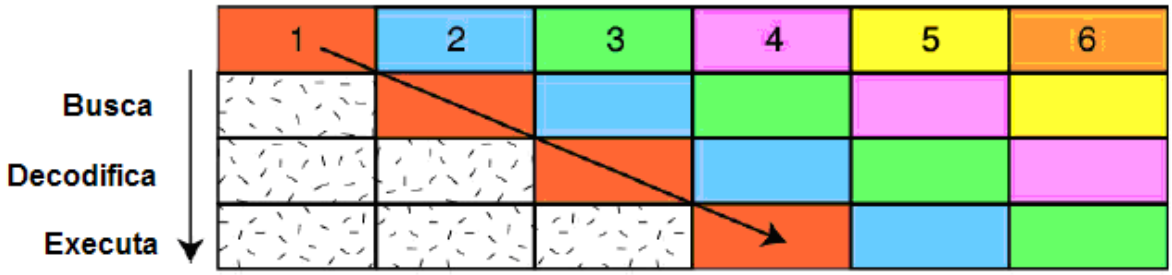


Figura 7 – Estrutura de pipeline do processador ARM.

Fonte: Souza *et al.* – *Microcontroladores ARM7*.

Todas as operações realizadas no processador ARM7 fazem o uso de registradores. Isso se deve ao fato de o processador possuir uma estrutura *load-and-store*. Na Figura 7 pode ser visualizado um pequeno exemplo de soma. A ideia nesse exemplo é a de realizar a soma de M1 com M2, e que o resultado esteja na variável M3. Para esse fim, faz-se necessário mover o valor de M1 para o registrador R1, e o valor de M2 para R2. Na sequência, é feita a operação de soma que mantém o valor no registrador quatro. Para acessar o resultado é necessário mover o valor do R4 para a variável M3 (IBIDEM).

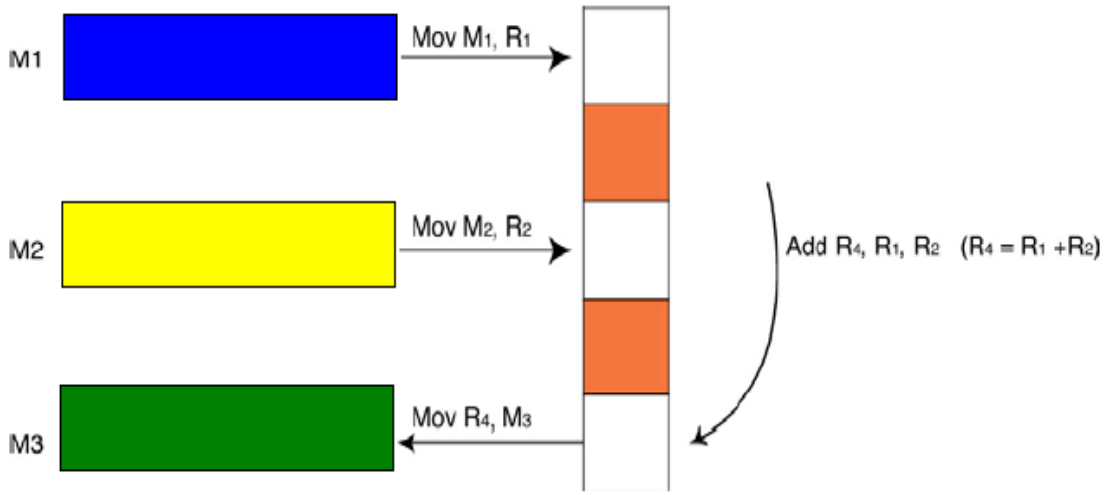


Figura 8 – Uso de Registradores do processador ARM.

Fonte: LPC ARM book.

Na Figura 8 é possível visualizar os sete modos de operação do processador ARM7. A diferença entre os modos está no uso dos registradores. Os sete modos de operação são:

• **Usuário**: Execução normal de programas;

• **Sistema**: executa rotinas privilegiadas do Sistema Operacional;

• **Supervisor**: Modo protegido para o Sistema Operacional;

• **IRQ**: Tratamento de interrupções comuns;

• **FIQ**: Tratamento de interrupções rápidas;

• **Abort**: Usado para implementar memória virtual ou proteção de memória;

• **Indefinido**: Suporta a emulação em software de co-processadores.

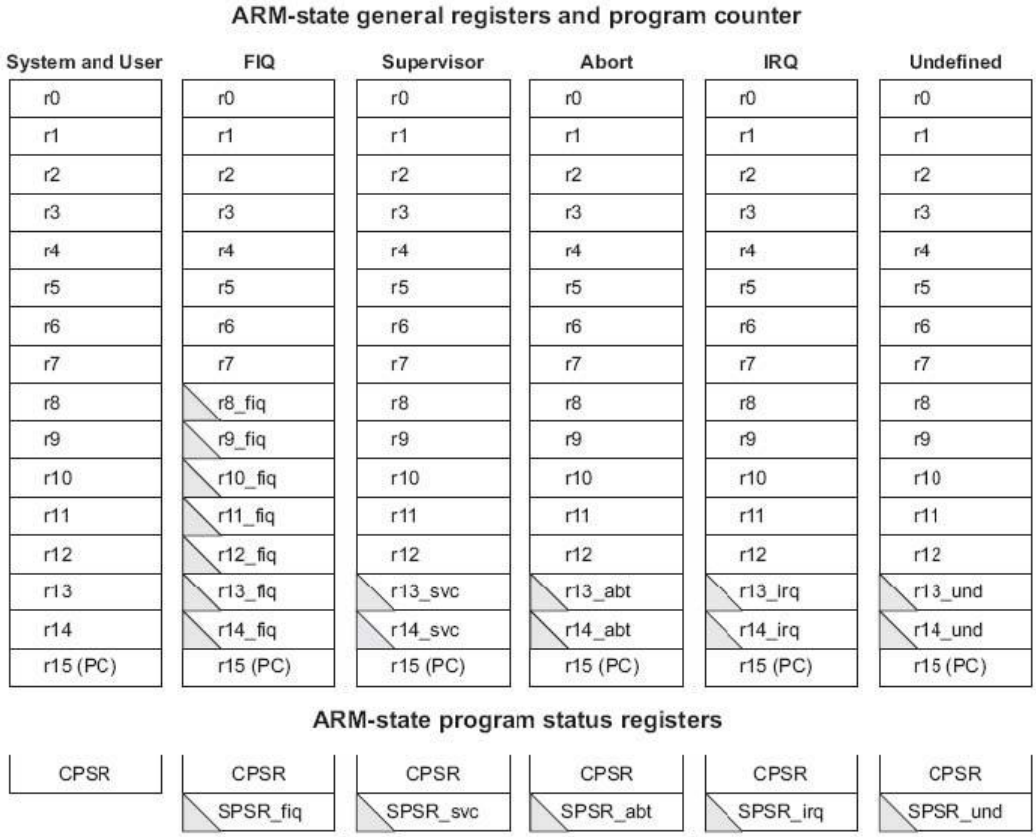


Figura 9 – Sete modos de operação do processador ARM7.

Fonte: Manual de referência do processador ARM7.

O núcleo do processador ARM7 possui dois modos de trabalho das variáveis: *bigendian* e *little-endian*. No modo *big-endian*, o *byte* menos significativo está na esquerda, já no modo little-endian, o byte menos significativo está na direita. O padrão da Philips é *littleendian*, mais comumente encontrado na maioria dos microprocessadores e microcontroladores. A Figura 10 ilustra a diferença entre os dois modos. Pode-se reparar que a demonstração realizada na figura

leva em conta um valor de quatro bytes que representam 32 bits (0-31), no modo *little-endian* o byte A está na direita e no modo *big-endian* o byte A está na esquerda.

Figura 10 – Arquitetura *ARM*.

Fonte: Manual de referência do processador ARM7.

2.9 AMPLIFICADORES OPERACIONAIS DE INSTRUMENTAÇÃO

Amplificadores de Instrumentação têm grande importância para o processamento de sinais em diversas aplicações, tais como: sistemas biomédicos, instrumentação industrial e equipamentos de análises científicas. As principais características são a alta taxa de rejeição a sinais de modo comum (CMRR), baixo ruído e baixa tensão de *offset*.

Os Amplificadores de Instrumentação podem ser implementados em topologias de um ou mais amplificadores operacionais com o intuito de obter características melhores que as apresentadas por um único amplificador operacional.

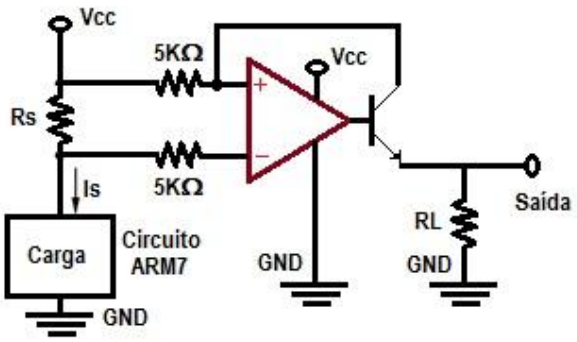


Figura 11 - Amplificador operacional de Instrumentação.

Fonte: Autor

$$A = \frac{V_{saída}}{V_{entrada}} \qquad (2.1)$$

Onde:

A é o ganho nominal do amplificador operacional,$V_{saída}$ é a tensão de saída e $V_{entrada}$ é a tensão de entrada.

2.10 **FONTES DE ENERGIA QUÍMICA – BATERIA DE LÍTIO**

As baterias de íon-lítio, hoje em dia, são bastante populares. É possível encontrá-las em laptops, PDAs, telefones celulares e iPods. Elas são tão comuns porque, proporcionalmente, são as baterias recarregáveis com maior capacidade de armazenamento de energia, atualmente existentes.

As baterias de íon-lítio também têm aparecido nos noticiários ultimamente. O motivo? Ocasionalmente, elas pegam fogo. Embora isso não seja muito comum (apenas dois ou três conjuntos a cada milhão delas apresentam referido problema), quando acontece, a situação é bem grave. Em alguns casos, o índice de falha pode aumentar, obrigando os fabricantes a fazer um recall que lhes custa milhões de dólares.

As baterias de íon-lítio são populares uma vez que possuem várias vantagens importantes sobre as concorrentes, dentre as quais pode-se citar o fato de serem muito mais leves do que outros tipos de baterias recarregáveis do mesmo tamanho. Os eletrodos de uma bateria de íon-lítio são feitos de **lítio** e **carbono** leve. Além disso, o lítio também é um elemento altamente reativo, o que significa que é possível armazenar bastante energia em suas ligações atômicas, denotando uma **densidade de energia** muito alta para essas baterias.

Figura 12 – Batéria de lítio.

Fonte: Autor

2.11 **GRAFO DE FLUXO DE CONTROLE – CFG**

Neste tópico apresenta os fundamentos teóricos em que foi baseado o desenvolvimento da ferramenta de estimação de consumo de energia – PESTI. Os trechos escritos foram retirados do trabalho desenvolvidos em [2].

2.11.1 Definição de Grafo

É um modelo matemático que descreve objetos interconectados e suas respectivas relações. Um grafo é um conjunto de vértices (ou nodos ou nós) conectados por arestas (ou arcos ou links). Os vértices de um grafo podem ser tratados como simples objetos, possuindo nomes e atributos como dados e os próprios links.

Podem ser considerados como um modelo geral, formal, rigoroso e completo em sua teoria, o que torna possível sua utilização para especificação de vários tipos de dados, pois a maioria dos problemas no domínio da Ciência da Computação, Redes, Sistemas Distribuídos, podem ser formulados e modelados através de grafos [31].

Formalmente, um grafo $G = (V, A)$ é um par ordenado formado por dois conjuntos finitos, de vértices V e de arcos A. Por sua vez, cada arco é denotado pelo par de vértices por ele ligados, isto é, $a = (V_1, V_2)$, onde V_1 e V_2 são dois vértices ligados pelo arco a.

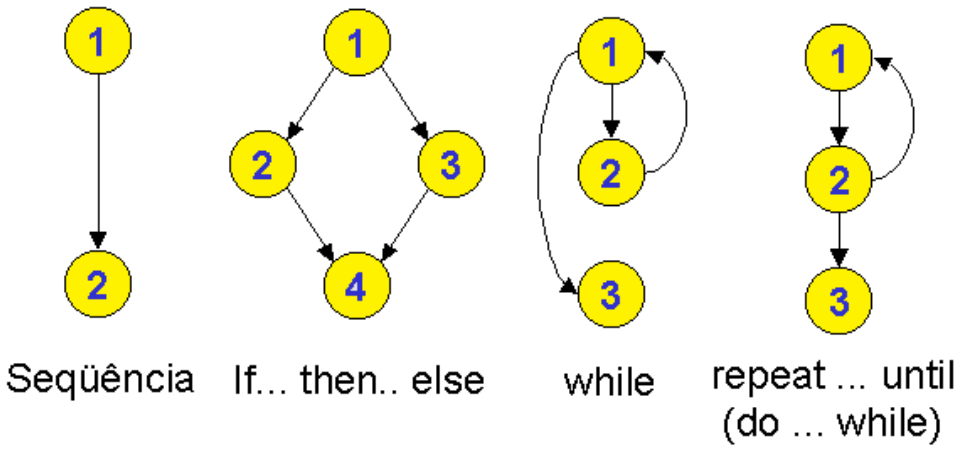


Figura 13 – Exemplos de um grafo de fluxo controle fundamentais.

Fonte: Autor

2.11.2 Consumo de Energia devido ao Software

Neste tópico, apresentam-se os conceitos e modelos matemáticos utilizados para determinar o consumo de energia de um sistema embarcado devido ao seu componente de software.

O consumo de energia devido ao software é descrito por um modelo que representa o consumo de energia de cada uma das instruções do conjunto de instruções do processador. Esse modelo está baseado no fato de que o software determina sobre o processador um padrão dinâmico de execução [24].

Uma instrução consome energia de duas formas: quando executada, uma instrução provoca alterações no estado interno do processador, tais mudanças determinam um padrão de consumo ao hardware, o chamado custo básico de instrução [24]; a segunda forma ocorre conforme os operandos envolvidos na execução da instrução. Eles fazem com que a instrução provoque mudanças nos registradores e acessos à memória, gerando chaveamento de circuitos internos e barramentos, ou seja, consumo dinâmico.

Logo, o consumo de uma instrução é determinado pelo seu custo básico adicionado do custo das transições e acessos à memória gerados devido aos operandos envolvidos. Em [13], o custo dinâmico é modelado como fatores de sensibilidade da instrução ao contexto. Os fatores de sensibilidade podem ser: número do registrador, valor do registrador, valor imediato presente na instrução, endereço do operando e valor do operando.

$$A = E_i = b_i + \sum_i a_{ij} \, . \, n_{ij} \tag{2.2}$$

A Equação 2.2 modela o consumo de energia de uma instrução.

Onde E_i é o consumo de energia da instrução i, b_i é o custo básico da instrução i, a_{ij} é o coeficiente de sensibilidade da instrução i ao fator j e n_{ij} é o número de 1's presentes no fator j.

A caracterização do consumo de energia de uma instrução é feita realizando-se a execução da mesma dentro de um loop infinito e aferindo o consumo gerado pelo processador [13]. Esse método, entretanto, não captura o consumo de energia devido à mudança de contexto ou chaveamento do circuito, pois tal efeito ocorre quando duas instruções são executadas de forma consecutiva. Seria razoável pensar que a soma das medidas individuais de duas instruções fosse igual ao consumo medido quando ambas são executadas de forma consecutiva. Porém, [13] demonstra que existe diferença entre tais valores. A diferença existente

corresponde aos efeitos gerados pela última transição interna referente à primeira instrução e a primeira transição interna referente à segunda instrução. Essa diferença de consumo é chamada de inter-instruction cost ou consumo inter-instrucao [24].

A Equação 2.3 modela o consumo de energia de um programa.

$$E_p = \sum_i E_i + \sum_{ij} O_{ij} \cdot N_{ij} \quad (2.3)$$

Onde E_p é o consumo de energia associado ao programa p, E_i é o consumo da instrução i, O_{ij} é o consumo *inter-instruction cost* associado ao par de instruções i e j e N_{ij} é o número de ocorrências do par de instruções i e j no programa p.

Os tópicos discutidos até agora, quanto ao consumo de energia devido ao software, partem do pressuposto que o processador executa apenas uma instrução por vez. Em arquiteturas onde os processadores são aparelhados com pipeline, o computo do consumo de energia de uma instrução segue o mesmo princípio básico descrito anteriormente, entretanto com uma leve modificação. Nesse caso é preciso modelar o consumo de energia de uma instrução em cada estágio do *pipeline* e, o somatório de tais consumos, resulta no consumo de energia da instrução.

TomemosE_{ijk} como o consumo de energia da instrução I_k no estágio j do *pipeline*.

Assumindo um processador com um *pipeline* de cinco estágios e o seguinte conjunto de instruções I_1, I_2, I_3, I_4 e I_5. Quando todo o pipeline estiver cheio, cada uma das cinco instruções ocupará um respectivo estágio no *pipeline*. Dessa forma, após um ciclo de *clock* a energia consumida é dada pela Equação 2.4.

$$E_{ciclo} = E_{5I_1} + E_{4I_2} + E_{3I_3} + E_{2I_4} + E_{1I_5} \quad (2.4)$$

Analogamente ao exposto pela Equação 2.3, o consumo de uma instrução é definido pela sua execução em todos os estágios do *pipeline*. Logo, o consumo de energia de uma instrução num processador que possui pipeline é dado pela Equação 2.5 [24]:

$$I_{Kcons} = \sum_j E_{ijk} \quad (2.5)$$

O efeito inter-instrução pode ser modelado ao longo do *pipeline*. Assim como existe uma fusão para a execução das instruções no tempo, existe também uma fusão para o consumo de energia. Esse modelo pode sofrer abalos quando há a ocorrência de conflitos no *pipeline*. Os conflitos podem ser de estrutura, dados ou controle e inserem no consumo uma variante que

depende do contexto. Quando se considera a não ocorrência de conflitos, o modelo de consumo de energia empregado é o mesmo que o utilizado para arquiteturas sem *pipeline*. A ocorrência de conflitos estabelece consumo e varia segundo o tipo de conflito ocorrido.

A Equação 2.6, proposta por [24], modela todos esses aspectos levantados.

$$E_p = \sum_i B_i \; x \; N_i + \sum_{i,j} O_{ij} \; x \; N_{ij} \; + \sum_h E_h \tag{2.6}$$

Onde E_p é o consumo de energia do programa p, B_i é o consumo básico da instrução i, N_i é o número de instâncias da instrução i, O_{ij} é o consumo inter-instrução do par de instruções i e j, N_{ij} é o número de ocorrências consecutivas do par de instruções i e j, e E_h é o consumo associado ao contexto de execução h, por exemplo: paradas de *pipeline* e falta de cache.

CAPÍTULO III

TRABALHOS CORRELATOS

Neste tópico apresentam-se os trabalhos relacionados em que foi baseado o desenvolvimento da ferramenta de estimação de consumo de energia – PESTI. Os trechos escritos foram retirados do trabalho desenvolvidos por Carlos Mar [2].

Como a parte deste trabalho é estender a Ferramenta PESTI, os conceitos apresentados são usados para ambas as abordagens, porém com uma diferença da abordagem realizada em [2], que atende cenários probabilísticos unitários, e nesta nova abordagem os múltiplos cenários probabilísticos.

O consumo de energia em um sistema embarcado pode ser avaliado pelos seus componentes de *software* ou *hardware*. Nesta seção serão apresentados trabalhos relacionados à questão de estimativa de consumo de energia devido ao *software*.

As estimativas de consumo de energia de um sistema embarcado podem ser realizadas através de duas abordagens básicas: as baseadas na simulação de instruções e as baseadas na simulação de *hardware* [13]. Simulações de *hardware* são mais complexas, entretanto seus resultados são mais precisos. Elas podem ser baseadas em modelos matemáticos do processador ou em modelos *RTL-Register Transfer Level*. Desvantagens desse tipo de abordagem são: o alto custo computacional e de implementação do modelo, a pouca flexibilidade proporcionada e a necessidade de especialização do projetista nos detalhes técnicos de implementação do *hardware*.

Abordagens baseadas em simulação de instruções, como o trabalho desenvolvido em [3], que consistem em caracterizar o modelo de consumo de energia do processador, segundo o consumo de energia associado às suas instruções e a arquitetura do *hardware*. Sua desvantagem é que não possuem nenhuma especificação formal de seu mecanismo de computação interna. Entretanto, possuem vantagem de serem mais flexíveis que os mecanismos de simulação por hardware, permitindo um maior nível de abstração e não necessitando de que o projetista seja um especialista na plataforma alvo sob a qual o sistema irá executar. O microcontrolador AT89S8252 da família 8051 foi à plataforma embarcada escolhida para realizar as medições.

Em [5], desenvolveu um modelo de desempenho de consumo de energia para aplicações em plataformas embarcadas, utilizando código ANSI-C transformado em redes de *Petri* Temporizadas, originando uma rede chamada de *Power Petri Net*. Além disso, foi

implementada uma ferramenta que simulou esse modelo e realizou a estimativa de consumo de potência do modelo através de simulações estocásticas e o código foi embarcado no microcontrolador AT89S8252 da família 8051 em que foram realizadas as medições mencionadas no trabalho. Na metodologia de medições reais deste trabalho, não se encontrou nenhuma menção quanto às condições ambientais de ensaio e seu circuito de medição foi determinado de forma empírica.

Segundo [4] que realizou um trabalho para avaliar o consumo de energia em sistemas embarcados e de tempo real nas fases iniciais de projeto. Desenvolveu uma ferramenta chamada CPN *Tools*, comparando com outra ferramenta chamada ALUPAS para o suporte estocástico, além de calcular o consumo de energia, o tempo de execução das instruções com o código escrito em ANSI-C, o trabalho propôs um modelo de eventos discretos temporizados capaz de representar a linguagem *assembly* e ANSI-C, utilizando uma plataforma ARM, porém o mesmo não relata formalmente a forma de medição e o hardware utilizado para tal e não se encontrou nenhuma menção quanto às condições ambientais de ensaio.

Em [2], utilizou-se a linguagem ANSI-C e um modelo probabilístico em grafo de fluxo de controle CFG para estimar consumo de energia, além de medir o tempo de execução do código. E para dar suporte à abordagem, foi desenvolvida uma ferramenta em nível de algoritmo chamada PESTI, além de ser simulado probabilisticamente usando Método de Monte Carlo, tornando possível aumentar a velocidade de simulação dos cenários, porém somente é possível avaliar um cenário por vez e outro ponto que deve ser ressaltado é que não foram realizadas nenhuma comparação com valores de consumo de energia reais medidos.

3.1 **SIMULAÇÃO DE HARDWARE**

Em [7], propôs um *framework* para análise e otimização de potência baseado em arquitetura. O objetivo do framework é permitir que o projetista faça uma exploração do espaço de projeto utilizando ferramentas de alto nível durantes as fases iniciais do desenvolvimento do sistema. O modelo de potência proposto é baseado em estruturas comuns presentes em processadores superscalar. Trabalhos desenvolvidos em [14] e [9] e que são baseados em abordagens RTL. O primeiro apresenta um conjunto de técnicas para acelerar a tomada de estimativas de potência em projetos de larga escala. O segundo propõe uma abordagem probabilística para o modelo de dissipação de potência do circuito. Nesse modelo, as atividades de entrada e saída do circuito são modeladas por polinômios. Dessa forma, os resultados podem rapidamente ser computados. Seus resultados demonstraram um ganho em velocidade de

simulação sobre modelos baseados em estimativas de nível lógico, entretanto foram menos precisos.

3.2 SIMULAÇÃO DE INSTRUÇÕES

As abordagens baseadas em simulação de instruções consistem na caracterização do modelo de consumo de energia do processador segundo o consumo de seu conjunto de instruções. Já em [24], que propôs um método para caracterizar o consumo de energia de um processador através da caracterização do consumo de suas instruções, realizou-se um conjunto de experimentos que consistiram, basicamente, em executar programas em um loop infinito no dispositivo e medir a corrente que trafegava no circuito. Os resultados mostraram-se satisfatórios para o conjunto de processadores avaliados. A partir de então outros trabalhos como [12], [27] e [28] surgiram seguindo sua análise baseada na abordagem de simulação de instruções. Em [13], é proposto um método baseado em caracterização de consumo de instruções para processadores que possuem pipeline. A principal vantagem desse tipo de abordagem é que a caracterização do consumo de energia pode ser estabelecida a partir de medições realizadas no processador de interesse, sem a necessidade de se ter conhecimento detalhado a respeito deste.

A caracterização do consumo de um dispositivo com base no seu conjunto de instruções requer que o modelo de consumo definido represente alguns aspectos, dentre eles o chamado efeito inter-instrução. Caracterizar esse efeito no modelo de instruções o torna complexo, pois é necessário avaliar e modelar as combinações possíveis do conjunto de instruções em pares. Arquiteturas como superscalar e VLIW tornam-se muito complexas devido a essa questão. Nos Trabalhos de [9] e [16], propuseram mecanismos para reduzir essa complexidade sem introduzir grandes perdas de precisão ao modelo. Em [9], a proposta é agrupar instruções que possuam consumo semelhante.

Em [3], propôs dois modelos baseados em redes de Petri coloridas para estimar o consumo de energia de sistemas embarcados devido ao software. Ambas os modelos são propostos para realizar simulação e análise. Além disso, os modelos operam com base na descrição do conjunto de instruções da plataforma alvo na forma de Redes de Petri Colorida. O primeiro modelo funciona de forma determinística. Nesse modelo, os desvios são definidos pelos dados que alimentam o código, não ocorrendo à exploração de cenários de execução em tempo de simulação. O segundo modelo propõe uma simulação probabilística do fluxo de execução das instruções. Dessa forma, consegue-se explorar um espaço maior de possíveis

comportamentos do código. Para dar suporte aos modelos propostos, foram desenvolvidas ferramentas de simulação e análise. A simulação esbarrou em um problema de performance devido à escolha da ferramenta de simulação. Existem alguns problemas abertos, como a dependência de desvios, o que leva a uma questão de modelagem de probabilidades dependentes. Outro ponto passível de propostas de melhorias é a exploração do espaço de fluxos de execução, que é um problema de ordem combinatorial.

O presente capítulo apresentou uma revisão da literatura a respeito de técnicas, ferramentas, métodos e modelos para estimar consumo de energia devido ao software. O trabalho apresentado nesta dissertação é uma extensão do modelo probabilístico proposto por [3]. Em relação aos outros trabalhos, o diferencial deste é a criação de um modelo simples para representar fluxo de execução de programas e a implementação de uma ferramenta de simulação que gera vetores de testes de maneira dinâmica por meio de probabilidades. Em relação ao trabalho de Oliveira Junior [3] o diferencial está na linguagem de programação abordada, que é ANSI C, e no aumento da velocidade do processo de simulação em função da ferramenta implementada. Por se tratar de uma proposta que tem como objetivo elevar o grau de abstração do problema abordado, os resultados apresentados tendem a ser menos precisos do que os trabalhos anteriores.

O próximo capítulo apresenta o modelo proposto nesta dissertação, que é baseado em Grafo de Fluxo de Controle para representar sistemas embarcados escritos em linguagem ANSI C.

CAPÍTULO IV

MODELO BASEDO EM GRAFO DE FLUXO DE CONTRLOE

Este capítulo descreve o modelo probabilístico baseado em grafo de fluxo de controle para representar sistemas embarcados escritos em linguagem ANSI C. O modelo proposto é uma extensão do modelo probabilístico proposto em [3]. Em seu modelo, o fluxo de execução de um programa escrito em assembly é modelado por uma Rede de Petri Colorida e a execução de uma instrução é modelada pelo disparo de uma transição da rede. Vetores de teste não são criados durante a simulação, pois são gerados dinamicamente através de modelos randômicos de simulação implementados como funções de probabilidade. Os valores aplicados à função de probabilidade são extraídos do código fonte anotado. O critério de parada da simulação é monitorado por um processo estatístico que implementa o Método de Monte [11]. Genericamente falando, em uma simulação de Monte Carlo, um modelo estocástico do sistema é simulado iterativamente até que um padrão de qualidade da métrica alvo de avaliação é alcançado. O principal propósito do Método de Monte Carlo é determinar a parada da simulação garantindo sua exatidão sem executar ciclos de simulação desnecessários.

O objetivo do modelo proposto é representar e avaliar o fluxo de execução de um programa escrito em linguagem ANSI C. Esse modelo representa os custos do tempo de execução, de consumo de energia e os cenários probabilísticos, aos quais o código ANSI C será submetido pela simulação

4.1 FORMALIZAÇÃO DO MODELO

Para fins de análise, o mecanismo utilizado para representar as estruturas de um programa ANSI C foi implementado como um grafo de fluxo de controle. Esse grafo é composto por um conjunto de vértices e arestas, no qual cada vértice representa uma estrutura ANSI C e as arestas representam os elos entre essas estruturas. Esta definição formal para grafo de fluxo de controle é utilizada neste trabalho, e afirma que os vértices são classificados em 4 (quatro) categorias: sequencial, desvio condicional, repetição e escolha. As próximas subseções apresentam e formalizam cada uma dessas quatro categorias [2].

4.2 GRAFO DE FLUXO DE CONTROLE

Portanto, fazendo uso do modelo proposto podemos representar um programa P (ANSI C) como grafo de fluxo de controle da seguinte forma: CFG = (V,E, s), onde:

a) V = *({Seq;CondBranch; LoopD; LoopI;Choice})* é o conjunto de vértices que representam os blocos de comandos do programa P;

b) E = *({Edge})* é o conjunto de arestas que conectam os vértices contidos em V;

c) s = *Bstart* é o bloco de entrada do grafo de fluxo de controle.

4.3 PADRÃO DE ANOTAÇÃO DO CÓDIGO ANSI-C

O padrão de anotação de código foi definido como forma de permitir o desenvolvimento de ferramentas automáticas para dar suporte ao modelo proposto. O propósito deste padrão é fornecer ao projetista umas notações probabilísticas claras e não ambígua.

Com esta notação o projetista é capaz de indicar sob quais cenários probabilísticos seu código deve ser simulado.

Desta forma as seguintes anotação são definidas:

Anotação	Estrutura ANSI C
@LoopMax	For
@LoopProb	While e Do While
@BranchProb	If
@Choice	Switch

Tabela 2 – Correspondência entre Anotações e Estruturas ANSI C.

Fonte: Autor

a) /* @LoopMax(exectimes) */

b) /* @LoopProb(P1,...,Pn) */

c) /* @BranchProb(P1,...,Pn) */

d) /* @ChoiceProb(P1,...,Pn) */

As anotações são incluídas dentro do código do programa da mesma forma que os comentários. Elas são ignoradas pelo compilador e podem ser reconhecidas por qualquer mecanismo automático que tenha a intenção de capturar seu sindicado. A anotação @LoopMax

é utilizada para indicar o número máximo de iterações de repetições determinísticas. @LoopProb é utilizado para repetições onde o número máximo de interações não é conhecido à priori. Nesse caso uma condição lógica determina a parada da repetição.

Uma condição lógica é composta por um conjunto de cláusulas lógicas. Desta forma, os parâmetros que vão de P1 a Pn representam a probabilidade de que cada cláusula lógica seja verdadeira. @BranchProb define a probabilidade de um desvio condicional ser tomado ou não. Seus parâmetros são definidos da mesma forma como em @Loop- Prob. Finalmente, a anotação @ChoiceProb é utilizada para informar a probabilidade de ocorrência de cada escolha de um conjunto de escolhas. Para cada uma dessas escolhas esta anotação deve ser utilizada sobre uma estrutura ANSI C. A Tabela 2 mostra a correspondência entre as anotações e as estruturas ANSI C.

O trecho de código abaixo mostra o código de um programa ANSI C anotado com a notação proposta neste trabalho. O código foi anotado com dois tipos de anotações, @LoopMax e @BranchProb. Essas anotações determinam o número de vezes que cada repetição executará e a probabilidade da expressão relacional numbers[i] > numbers[j] ser verdadeira, respectivamente, mostra no trecho se programa a seguir.

```
/* @LoopMax(49) */
for (j =1; j<=i; j++)
{
/* @BranchProb (0.4946) */
 if (numbers [i] > numbers [j])
   {
temp = numbers [j-1] ;
numbers [j-1] = numbers [i] ;
number [i] = temp;
   }
}
```

CAPÍTULO V

FERRAMENTA PESTI

Este capítulo apresenta a ferramenta PESTI modificada além de seus conceitos e seu padrão de anotação a nível algorítmico, desenvolvida para dar suporte ao modelo baseado em Grafo de Fluxo de Controle proposto.

5.1 VISÃO GERAL DA FERRAMENTA

A ferramenta é baseada em um modelo probabilístico que representa um gráfico de controle de fluxo do sistema, que durante a execução da sua simulação é executada através de sua entrada, que é um código escrito em ANSI C anotado.

A ferramenta PESTI se divide em três:

a) Implementar o modelo probabilístico baseado em Grafo de Fluxo de Controle;

b) Considerar sistemas embarcados escritos em linguagem ANSI C;

c) Otimizar os mecanismos de simulação e, consequentemente, aumentar a velocidade do processo de simulação.

É composta de três componentes:

a) Tradutor - O componente de tradução é um conjunto de *scripts Python* cuja função é materializar o fluxo de controle de um código ANSI C em forma de um grafo direcionado. O componente recebe como entrada um código ANSI C anotado, realiza um processo de tradução que identifica as estruturas da linguagem e suas conexões e converte-as em vértices e arestas de um grafo, materializando em um arquivo XML, e apoiada por um modelo de consumo de energia (MCE) e o compilador é utilizado com a finalidade de converter o código ANSI C em código assembly.

b) Simulador - Projetado com o objetivo de aumentar a velocidade do processo de simulação e fornecer estimativas de consumo de energia confiáveis a respeito do comportamento do código.

c) Analisador – Interpreta os dados geradores no formato CVS, de modo a serem tratados e interpretados por diversos programas comerciais.

Na Figura 14 é mostrada a arquitetura da ferramenta PESTI.

.

Figura 14 – Arquitetura da Ferramenta PESTI

Fonte: Autor

5.2 **FERRAMENTA PESTI ESTENDIDA**

A ferramenta PESTI foi desenvolvida apenas com o intuito de estimar o consumo de energia em apenas um cenário, ou seja, é considerado apenas um valor de probabilidade anotado no código do programa.

A extensão da ferramenta trata, agora, com múltiplos cenários, ou seja, em vez de considerarmos apenas um valor de probabilidade, agora se pode incluir faixas de probabilidades. Isso daria uma ampla visão aos projetistas de sistemas embarcados, pois a estimativa agora gera vários perfis de consumo de energia.

Sua modificação segue o seguinte principio: cada execução de código gera arquivos de saída que devem ser analisados e interpretados para a geração de gráficos. Então a forma encontrada para a implementada foi a alteração no código fonte da ferramenta, de modo que a modificação atendesse aos critérios necessários a atender às faixas de probabilidades, e para cada valor probabilístico estimado deve ser gerado uma saída de dados, ou seja, cada faixa de frequência gera várias saídas de acordo com o intervalo de probabilidade e o número de *steps* atribuídos à ferramenta.

O principio de execução da Ferramenta é:

- Escolhe o código anotado;
- Executa o código na ferramenta;
- Atribui a faixa de probabilidade;
- Atribui o *step* de execução da faixa de probabilidade;
- Atribuir o número de iterações, o valor da taxa de confiança e fator de erro aceitável;
- Executar a ferramenta;

- Colher e analisar os resultados.

5.3 PROPOSTA DE UM NOVO PADRÃO DE ANOTAÇÃO DE CÓDIGO ANSI C

Um novo padrão de anotação de código foi definido, de forma a permitir o desenvolvimento de ferramentas automáticas para dar suporte ao modelo proposto. O propósito deste padrão é fornecer ao projetista múltiplas anotações probabilísticas.

Com esta notação o projetista é capaz de indicar sob quais cenários probabilísticos seu código deve ser simulado.

Desta forma as seguintes anotações são definidas:

Anotação	Estrutura ANSI C
@LoopMaxMC	For
@LoopProbMC	While e Do While
@BranchProbMC	If
@ChoiceMC	Switch

Tabela 3 – Correspondência entre Anotações e Estruturas ANSI C para Múltiplos Cenários

Fonte: Autor

a) / @LoopMaxMC(exectimes) */*

b) / @LoopProbMC(ProbSTART;ProbSTOP;STEP) */*

c) / @BranchProbMC(ProbSTART;ProbSTOP;STEP) */*

d) / @ChoiceProbMC(ProbSTART;ProbSTOP;STEP) */*

Para estas novas anotações que o projetista poderá incorporar dentro do código do programa como se fossem comentários, para o compilador ignorá-las e deforma uma forma coerente, essas anotações possam ser reconhecidas por qualquer mecanismo automático que tenha a intenção de capturar sua sintaxe.

5.3.1 Conceitos das novas Anotações proposta

- *@LoopMaxMC* - é utilizada para indicar o número máximo de iterações de repetições determinísticas.

- *@LoopProbMC* - é utilizado para repetições probabilísticas é definida pela faixa de probabilidade atribuída a anotação.
- *@BranchProbMC* - define a faixa de probabilidade de um desvio condicional ser tomado ou não. Seus parâmetros são definidos da mesma forma como em *@Loop- Prob.*
- *@ChoiceProbMC* - é utilizada para informar a faixa de probabilidade de ocorrência de cada escolha de um conjunto de escolhas. Para cada uma destas escolhas esta anotação deve ser utilizada sobre uma estrutura ANSI C.

A seguir é mostrado um trecho de um código ANSI C anotado, no qual essas anotações determinam o número iterações e repetições para executará e a faixa probabilidade atribuída.

```
/* @LoopMaxMC(50) */
for (j =1; j<=i; j++)
{
/* @BranchProbMC (0.45; 0.55; 0.01) */
 if (numbers [i] > numbers [j])
   {
temp = numbers [j-1] ;
numbers [j-1] = numbers [i] ;
number [i] = temp;
   }
}
```

CAPÍTULO VI

CARACTERIZADOR DE CONSUMO DE ENERGIA

Este capítulo apresenta o projeto do caracterizador e resultados obtidos com a aplicação da ferramenta para fazer um comparativo com as estimadas com a ferramenta PESTI.

6.1 PROJETO DO CARACTERIZADOR

O projeto do caracterizador está baseado em monitor corrente elétrica consumida pelo processador ARM7 LPC2148, conforme a Figura 15, quando o mesmo executar o código adaptado e embarcado em seu núcleo de processamento, o processador está acoplado ao kit de desenvolvimento e posto em um arranjo de forma que a alimentação DC passasse por apenas um caminho, de acordo com a Figura 16.

Figura 15 – Adaptador soquete com processador ARM7 LPC2148.

Fonte: Autor

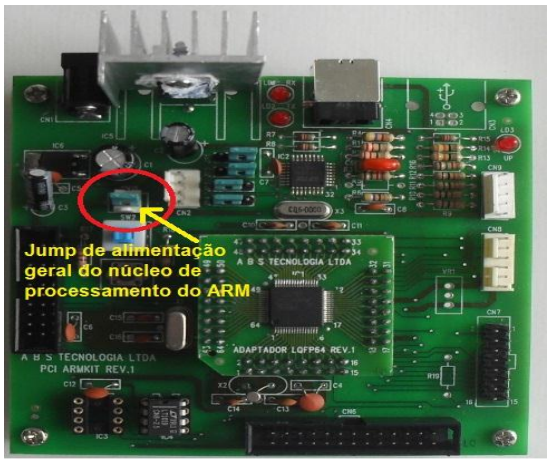


Figura 16 – Kit de desenvolvimento com processador ARM7 LCP2148 – fabricado pela NXP.

Fonte: Autor

O circuito de monitoramento de corrente com características *Shunt* é composto de um amplificador de instrumentação INA 138 e disposto conforme a Figura 15, de modo a dispor de um ganho de 20 e um resistor $R_s = 1\Omega$. Isso porque considerando a Lei de Ohm em que $V = R.I$, se $R_s = 1\Omega$, então o valor da corrente que atravessar R_s será igual a tensão medida em cima de R_s, grandeza necessária para o cálculo de energia.

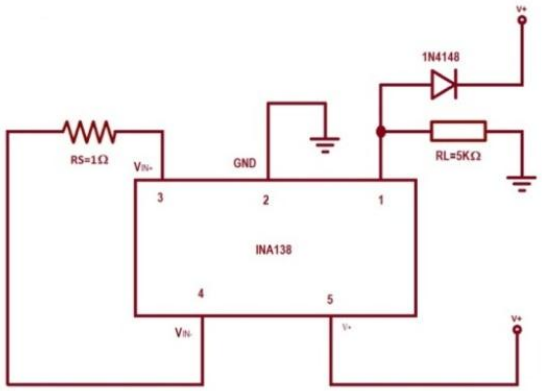


Figura 17 – Esquema de polarização do Amplificador Operacional de Instrumentação – INA138.

6.2 SOFTWARE DE DESENVOLVIMENTO DE FIRMWARE

A Keil Software desenvolveu a IDE MicroVision que permite o desenvolvimento de aplicações embarcadas em C e Assembly no mesmo código fonte.

MicroVision 3 é a interface de desenvolvimento e compilador oferecido pela Keil Software, também fabricante do kit de desenvolvimento MCB2130, através dessa ferramenta é possível criar os projetos, subdividir os projetos em unidades e gerar o código HEXA para ser usado no processador ARM. A IDE foi desenvolvida para ser executada sobre o sistema operacional Windows e conta com um compilador C e macro assembler (KEIL, 2008). Nessa ferramenta é necessário configurar o hardware que está sendo utilizado, assim como as variáveis de compilação, enquanto que o restante dos seus itens se assemelham a outras ferramentas gráficas de desenvolvimento.

Visualizando a Figura 18, que ilustra a IDE MicroVison, é possível observar a sua divisão gráfica de tarefas. Na parte superior encontra-se o menu principal e logo abaixo é possível ter acesso aos botões de acesso rápido. No lado esquerdo, tem-se como principal função manter aos olhos do programador a estrutura e organização do seu projeto. A parte central e maior é destinada a escrita do código fonte. Esse quesito possui vários recursos como: Minimizar e Maximizar as funções, adicionar pontos de parada do código fonte nas simulações, linhas de erros assinaladas.

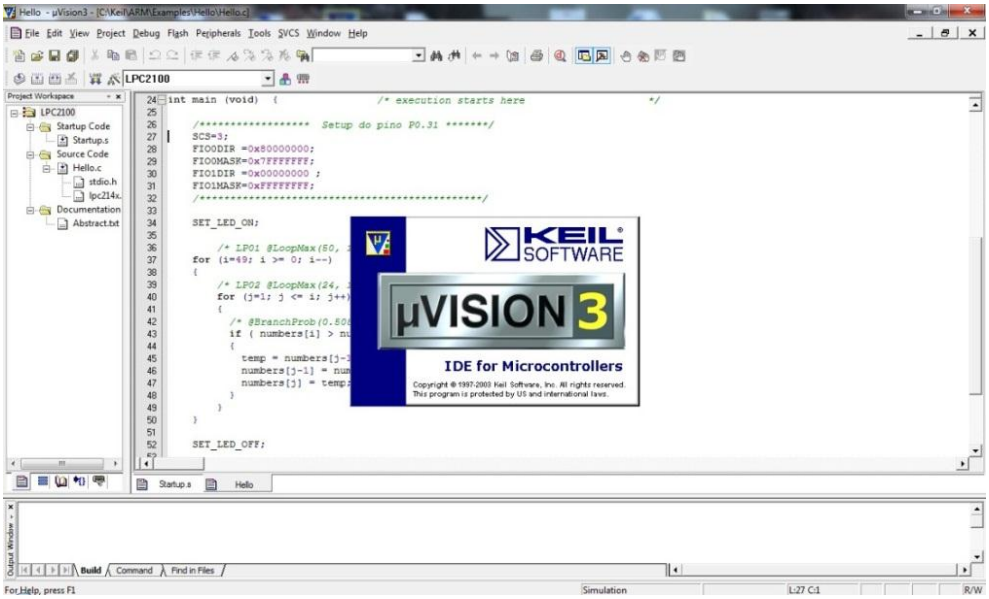


Figura 18 – IDE de desenvolvimento Micro Vision da Keil Software.

Fonte: Autor

6.3 SISTEMA DE MEDIÇÃO DE CORRENTE DE CONSUMO

O resistor R_s foi polarizado em um ponto estratégico do circuito de medição e em série com o *jump* de alimentação no Kit de Desenvolvimento de modo a monitorar o consumo de corrente do processador.

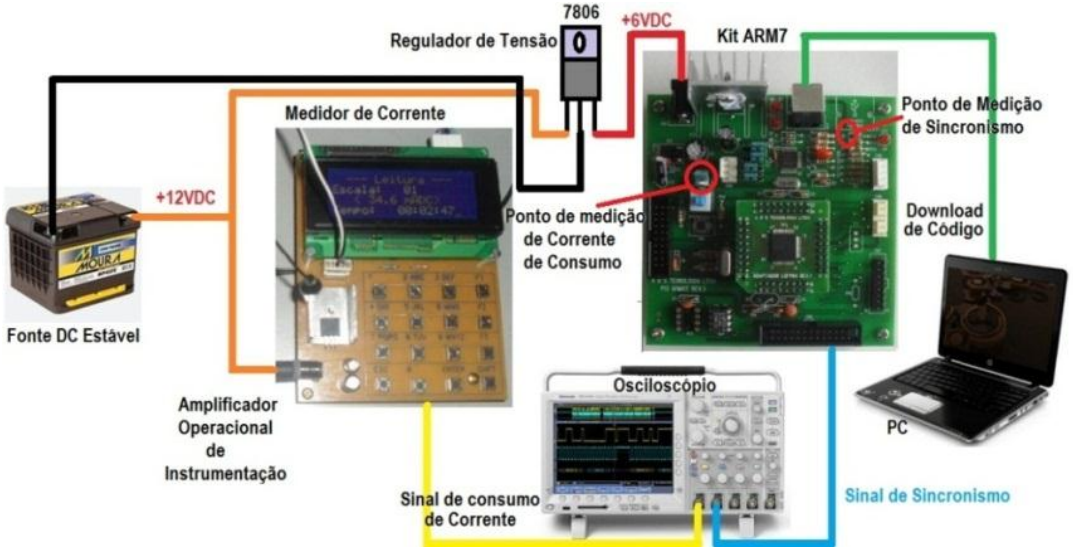


Figura 19 – Esquema do sistema de Caracterização e Medição de Consumo de Energia.

Fonte: Autor

O sistema de caracterização esquematizado na Figura 19 mostra todos os elementos pertencentes ao processo de medição de consumo de energia.

- ***Fonte Estável*** – De origem química para garantir a ausência do ruído de 60Hz, espúrio e a estática, tem a função de alimentação DC para o conjunto de hardware (Amplificador e o Kit ARM);
- ***Kit de medição Amplificador Operacional de Instrumentação*** – Projetado com a sensibilidade de variação de corrente consumida pelo processador ARM7 LPC2148 durante a execução dos testes;
- ***Regulador de Tensão 7806*** – Este componente se fez necessário por conta de segurança do circuito de medição e proteção do Amplificador operacional de instrumentação;
- ***Kit de desenvolvimento ARM7*** – Desenvolvido de acordo com a necessidade deste trabalho, com uma entrada específica para uma resistência *shunt*;
- ***Osciloscópio*** – Equipamento utilizado neste trabalho com um poderoso processamento de sinais e uma gama excelente de funções para tratamento e exibição de sinais;
- ***PC*** – Necessário para a compilação dos códigos ANSI C: Bolha, CRC e QURT, assim como o embarque dos mesmos códigos no processador ARM7.

6.4 ADAPTANDO UM AMBIENTE CONTROLADO PARA TESTES

Para a realização de ensaios e uma coleta de dados confiáveis, foi necessário criar um ambiente um tanto controlado, considerando certas variáveis, como: ruído de 60Hz, a temperatura do ambiente de testes, humidade ar e cargas eletroestática. No entanto não foi possível controlar com mais eficácia, mas foi possível monitorá-la a todo o momento durante a execução dos testes.

O ambiente ideal para ensaios do tipo realizado neste trabalho requer um investimento considerável, no entanto por falta de recursos, algumas considerações e recomendações foram levadas em conta para que fossem realizadas as medidas de consumo de energia no ARM7 com a presença mínima de variáveis elétricas que poderiam interferir nas e mascarar os resultados deste trabalho.

As mantas dissipativa, luvas e pulseiras antiestáticas também foram utilizadas para prover resultados mais confiáveis, e a utilização de um termo higrômetro para o monitoramento da humidade do ambiente e a temperatura.

Figura 20 – Termo Higrômetro – monitora a temperatura e a humidade.

Fonte: Site Minipa (2011).

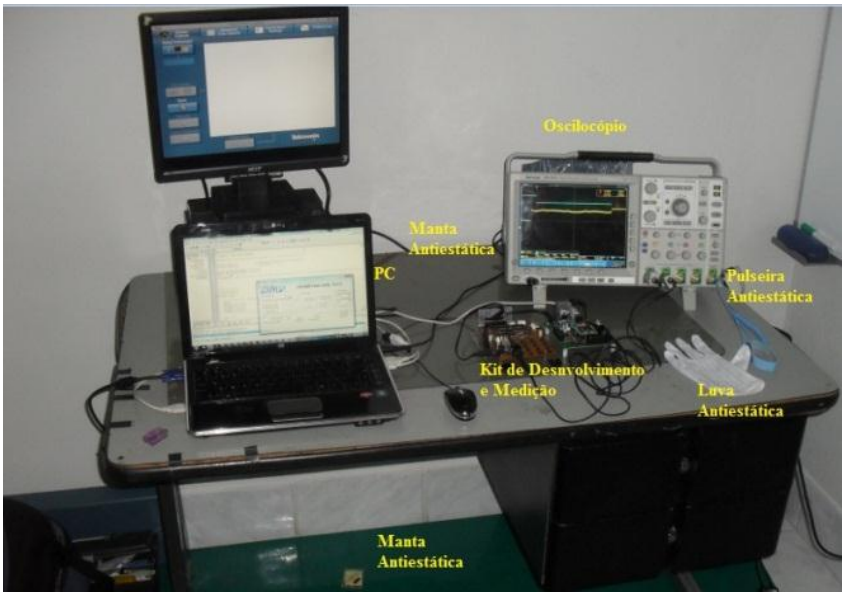


Figura 21 – Ambiente de Ensaios.

Fonte: Autor

CAPÍTULO VII

EXPERIMENTOS E RESULTADOS

Neste capítulo, apresentam-se as devidas considerações para a montagem do ambiente real de testes, medidas e equipamentos usados na execução dos testes, considerações para se analisar os resultados obtidos no *hardware* e com a aplicação da ferramenta PESTI nas estimativas de consumo de energia e tempo de execução dos algoritmos de Ordenação em Bolha, CRC e QURT.

Em um cenário ideal, esses experimentos foram executados sobre uma plataforma de hardware real através da execução de múltiplos cenários, da mesma forma que fora implementada em [3], com a diferença que em seu trabalho usa uma plataforma 8051. As métricas coletadas nos experimentos foram consumo de energia e tempo de execução.

Os objetivos destes experimentos foram:

- validar a abordagem proposta;

- comparar os resultados e tempo de simulação entre a ferramenta PESTI e o caracterizador para a mesma plataforma de hardware (ARM7).

Os experimentos foram divididos em 3 passos:

- Simulação dos algoritmos na ferramenta PESTI;

- Caracterização do cenário probabilístico para o algoritmo bolha, CRC e QURT;

- Caracterização dos códigos embarcados no processador ARM.

Os experimentos foram conduzidos sobre AMD Turion(tm) II Dual Core Mobile M500, 2.20GHz, 4GB de RAM, SO Windows 7 32 bits, SO Linux e no Kit de Desenvolvimento ARM o qual usa um processador ARM7 processador ARM7 LPC2148 Figura 15.

7.1 CONSIDERAÇÕES PARA ANALISAR OS RESULTADOS DOS TESTES

No circuito a seguir, considera-se uma análise bem coerente para se definir os resultados coletados durante os ensaios executados no processador AMR7 e sendo medido pelo circuito de medição de corrente, conforme a Figura 22.

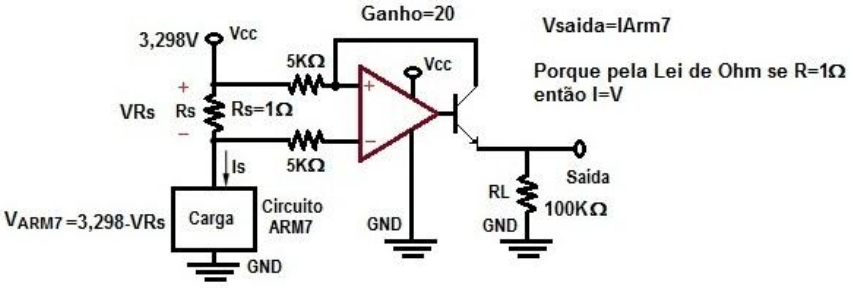


Figura 22 –Esquema do circuito de medição.

Fonte: Autor

Considerando $V_{Batéria} = 12{,}45VDC$ na bateria, e que é aplicada ao Kit de medição e também ao regulador de tensão (7806), componente responsável para reduzir a tensão aplicada ao Kit de desenvolvimento ARM, a fim de preservar a integridade de seus componentes eletrônicos, nesse caso o valor é reduzido de $V_{Batéria} = 12{,}7VDC$ Figura 23 para $V_{Kit\ ARM} = 6{,}05VDC$ Figura 25, que em seguida é novamente é reduzida para $V_{ARM7} = 3{,}302VDC$.

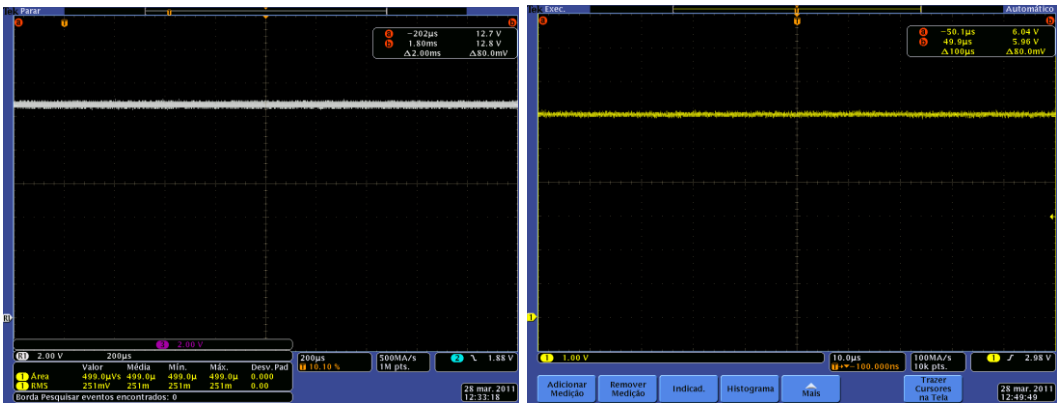


Figura 23 – $V_{Bateria}$ Medida na fonte química e $V_{Kit\ ARM}$ Medida no regulador de tensão do Kit ARM.

Fonte: Autor

Analisando o funcionamento do circuito considera-se uma pequena queda de tensão sobre o resistor R_s de valor igual a 1Ω. O motivo de se escolher o valor de 1Ω para R_s é, simplesmente, pelo fato de se adequar a Lei de Ohm equação 2.3, necessária para se calcular a potência equação 2.2 consumida pelo processador ARM7.

A forma de análise a seguir mostra matematicamente passa a passo como são consideradas as variáveis em questão necessárias para se encontrar a energia consumida pelo processador ARM em função do código embarcado e posto em execução.

Se equação 2.3 $R_s = 1\Omega$ e substituindo o valor de R_s, na Lei de Ohm, temos que $V = I$, então para qualquer que seja o valor de Vem R_s tem-se o valor da corrente consumida pelo processador ARM.

Considerando que o amplificador de instrumentação em seu arranjo tenha um ganho nominal de 20 vezes o valor de saída em relação à entrada, e deverão ser considerados nos resultados finais no cálculo do consumo de energia $E = P.\Delta t$ do processador.

$$P = V.I \qquad (2.2)$$

$$\mathrm{V} = \mathrm{R}.\mathrm{I} \qquad (2.3)$$

$$E = P.\Delta t \qquad (2.4)$$

7.2 FUNDAMENTOS TEORÍCOS PARA O PROCESSO DE MEDIÇÃO DE CONSUMO DE ENERGIA

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, o primeiro bloco da Figura a seguir, é um documento que descreve o protocolo de medição, Este protocolo descreve "o que, onde, como e a frequência" do processo de coleta de resultados, assim como a forma de armazená-los, o plano da análise e quem deverá executar cada tarefa.

Em [23], as tarefas necessárias para a realização dos experimentos podem ser observadas na Figura 24 a seguir.

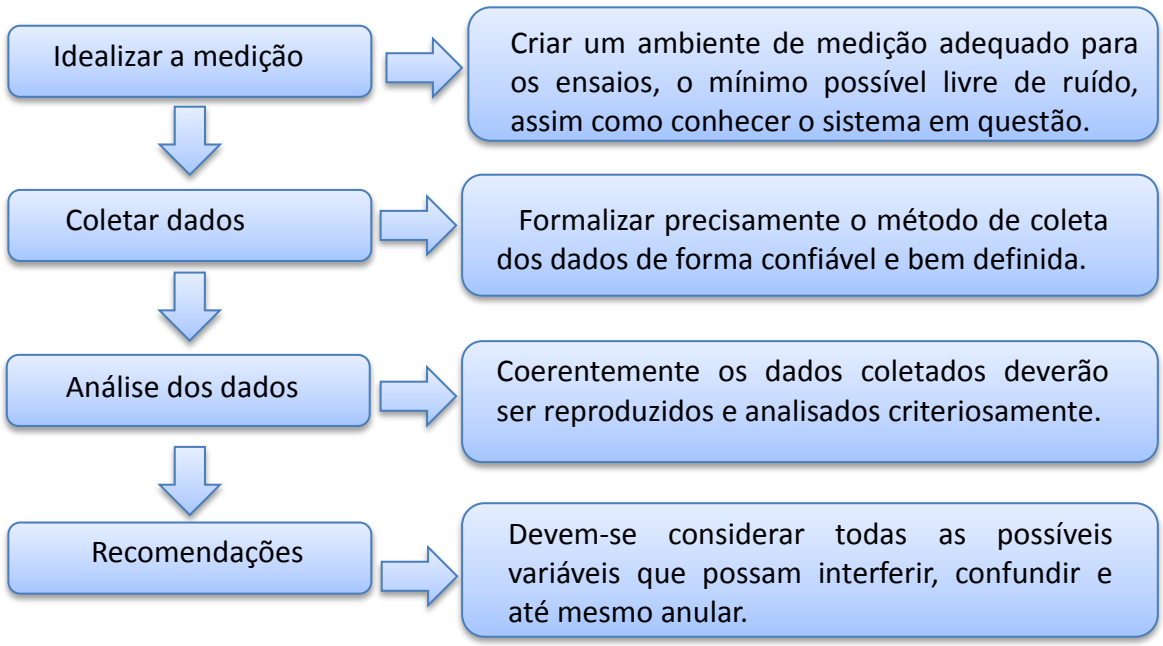


Figura 24 – Tarefas necessárias para a realização dos experimentos.

Fonte: Autor

7.3 HARDWARE USADO PARA O PROCESSO DE MEDIÇÃO DE CONSUMO DE ENERGIA

O projeto de monitoramento de consumo de energia devido ao código embarcado em um processador, neste caso o ARM, está baseado na escolha dos componentes com características *shunt*, ou seja, dispositivos que trabalham com sensibilidade a passagem de corrente elétrica.

O INA 138 *High-Side Measurement CURRENT SHUNT MONITOR*, componente desenvolvido pelo fabricante *Burr-Brown Product from Texas Instruments*, possui características particulares e essenciais para sua escolha e utilização neste projeto, tendo aplicações nas mais diversas áreas de sistemas eletrônicos e particularmente em sistemas embarcados para monitoramento de corrente de baixo valor nominal.

Na Figura 25, temos um circuito elétrico do monitor de corrente elétrica que será usado no projeto do caracterizador para mensurar o consumo da plataforma escolhida, no caso ARM, mostra uma resistência R_S que irá monitorar a corrente I_S durante a execução o código embarcado no processador ARM7, gerando uma tensão de saída V_O em cima da resistência R_L, a equação $V_O = \frac{I_S R_S R_L}{5K\Omega}$ que é tratada de forma coerente para obter uma relação do consumo de corrente necessária para este trabalho.

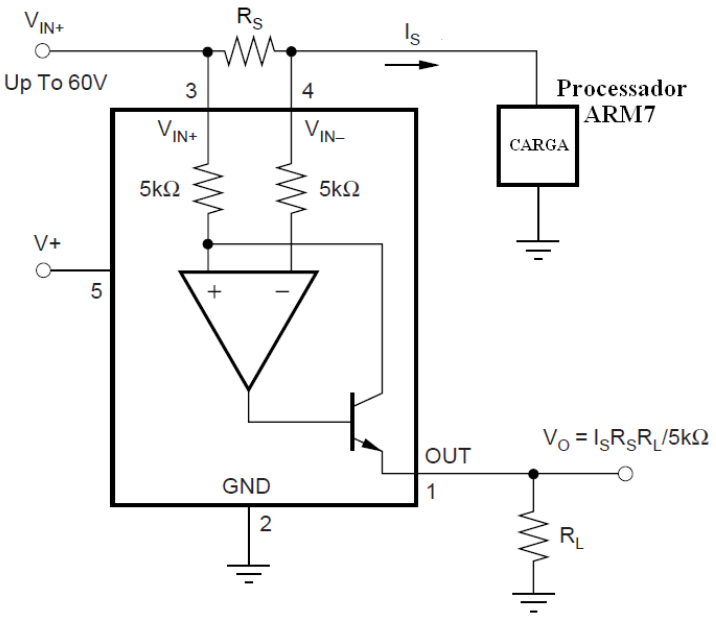


Figura 25 – Diagrama elétrico do monitor *Shunt* de corrente – caracterizador.

Fonte: Autor

7.4 **PROCESSO DE ENSAIO PARA MEDIÇÃO DE CONSUMO DE ENERGIA**

Em relação à técnica de medição, o sistema caracterizador tem que lidar com questões básicas:

- A definição da resistência *shunt* usado em conjunto com o amplificador de instrumentação,
- As questões do ambiente de testes,
- Adaptação do código C para serem embarcados no processador ARM7;
- A forma da taxa de amostragem do osciloscópio;

Em relação à resistência *shunt*, sua definição foi feita com base na lei de Ohm e na polarização do amplificador de instrumentação, de forma que não interferisse na medição do consumo de corrente e de forma coerente.

O ambiente de teste foi no máximo do possível controlado, sendo utilizados artifícios de projeto e de execução de testes controlados: a substituição da tensão AC da rede por uma batéria de carro de valor nominal de 12V aproximadamente, isso fez com que reduzisse altamente a possibilidade de interferência espúrios e ruído de 60Hz nas medições e também o uso de forma estreitamente obrigatória de acessórios antiestéticos (Luvas e Mantas Dissipativa) para a realização dos ensaios e manuseio das placas e equipamento.

Sobre a adaptação do Código C para upload no processador, essa foi trabalhosa, pois o código que fora gerado para o estimador não é possível rodar no processador diretamente, então foi feito de forma criteriosa o isolamento do trecho do código que se quer medir.

O osciloscópio utilizado para a medição foi o DPO 4034 Figura 26 fabricado pela Tektronics, cujas principais características são: Larguras de banda de 1 GHz, 500 MHz e 350 MHz, 4 canais de medição, taxas de amostragem de até 5 GS/s em todos os canais analógicos, comprimento de registro de 10 Megapoints, taxa de exibição de 50.000 formas de onda/segundo e porta de dispositivo USB 2.0 para controle do osciloscópio com o protocolo USBTMC.

E com relação à configuração do osciloscópio, as medidas foram realizadas de acordo com a necessidade do tempo de execução do código e o sinal mostrado na tela do osciloscópio em modo trigger e a função ativada de área de medição do mesmo.

O processo de medição consiste em:

- Compilar o código C adaptado para o ARM;
- Carregar o arquivo HEX gerado pelo Micro Vision usando o Flash Utility;

- Fazer o upload para o Chip LPC2148 ARM7 no kit de desenvolvimento;
- Configurar o osciloscópio no modo trigger;
- Executar o código acionando a tecla reset do Kit ARM;
- Capturar o evento de execução código visualizando na tela do osciloscópio;
- Gravar os dados lidos e fornecido pelo osciloscópio no pendrive;
- Armazenar os dados no PC.

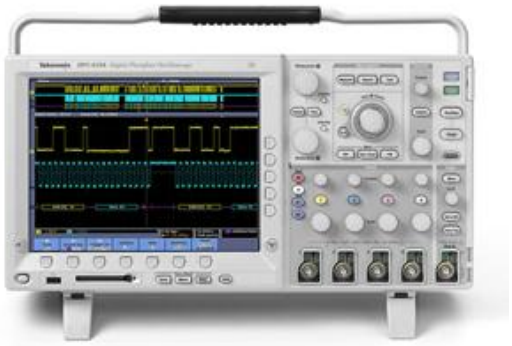


Figura 26 – Osciloscópio usado

Fonte: Autor

7.5 ANÁLISE DOS DADOS COLETADOS NO PROCESSO DE MEDIÇÃO DE CONSUMO DE ENERGIA

A análise dos resultados é um dos passos mais importante de qualquer trabalho, em que os dados coletados devem ter: coerência, obedecer aos critérios estabelecidos e mostrar o comportamento do evento em questão.

A Figura 27 mostra um dado coletado durante um dos ensaios de medição do código, o sinal em amarelo no canal 1 do osciloscópio é o valor da corrente que o processador ARM7 está consumindo durante a execução do código, já o sinal em azul no canal 2 do osciloscópio é o sincronismo e mostra o tempo de execução do código embarcado no ARM7.

As pontas de prova do osciloscópio foram fixadas de em pontos estratégicos do Kit ARM e da placa de medição de consumo de energia conforme a Figura 28.

Esse procedimento foi realizado para todos os ensaios realizados neste trabalho, sendo claro adequado a cada tipo de ensaio.

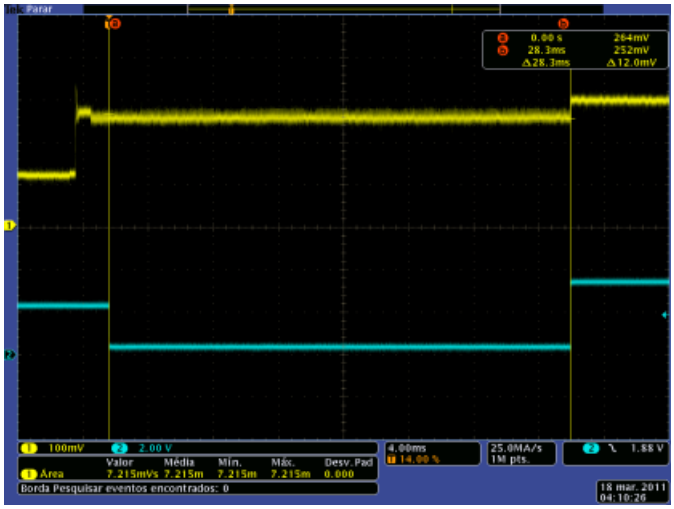


Figura 27 – Dado coletado em um ensaios.

Fonte: Autor

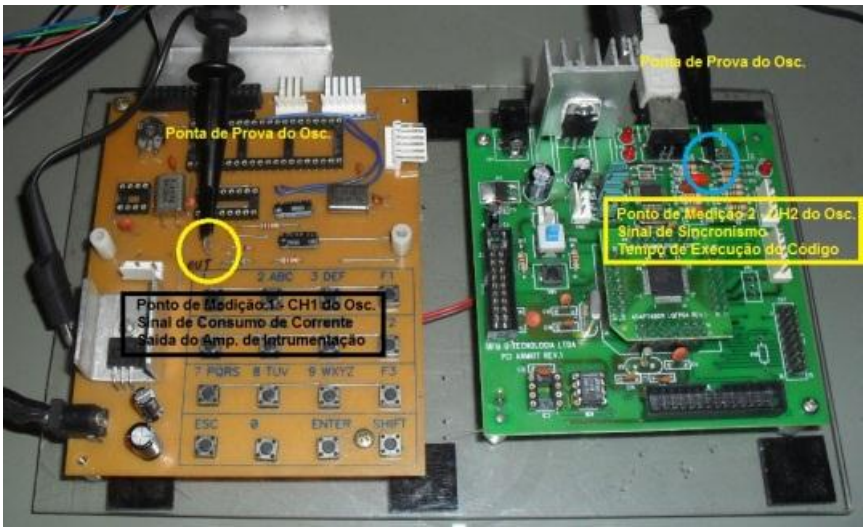


Figura 28 – Kits com as pontas de prova do osciloscópio fixadas em pontos estratégicos.

Fonte: Autor

7.6 CÓDIGOS ANSI C USADOS NOS ENSAIOS

Foram realizados experimentos para validar o processo de caracterização proposto neste trabalho usando códigos a fim de se explorar os padrões comportamentais dos mesmos e os experimentos que ilustram casos típicos de códigos em sistemas embarcados e em todos eles é feita uma análise de estimativa de consumo de energia.

Os resultados são apresentados através de valores médios para cada cenário e os algoritmos utilizados foram os desenvolvidos pela Motorola e são chamados de *Power Stone Benchmark*[21]. Esses algoritmos possuem a particularidade de terem sido desenvolvidos

especificamente com o objetivo de exercitar diferentes aspectos de consumo de energia em sistemas embarcados especificados em ANSI C.

7.6.1 Código BOLHA

O algoritmo *bubble sort*, ou ordenação por flutuação (mais conhecido como "bolha"), é um algoritmo de ordenação dos mais simples. A ideia deste algoritmo é percorrer o vetor diversas vezes, a cada passagem fazendo flutuar para o topo o maior elemento da sequência. Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível; daí vem o nome do algoritmo, mostrado no conjunto de Apêndice deste trabalho.

7.6.2 Códigos CRC e QURT

CRC é um algoritmo para calcular *Cycle Redundant Checking*, ou seja, verificações do conteúdo de uma determinada série de dados é usada na verificação de dados em barramentos de comunicação influenciados por ruídos.

Já o 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, usados em ambientes de transição de dados, usa vários *arrays* e operações aritméticas com apenas duas expressões condicionais.

Além disso, são algoritmos que resolvem problemas do mundo real. Esses experimentos seguiram o mesmo método de experimentação do algoritmo de Ordenação em Bolha.[2]. E é mostrado no conjunto de Apêndice de trabalho.

7.7 GRÁFICOS DOS RESULTADOS E COMPARATIVOS

Esta seção apresenta os resultados da abordagem proposta aplicada aos códigos bolha, CRC e QURT. Os algoritmos foram selecionados por tratar-se de algoritmos que resolvem problemas do mundo real.

Com os cenários probabilísticos determinados, seguimos o processo de estimação e caracterização.

a) numero mínimo de simulações: 1.000

b) intervalo de confiança: 0.95

c) erro máximo

permitido: 0.03

7.7.1 Resultados – Código BOLHA – PESTI-1

Os códigos bolha embarcados e executados no processador ARM7 apresentaram um consumo de energia significativo, mas coerente em relação aos estimados na ferramenta PESTI.

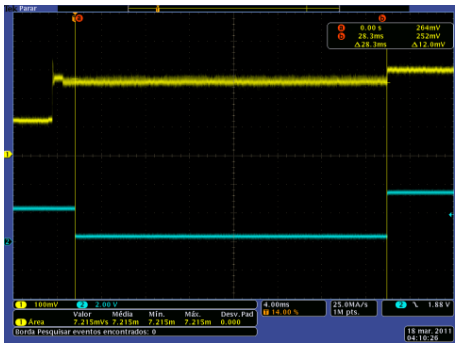


Figura 29 – Dado medido coletado – Bolha – Sinal característico.

Fonte: Autor

Os gráficos a seguir ilustram o comportamento e energético devido a execução do código bolha estimado e medido, mostrado na Figura 30. Nota-se que o consumo do código no processador e do estimado estão na ordem de ηJ. É possível notar, ainda, que ocorre certa fidelidade de nos dados apresentados. Especificamente falando, o consumo no ARM7 foi relacionado com o tempo de execução dos códigos embarcados e porcentagem das faixas de consumo descritas nos gráficos da Figura 30.

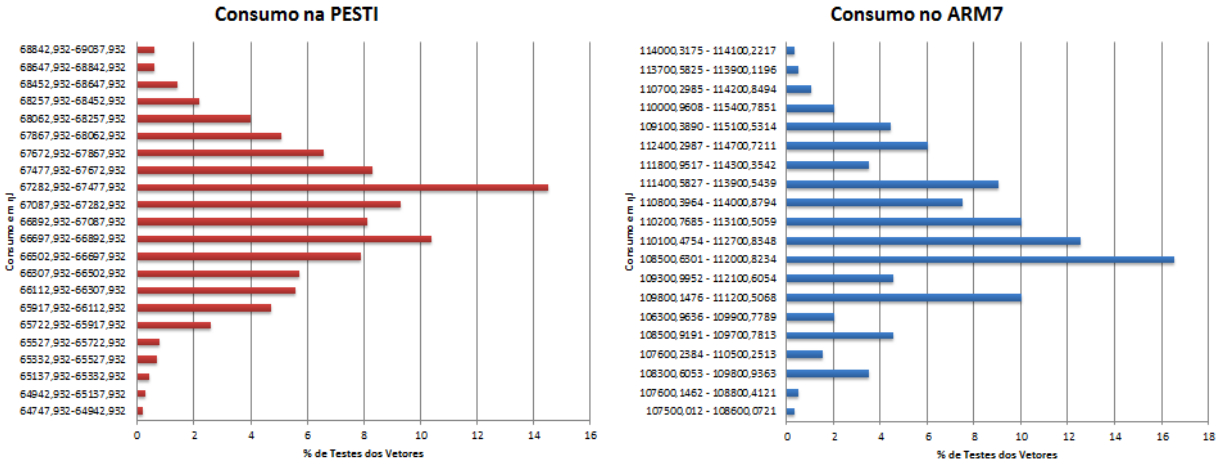


Figura 30 – Gráfico comportamental de consumo energético PESTI e ARM7– Bolha.

Fonte: Autor

Essa diferença entre os valores medidos e estimados é observado na Figura 31 o gráfico comparativo de valores médio de consumo energético entre o ARM7 e a ferramenta PESTI, sendo o valor médio de consumo da PESTI é 66892,932ηJ e o valor médio de consumo do código bolha executado no processador ARM7 de 111140,554 ηJ.

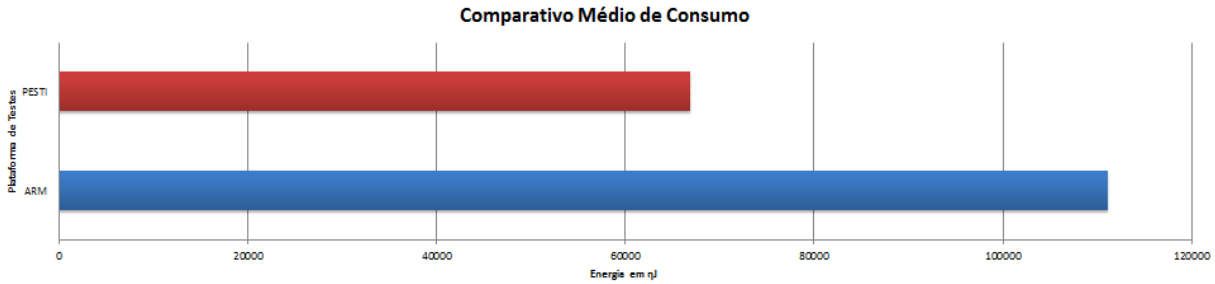


Figura 31 – Gráfico comparativo de consumo energético ARM7 X PESTI – Bolha.

Fonte: Autor

7.7.2 Resultados – CRC PESTI-1

Neste item os códigos embarcados no processador ARM7 são os CRC's, apresentaram um consumo de energia também significativo coerente com os estimados na ferramenta PESTI.

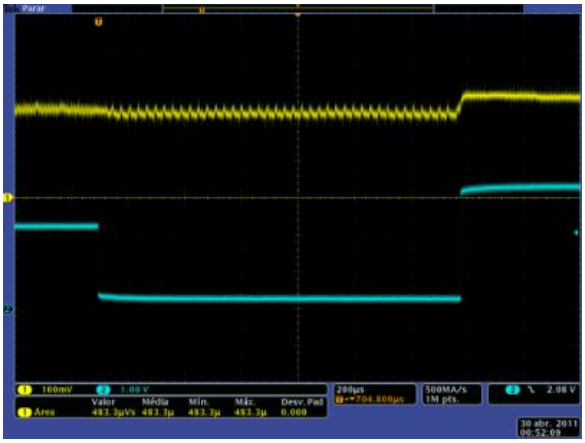


Figura 32 – Dado medido coletado – CRC – Sinal característico.

Fonte: Autor.

Os seguintes gráficos ilustram o comportamento e energético do código CRC estimado e medido Figura 32, sendo a diferença entre os valores medidos e estimados observados na Figura 33 e o gráfico comparativo de valores médio na Figura 34, mostra consumo médio energético entre o ARM7 e a ferramenta PESTI, com o valor médio para o consumo da PESTI

de 44556,334η e o valor médio de consumo do código CRC executado no processador ARM7 de 79211,262ηJ

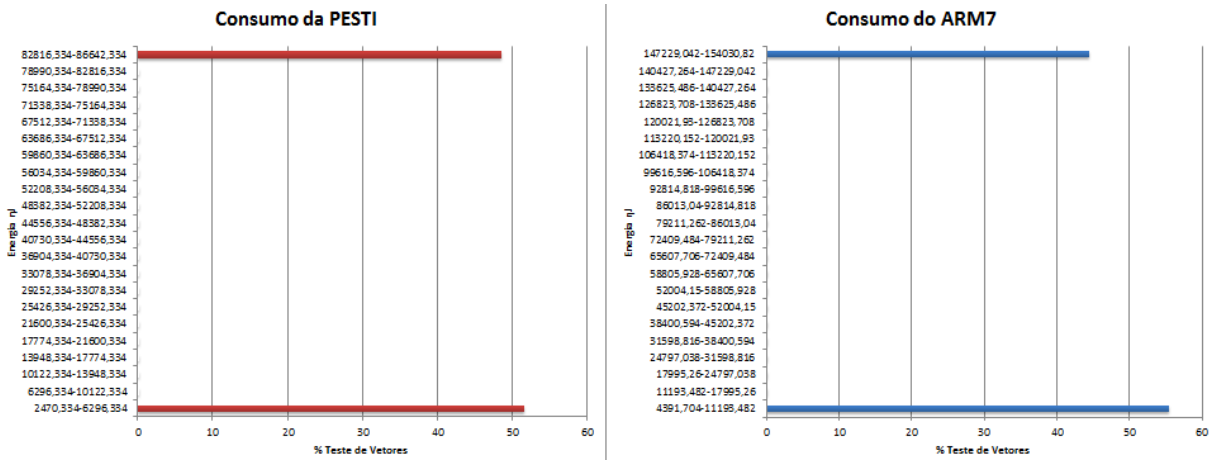


Figura 33 – Gráfico comportamental de consumo energético PESTI e ARM7– CRC.

Fonte: Autor.

E na Figura 34, o gráfico comparativo de valores médio de consumo energético entre o ARM7 e a ferramenta PESTI.

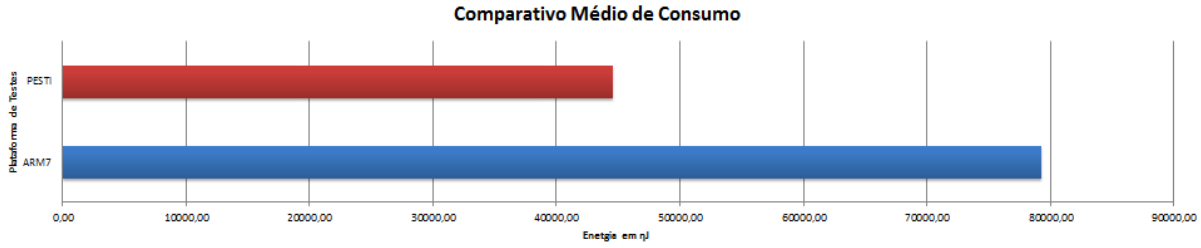


Figura 34 – Gráfico comparativo de consumo energético ARM7 X PESTI – CRC.

Fonte: Autor

7.7.3 Resultados – QURT PESTI-1

Para este tópico, os códigos embarcados no processador ARM7 são os QURT's e apresentaram um consumo de energia também significativo coerente com os estimados na ferramenta PESTI.

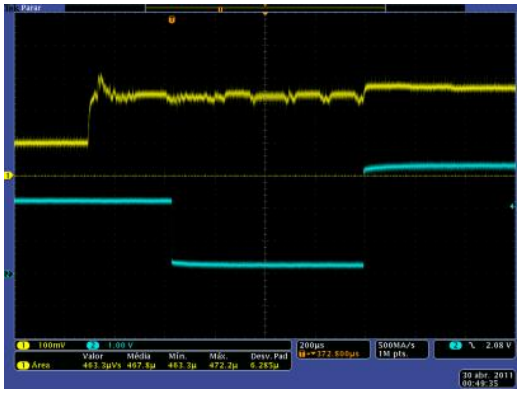


Figura 35 – Dado medido coletado – QURT – Sinal característico.

Os seguintes gráficos ilustram o comportamento e energético do código QURT estimado e medido na Figura 36, apresentando os valores médios de consumo da PESTI com 1426,228 ηJ e para o caracterizador ficaram entre 1716,164 ηJ.

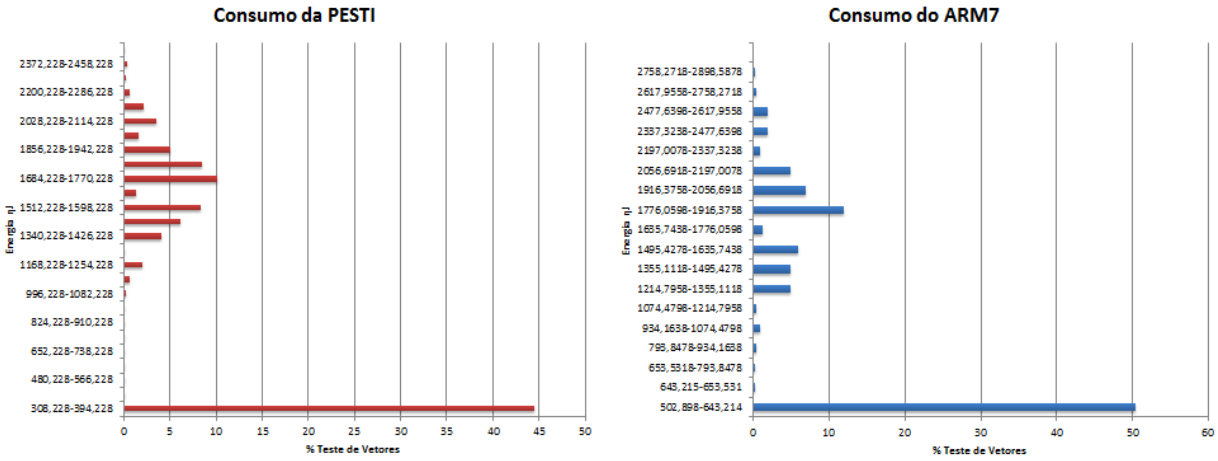


Figura 36 – Gráfico comportamental de consumo energético PESTI e ARM7– QURT.

Fonte: Autor

Finalizando os resultados dos experimentos PESTI X CARACTERIZADOR, a Figura 37 mostra o gráfico que compara os valores médios de consumo energético entre o ARM7 e a ferramenta PESTI para os mesmos códigos do QURT.

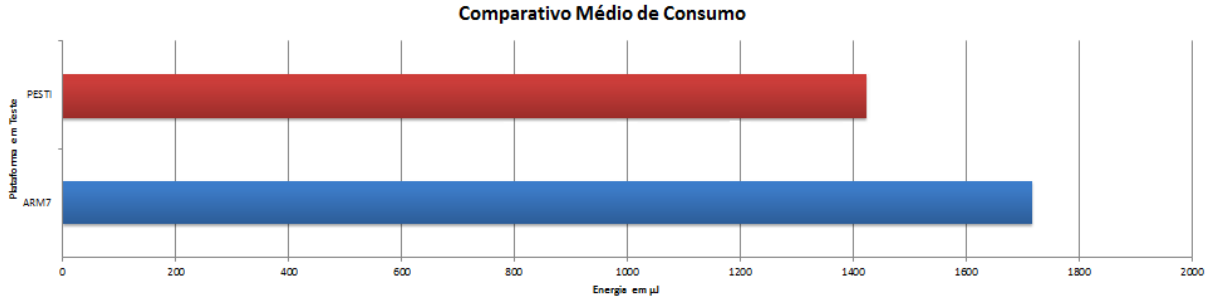


Figura 37 – Gráfico comparativo de consumo energético ARM7 X PESTI – QURT.

Fonte: Autor

7.7.4 Gráficos e Resultado – PESTI - Múltiplos

Nesta seção são apresentados os resultados da abordagem proposta aplicada aos códigos bolha, CRC e QURT, usando a ferramenta PESTI estendida de acordo com a proposta deste trabalho. Para os Múltiplos Cenários todos os códigos foram estimados numa faixa probabilística escolhida aleatoriamente, a faixa escolhida foi de 0.45 a 0.55 com um passo a passo de 0.01, perfazendo uma faixa com 11 cenários e 1000 iterações para a ferramenta PESTI num total de 11000 vetores testados.

Com os cenários probabilísticos determinados, seguimos o processo de estimação e caracterização.

- Número mínimo de simulações: 1.000
- Intervalo de confiança: 0.95
- Erro máximo permitido: 0.03

Para cada cenário a ferramenta gera um conjunto de dados no formato CSV, ou seja, um conjunto de dados de 11 arquivos CSV, preparado para a geração de gráficos e todos esses dados foram plotados e são mostrados nos tópico a seguir.

7.7.5 Resultados – BOLHA - PESTI Múltiplos

O código bolha quando submetido ao processo de estimação em múltiplos cenários se comporta energeticamente conforme o gráfico da Figura 38.

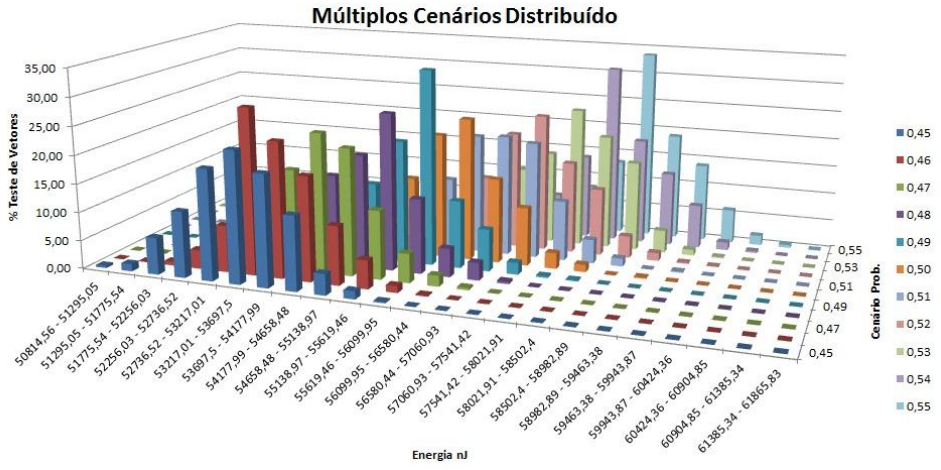


Figura 38 – Gráfico Múltiplo Cenário probabilístico Distribuído – BOLHA.

Fonte: Autor.

Para um melhor entendimento os dados da Figura 41, foram tratados e mostrados na sua forma de média e é mostrado na Figura 39.

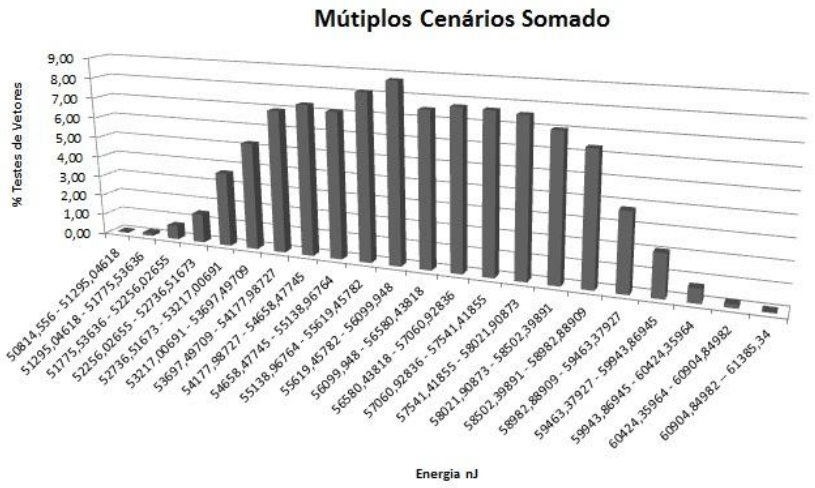


Figura 39 – Gráfico Múltiplo Cenário probabilístico Somado– BOLHA.

Fonte: Autor.

7.7.6 Resultados – CRC - PESTI Múltiplos

O código CRC tem seu comportamento mostrado no gráfico da Figura 40, quando submetido aos múltiplos cenários da ferramenta PESTI.

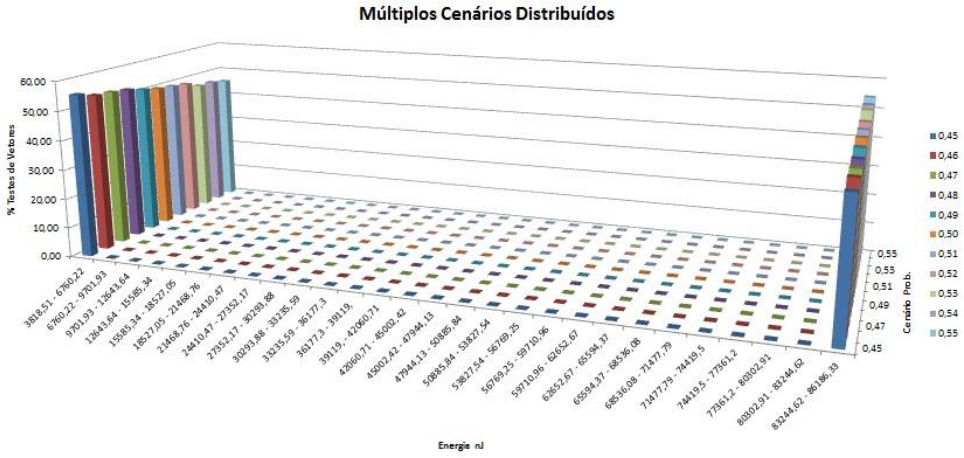


Figura 40 – Gráfico Múltiplo Cenário probabilístico Distribuído – CRC.

Fonte: Autor

É tratado na forma de média na Figura 41 para uma melhor compreensão do seu comportamento.

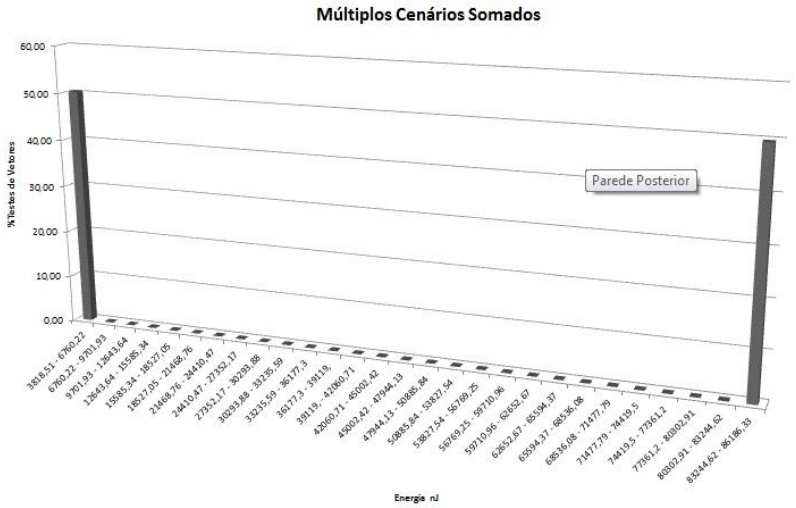


Figura 41 – Gráfico Múltiplo Cenário probabilístico Somado– CRC.

Fonte: Autor.

7.7.7 Resultados – QURT - PESTI Múltiplos

Finalmente, o código QURT sendo submetido aos múltiplos cenários da ferramenta PESTI. Seu comportamento pode ser visualizado na Figura 42.

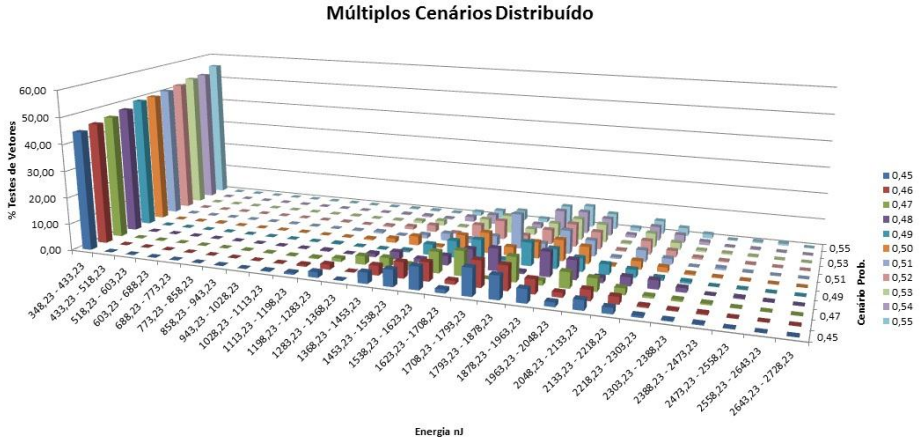


Figura 42 – Gráfico Múltiplo Cenário probabilístico Distribuído – QURT.

Fonte: Autor

E da mesma forma que foi feito para o bolha e CRC. Seus dados foram tratados na forma de média e são mostrados no gráfico da Figura 43 para um melhor entendimento do seu comportamento energético.

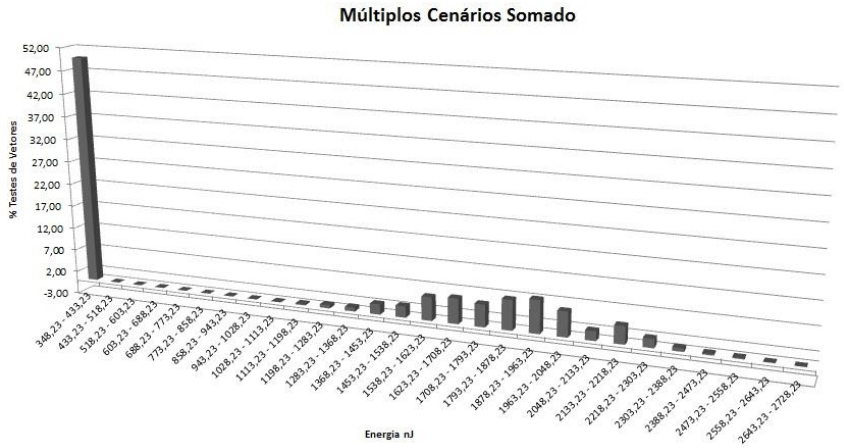


Figura 43 – Gráfico Múltiplo Cenário probabilístico Somado– QURT.

Fonte: Autor

A tabela a seguir mostra que o consumo no Processador ARM sempre é maior do que o consumo estimado pela ferramenta PESTI.

Código ANSI C	Estimador – PESTI – Energia em ηJ	Caracterizador – Energia em ηJ	Estimador – PESTI – Múltiplos Somado- Energia em ηJ	Diferença em % Caracterizador X PESTI
Bolha	66892,932	111140,554	117062,631	39,81%
CRC	44556,334	79211,262	77973,584	43,75%
QURT	1426,228	3565,57	2495,899	38,1%

Tabela 4 – Resultados Comparativos entre Estimado e Executado no ARM.

Portanto é fato que ambas as ferramentas PESTI e o CARACTERIZADOR possuem coerência em seus valores adquiridos em seu funcionamento, porém com fidelidade de valores para menos da PESTI em relação ao CARACTERIZADOR.

7.8 CONSIDERAÇÕES FINAIS PARA ANÁLISE DOS DADOS COLETADOS USANDO INFERÊNCIA ESTATÍSTICA

Para uma maior confiabilidade dos resultados obtidos neste trabalho, utilizou-se uma inferência estatística, em que consideramos espaços amostrais no valor de 1000 amostras para cada resultado obtido durante a realização dos ensaios com os códigos executados tanto na ferramenta PESTI e no Caracterizador.

Seja os códigos bolha, CRC e QURT, com 1000 amostras para cada ensaio, com $(X_1, X_2, \dots, X_{1000})$ o resultado destes mesmo ensaios. Temos que calcular a média amostral de $\bar{X}$ para tipo de código, para em seguida calcular a variância destas amostras e por fim o realizar o cálculo de intervalo de confiança.

Apresentamos a seguir os intervalos de confiança, com probabilidades de 0,95 para a média das medidas obtidas pela ferramenta PESTI e para a média das medidas obtidas no ARM7 através do caracterizador

Para o código Bolha:
Intervalo de Confiança para a Ferramenta PESTI [64468,144; 69317,719].
Intervalo de Confiança para o Caracterizador - ARM7 [100584,034; 121697,075].
Para o código CRC:
Intervalo de Confiança para a Ferramenta PESTI [793,176; 88319,491].
Intervalo de Confiança para o Caracterizador - ARM7 [710,347; 157712,176]
Para o código QURT:
Intervalo de Confiança para a Ferramenta PESTI [162,028; 2690,428]
Intervalo de Confiança para o Caracterizador - ARM7 [121,797; 3310,53]

Observou-se que não há sobreposição dos intervalos, implicando que há diferença significativa entre as médias, com indicação de que a média obtida com o caracterizador no ARM7 é superior a da ferramenta PESTI, com nível de confiança adotado.

7.9 **RESUMO**

Este capítulo apresentou os resultados obtidos na aplicação da abordagem proposta sobre os códigos Bolha, CRC e QURT. Os experimentos foram executados seguindo o mesmo método de experimentação que são divididos em duas fases:

A primeira é definida como a fase estimação/caracterização e definido por 6 passos:

- Estimação e caracterização de código;
- Compilação do código;
- Upload do código para o processador ARM7;
- Trigger do osciloscópio para medição;
- Medição e armazenamento dos valores medidos.
- Comparação dos resultados alcançados entre a Ferramenta PESTI e Hardware de caracterização.

A segunda é definida como a fase de estimação de múltiplos cenários e definida por 3 passos:

- Caracterização do Cenário Probabilístico;
- Simulação na Ferramenta PESTI;
- Análise dos resultados alcançados pela Ferramenta PESTI

E por fim a utilização da inferência estatística para uma maior confiabilidade dos resultados dos ensaios realizados.

Percebe-se que a caracterização do processador ARM7 em relação à ferramenta PESTI conseguiu alcançar os objetivos proposto, porém com uma diferença superior em relação à estimação.

Também nota-se a fidelidade entre os dados coletados pela Ferramenta PESTI e o hardware de medição.

Por essa hipótese, é possível fazer comentário referente a esta diferença entre a PESTI e HARDWARE: provavelmente a PEESTI não considera o funcionamento de blocos internos do ARM, necessários para o processamento dos códigos, isto fica coerente que na plataforma real onde consumo é bem maior. Já na Ferramenta PESTI é feita uma a relação de %Freq dos vetores testados e esta relação está diretamente relacionada com %Freq dos vetores testados e o Tempo de execução do código embarcado e executado no ARM7, mostrado em isso conforme os gráficos apresentados nesta seção.

CAPÍTULO VIII

CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou uma abordagem para fase de planejamento e desenvolvimento de sistemas embarcados, possibilitando a análise de aspectos de consumo de energia.

Descrever uma forma de estimar e caracterizar o consumo de energia e tempo de execução de sistemas embarcados desenvolvidos em linguagem ANSI C. A abordagem conta com o suporte da ferramenta PESTI modificada para atender simples cenários ou múltiplos cenários e validada aplicando através de estudos de casos diferentes de algoritmos:

- De ordenação em bolha (*bubblesort*);
- De redundância cíclica (*CRC*);
- De cálculo de raiz quadrada (*QURT*).

Da mesma forma, os códigos usados foram embarcados em um processador ARM7 executados e medidos de forma coerente em um ambiente real.

O primeiro estudo de caso trata-se de um código um tanto conhecido e que possui um elemento de natureza probabilística que define cenários probabilísticos usando uma matriz de elementos classificados em ordem direta e os dois últimos estudos de caso foram extraídos do *benchmark Powerstone* [32]. O *Powerstone* é um benchmark especificamente desenvolvido para avalição de consumo de energia por meio da exploração de diferentes aspectos do sistema como já foi dito anteriormente.

Os resultados adquiridos foram comparados aos resultados produzidos pelo caracterizador. Em ambas as simulações a plataforma alvo utilizada foi de um processador ARM7, na ferramenta PESTI. Isso foi caracterizado pela utilização do modelo de consumo de energia baseado em instruções.

No caracterizador, através da medição do consumo de corrente do processador e a comparação dos resultados, mostra-se que existe uma diferenças significativas entre estimativas produzidas pela ferramenta PESTI e pelo caracterizador. Por hipótese o consumo energético dos blocos internos do processador ARM7, que trata aritmeticamente os valores de variáveis nos códigos usado neste trabalho, não foi considerado, isto ocorreu para os três códigos bolha. CRC e QURT usado nos experimentos.

Acreditamos, por meio das evidências produzidas pelos estudos de casos, que nossa abordagem pode ser empregada no projeto de aplicações de pesquisas e desenvolvimento de sistemas embarcados que utilizam códigos escritos em ANSI C em produtos portáteis comerciais, desde que existam ferramentas as quais suportem o modelo proposto, automatizem o processo de tomada de estimativas e forneçam resultados confiáveis em tempo de projeto. Isso representa uma importante tarefa no projeto de sistemas embarcados, pois nas fases iniciais do desenvolvimento do sistema é que o projetista deve tomar a decisão sobre quais algoritmos e plataformas de hardwares (processador) deverão ser utilizados. Nossa ferramenta permite que o desenvolvedor estime diversos cenários de execução para o mesmo algoritmo sobre um determinado processador, não só ARM, mas outros processadores. Portanto, o desenvolvedor é capaz de comparar os resultados obtidos, não só estimativas de consumo de energia, mas também tempo de execução, para vários tipos de algoritmos e outros processadores desejados.

8.1 **CONTRIBUIÇÕES**

A proposta desta dissertação de mestrado teve como objetivo definir uma abordagem de medição e comparação para estimar consumo de energia, caso necessário, o Tempo de Execução de Códigos para sistemas embarcados desenvolvidos em linguagem ANSI C.

A abordagem definida e apresentada neste trabalho teve como base as ideias de proposta em [3] e [2]. As principais diferenças entre ambos os trabalhos são os modelos utilizados para representar o fluxo de controle de um programa, a linguagem de programação utilizada para codificar os programas, além de que ambos utilizaram plataformas distintas (8051) em [3] e (ARM7, mas não usada, e é substituída pelo simulador *Sim-PAnalyzer*) em [2].

Neste trabalho as principais contribuições foram:

- A definição formal é um modelo de caracterização real para códigos embarcados em um processador ARM7 sugeridos muitos autores de trabalhos relacionados utilizados nesta dissertação;
- A definição de um novo padrão de anotação de código para representação de múltiplos cenários probabilísticos;
- A extensão da ferramenta PESTI para estimar o consumo de energia em sistemas embarcados para múltiplos cenários.

Avaliando sistemas que possuam códigos escritos em linguagem ANSI C, percebe-se que a abordagem proposta tenha a possibilidade de ser empregada em grande parte de aplicações de sistemas embarcados, pois essa linguagem de programação é bastante difundida

no desenvolvimento de aplicações de linha de pesquisa de sistemas embarcados e de produtos portáteis. O padrão de anotação proposto neste trabalho permite ao desenvolvedor e projetista definirem desde simples cenários até múltiplos cenários probabilísticos de execução de seus sistemas, além de permitir o desenvolvimento de outras ferramentas automáticas de reconhecimento de tal notação.

8.2 **PROBLEMAS**

Cita-se alguns obstáculos que entendemos que fazem parte do processo de pesquisa. A falta de ambiente mais controlado para a realização dos ensaios e testes sem tanta influência do ruído de 60Hz, para a medição do consumo de corrente do software em execução no processador ARM7. O ruído de 60Hz foi o mais complexo de se trabalhar, isso porque está presente na rede AC de alimentação dos equipamentos usados nos testes, osciloscópio, PC e fonte de alimentação dos Kit's de desenvolvimento ARM e de medição.

A solução encontrada foi, primeiramente, isolar estaticamente a mesa dos equipamentos e o piso. Para tanto foram utilizadas mantas antiestáticas, que foram usadas sobre a mesa e sobre o piso onde eram realizados os testes. Para realizar os ensaios, utilizou-se pulseira antiestática e luva antiestática, isso permite que qualquer acúmulo de cargas no corpo do operador do equipamento seja desfeito com o retorno ao equilíbrio.

Outro passo foi diminuir o ruído AC gerado pela rede de alimentação AC. Foram substituídas as fontes de alimentação DC, utilizadas pela placa de medição e pelo Kit de desenvolvimento ARM, usando uma fonte química de geração de tensão DC, no caso uma batéria de carro. Com isso, foi possível deixar o sinal medido um tanto "livre" do ruído de 60Hz.

8.3 **TRABALHOS FUTUROS**

Propomos, para a continuação deste trabalho de pesquisa, alguns tópicos que devem ser analisadas com o propósito de iniciar novos trabalhos de graduação e dissertações de mestrado.

- Automatizar o sistema de medição para a caracterização de códigos ANSI C para sistemas embarcados em ARM7;
- Caracterizar comandos exclusivos do ARM7 ou plataformas superiores.

- Desenvolvimento de um mecanismo de tradução de código ANSI C para grafo de fluxo de controle que reconheça qualquer sistema desenvolvido em linguagem ANSI C;
- Desenvolver uma ferramenta própria para análise dos resultados que permita a visualização dos mesmos em forma de gráficos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] [Barreto 2005] Barreto, R. (2005). **A Time Petri Net-Based Methodology for Embedded Hard Real-Time Software Synthesis**. PhD thesis, Cin/UFPE.

[2] [Araújo Mar 2009] Araújo Mar, R. (2009). **Uma Abordagem Probabilística baseada em Grafo de Fluxo de Controle para Estimar o Consumo de Energia**. M.Sc. thesis, UFAM.

[3] [Oliveira Junior 2006] Oliveira Junior, M. N. (2006). **Estimativa do Consumo de Energia Devido ao Software: Uma Abordagem Baseada em Redes de Petri Coloridas**. PhD thesis, Cin/UFPE.

[4] [Callou 2006] Oliveira Junior, M. N. (2006). **Um Método Formal Baseado em Redes de Petri Coloridas para Estimar o Tempo de Execução e o Consumo de Energia em Sistemas Embarcados**. PhD thesis, Cin/UFPE.

[5] [Pires Ribeiro 2007] Pires Ribeiro, A. (2009). **Estimativa do Consumo de Energia de Código ANSI-C para sistemas embarcados: Uma abordagem baseada em simulação estocástica**. M.Sc. thesis, UFPE.

[6] [Bona et al. 2002] Bona, A., Sami, M., Sciuto, D., Zaccaria, V., Silvano, C., and Energy, R. Z. (2002). **Estimation and optimization of embedded vliw processors based on instruction clustering**. In In Proceedings of the 39th conference on Design automation, pages 886 - 891. ACM Press.

[7] [Brooks et al. 2000] Brooks, D., Tiwari, V., and Martonosi, M. (2000). Wattch: **a framework for architectural-level power analysis and optimizations**. In Proceedings of 27th annual international symposium on Computer architecture, pages 83 - 94. ACM Press.

[8] [Burch et al. 1993] Burch, R., Najm, F., Yang, P., and Trick, T. (1993). **A monte carlo approach for power estimation**. In IEEE Transactions on VLSI Systems.

[9] [Costa et al. 1999] Costa, J., Monteiro, J., Silveira, L., and Devadas, S. (1999). **A probabilistic approach for rt-level power modeling**. In 6th IEEE International Conference on Electronics, Circuits and Systems.

[10] [Klass et al. 1998] Klass, B., D.Thomas, Schmit, H., and Nagle, D. (1998). **Modeling inter-instruction energy effects in a digital signal processor**. in power-driven microarchitecture workshop. In conjunstion with ISCA98.

[11] [Laurent et al. 2004] Laurent, J., Julien, N., Senn, E., and Martin, E. (2004). **Functional level power analysis: An efficient approach for modeling the power consumption of complex processors**. In IEEE DATE.

[12] [Lee et al. 1997] Lee, M., Tiwari, V., Malik, S., and Fujita, M. (1997). **Power analysis and minimization for embedded DSP software**. In IEEE Trans. VLSI Systems, pages 123 - 135.

[13] [Nikolaidis et al. 2002] Nikolaidis, S., Kavvadias, N., and Neofotistos, P. (2002). **Instruction-level power measurement methodology**. Technical report, Electronics Lab., Physics Dept., Aristotle University of Thessalonik.

[14] [Ravi et al.] Ravi, S., Raghunatha, A., and Chakradhar, S. **Eficient RTL power estimation for large designs**. Computer and Commmunication Research Labs, NEC USA, Princeton, NJ 08540.

[15] [Leon-Garcia et al. 2008] LEON-GARCIA, A. **Probability and Random Processes for Electrical Engineering**. 3rd Edition. Prentice Hall.

[16] [Allen, A. O. et al. 1990] ALLEN, A. O. **Probability, Statistics, and Queueing Theory with Computer Science Applications**. Second Edition. Academic Press.

[17] [Ball, Stuart et al. 2005] Ball, Stuart. – "**Embedded Microprocessor Systems: Real World Design**", 3rd edition, Editora: MCPros, EUA.

[18] – [Reis, Claiton et al. 2004] Reis, Claiton – "**Sistemas Operacionais para Sistemas Embarcados**", Tutorial, Editora: EDUFBA, BRASIL.

[19] – [Cunha, Alessandro et al. 2007] Cunha, Alessandro – "**Sistemas Embarcados**", Revista Saber Eletrônica, 414, Editora: Saber, BRASIL.

[20] – Prof. Rajesh K. Gupta – "**Introduction to Embedded Systems**", Website, Acessado em: 10/12/2010. UCLA, EUA, 2002. *http://www.ics.uci.edu/~rgupta/ics212/w2002/intro.pdf*

[21] – SOUSA, Daniel Rodrigues. **Microcontroladores ARM7**. São Paulo: Érica. 2006

[22] – RENALDI, Felipe. Genos - **Protótipo De Um Montador De Sistemas Operacionais para Sistemas Embarcados**. Santa Catarina: Monografia FURB, 2006. Disponível em: *http://www.bc.furb.br/docs/MO/2006/307628_1_*

[23] – DAVID, J. Lilja, ***Measuring Computer Performance***: A Practitioner's Guide. Cambridge University Press, 2000.

[23] – OSCILOCÓPIO TEKTRONICS, **Manual de Operação DPO 4034**.

[24] V. Tiwari, S. Malik, and A. Wolf. ***Power analysis of embedded software: a first step towards software power minimization***. In IEEE Trans. VLSI Systems, volume 2, pages 437-445. December 1994.

[25] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. **The design and use of simplepower**: *A cycle-accurate energy estimation tool. In Proceedings of 37th conference on Design automation*, pages 340-345. ACM Press, 2000.

[26] E. Senn, J. Laurent, N. Julien, and E. Martin. Softexplorer: **estimation, characterization and optimization of the power and energy consumption at the algorithmic level**. In IEEE PATMOS. September 2004.

[27] V. Tiwari and M. Lee. **Power analysis of a 32-bit embedded microcontroller**. In VLSI Design Journal, page 7(3). 1998.

[28] V. Tiwari, S. Malik, A. Wolf, and M. Lee. **Instruction level power analysis and optimization of software**. In J. VLSI Signal Process. Syst., pages 13(2-3):223-238. 1996.

[29] Vivek Tiwari. **Logic and System Design for Low Power Consumption**. PhD thesis, Princeton University, November 1996.

[30] J. Gross. **Graph Theory and its applications**. CRC, Boca Raton, 1999.

[31] Sim-Panalyzer, **The SimpleScalar-Arm Power Modeling Project**. Disponível em http://www.eecs.umich.edu/ panalyzer/., 2008.

[32] Afzal Malik, Bill Moyer, and Dan Cermak. **A low power unified cache architecture providing power and performance flexibility (poster session)**. In ISLPED '00: Proceedings of the 2000 international symposium on Low power electronics and design, pages 241-243, New York, NY, USA, 2000. ACM.

[33] **Synopsys products and solutions**. Disponível em http://www.synopsys.com/products/solutions/galaxy/power/power.html, 2008.

[34] **Introduction to Embedded Systems**. Disponível em http://www.ics.uci.edu/ rgupta/ics212/w2002/intro.pdf, 2009.

[35] **The Lex and Yacc Page**. Disponível em http://dinosaur.compilertools.net/, 2009.

[36] **Python Programming Language** - Oficial Website. Disponível em http://www.python.org/, 2009.

[37] **Sparse - a Semantic Parser for C**. Disponível em http://www.kernel.org/pub/software/devel/sparse/, 2009.

[38] **Manta Antiestática**. Disponível em http://www.clangsm.com.br/vb/esquemas-de-hardware-repara%E7%F5es-jumpers-em-geral/7222-manta-dissipativa.html.

[39] **Pulseira Antiestática**. Disponível em http://pt.wikipedia.org/wiki/Pulseira_antiest%C3%A1tica.

[40] Carlos Mar, Raimundo Barreto, Meuse Oliveira Jr e Edward Moreno Ordonez. **PESTI: A Probabilistic Tool for Energy Consumption and Execution-Time Estimation for ANSI C Embedded Systems**. IEEE International Conference on Industrial Technology (ICIT 2009). 10 - 13 February, 2009. Churchill, Victoria, Australia.

[40] **Batéria de Lítio** Disponível em http://eletronicos.hsw.uol.com.br/baterias-ion-litium.htm.

APÊNDICE

LISTAGEM DO CÓDIGO BOLHA ADAPTADO PARA O PROCESSADOR ARM7

```
/*****************************************************************************/
/*              Programa adaptado para ser embarcado no processador ARM      */
/*****************************************************************************/
/*                          Mestrado em Engenharia Elétrica                  */
/*****************************************************************************/
/*                  HELLO_BOLHA.C:  Hello World Example                      */
/*****************************************************************************/
#include <stdio.h>              /* declarations for I/O functions */
#include <LPC214x.H>            /* LPC214x definitions */

#define SET_LED_ON FIO0CLR = 0x80000000
#define SET_LED_OFF FIO0SET =0x80000000
int i, j, temp;
int numbers[] =
{368,589,750,199,274,674,430,340,214,516,706,939,50,696,1000,155,145,750,501,65,953,301
,814,159,530,593,305,241,893,420,698,685,253,726,754,987,60,833,622,300,315,396,601,543,
829,865,943,828,113,746} ;
/* main program */
int main (void)  {              /* execution starts here                */
/************************    Setup do pino P0.31      ************************/
        SCS=3;
        FIO0DIR =0x80000000;
        FIO0MASK=0x7FFFFFFF;
        FIO1DIR =0x00000000;
        FIO1MASK=0xFFFFFFFF;
 /*****************************************************************************/
        SET_LED_ON;
        /* LP01 @LoopMax(50, 1)  */
        for (i=49; i >= 0; i--)
```

```
        {
          /* LP02 @LoopMax(24, 1)  */
          for (j=1; j <= i; j++)
          {
           /* @BranchProb(0.5089, 1)  */
           if ( numbers[i] > numbers[j] )
           {
             temp = numbers[j-1];
             numbers[j-1] = numbers[j];
             numbers[j] = temp;
           }
          }
        }
                SET_LED_OFF;
        for(;;)
        {
        }
        return 0;
}
```

LISTAGEM DO CÓDIGO CRC ADAPTADO PARA O PROCESSADOR ARM7

```
/*****************************************************************************/
/*              Programa adaptado para ser embarcado no processador ARM      */
/*****************************************************************************/
/*                          Mestrado em Engenharia Elétrica                  */
/*****************************************************************************/
/*                                  HELLO_CRC.C                              */
/*****************************************************************************/
#include <stdio.h>              /* declarations for I/O functions */
#include <LPC214x.H>            /* LPC214x definitions */
#define SET_LED_ON FIO0CLR = 0x80000000
#define SET_LED_OFF FIO0SET =0x80000000

unsigned char aa[] = "asdffeagewaHAFEFaeDsFEawFdsFaefaeerdjgpim23";
```

```
unsigned short i1, i2;
unsigned short icrc(unsigned short crc, unsigned char *lin, unsigned int len, short jinit, int
jrev);
unsigned short icrc1(unsigned short crc, unsigned char onech);
/* main program */
int main (void)  {              /*          execution starts here                */

/************************    Setup do pino P0.31      ******************************/
        SCS=3;
        FIO0DIR =0x80000000;
        FIO0MASK=0x7FFFFFFF;
        FIO1DIR =0x00000000;
        FIO1MASK=0xFFFFFFFF;
 /*****************************************************************************/
        SET_LED_ON;
                i1 = icrc(0, aa, 40, (short) 0, 1);
         /* @BranchProb(1.0, 1) */
         if (i2==387)
         {
                i1=0;
         }
         SET_LED_OFF;
                for(;;)
                {
                }
        return 0;
}
unsigned short icrc1(unsigned short crc, unsigned char onech)
{
  int i, flag;
  unsigned short ans = (crc ^ onech << 8);
  /* LP01 @LoopMax(8, 1) */
  for (i = 0; i < 8; i++)
  {
```

```
            flag = ans & 0x8000;
            /* @BranchProb(0.50, 1) */
            if (flag != 0)
            {
                ans = (ans <<= 1) ^ 4129;
            }
        else
            {
                ans <<= 1;
            }
  }
  return ans;
}
unsigned short icrc(unsigned short crc, unsigned char *lin, unsigned int len, short jinit, int
jrev)
{
  static unsigned short icrctb[256], init = 0;
  static unsigned char rchr[256];
  unsigned short tmp1, tmp2, j, cword = crc;
  static unsigned char it[16] = {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15};

  /* @BranchProb(0.4896, 1) */
  if (init!=0)
  {
     init = 1;
     /* LP02 @LoopMax(256, 1) */
     for (j=0; j <= 255; j++)
     {
          icrctb[j] = icrc1(j << 8, (unsigned char) 0);
          rchr[j] = (unsigned char) (it[j & 0xF] << 4 | it[j >> 4]);
     }
  }
  /* @BranchProb(0.5186, 1) */
  if (jinit >= 0)
```

```
{
  cword = ((unsigned char) jinit) | (((unsigned char) jinit) << 8);
}
else
{
        /* @BranchProb(0.2364, 1) */
        if (jrev < 0)
        {
        cword = rchr[((unsigned char) ((cword) >> 8))] | rchr[((unsigned char) ((cword) &
0xFF))] << 8;
        }
}
/* LP03 @LoopMax(40, 1) */
for (j = 1; j <= len; j++)
{
   /* @BranchProb(0.4804, 1) */
   if (jrev < 0)
   {
        tmp1 = rchr[lin[j]] ^ ((unsigned char) ((cword) >> 8));
   }
   else
   {
        tmp1 = lin[j] ^ ((unsigned char) ((cword) >> 8));
   }
   cword = icrctb[tmp1] ^ ((unsigned char) ((cword) & 0xFF)) << 8;
}
/* @BranchProb(0.5218, 1) */
if (jrev >= 0)
{
   tmp2 = cword;
}
else
{
```

```
    tmp2 = rchr[((unsigned char) ((cword) >> 8))] | rchr[((unsigned char) ((cword) & 0xFF))]
<< 8;
  }
   return (tmp2);
}
```

LISTAGEM DO CÓDIGO QURT ADAPTADO PARA O PROCESSADOR ARM7

```
/*****************************************************************************/
/*              Programa adaptado para ser embarcado no processador ARM      */
/*****************************************************************************/
/*                          Mestrado em Engenharia Elétrica                  */
/*****************************************************************************/
/*                                HELLO_QURT.C                               */
/*****************************************************************************/
#include <stdio.h>              /* declarations for I/O functions */
#include <LPC214x.H>            /* LPC214x definitions */
#define SET_LED_ON FIO0CLR = 0x80000000;
#define SET_LED_OFF FIO0SET =0x80000000;

double a[3], x1[2], x2[2];
int result, r;
int qurt(double a[], double x1[], double x2[]);
/* main program */
int main (void)  {              /* execution starts here                */
 /************************ Setup do pino P0.31      ********************************/
        SCS=3;
        FIO0DIR =0x80000000;
        FIO0MASK=0x7FFFFFFF;
        FIO1DIR =0x00000000;
        FIO1MASK=0xFFFFFFFF;
 /*****************************************************************************/
        SET_LED_ON;
         a[0] = 1.75;
```

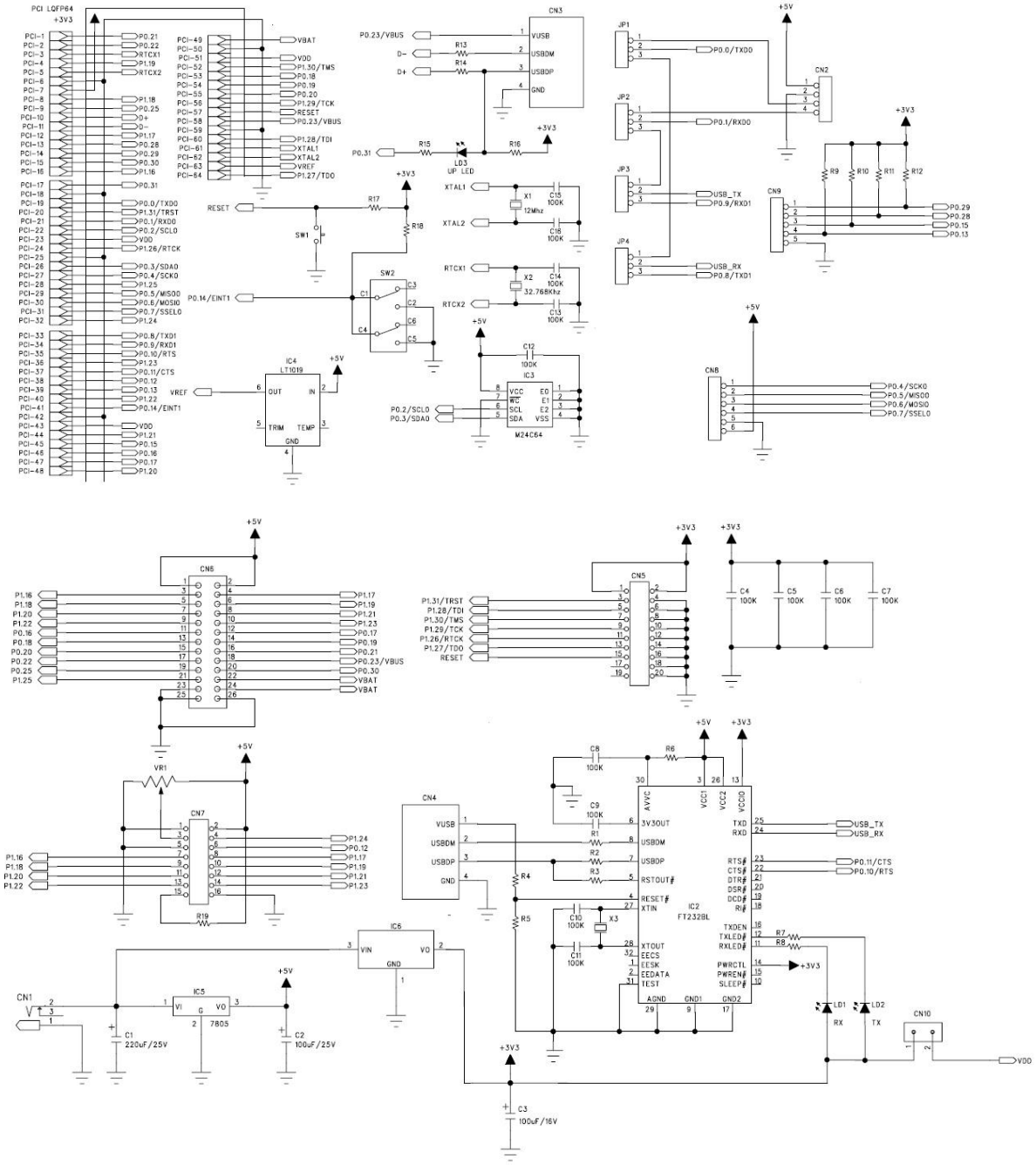
```
        a[1] = -3.2;
        a[2] = 2.45;
        r = qurt(a, x1, x2);
        result = *(int *) &x1[0];
      SET_LED_OFF;
      for(;;)
      {
      }
      return 0;
}
double fabs(double x)
{
      double y;
 /* @BranchProb(0.0, 1)*/
 if (x < 0)
 {
      y = -x;
 }
 else
 {
      y = x;
 }
 return  (y);
}
double sqrt(double val)
{
 double x = val/10;
 double dx;
 double diff;
 double min_tol = 0.00001;
 double fdiff;
 int i, flag;
 flag = 0;
 /* @BranchProb(0.0, 1) */
```

```
 if ( val == 0 )
 {
   x = 0;
 }
 else
{
   /* LP01 @LoopMax(20, 1) */
   for (i=1; i<20; i++)
    {
         /* @BranchProb(0.35, 1) */
     if ( flag != 0)
     {
         dx = (val - (x*x)) / (2.0 * x);
         x = x + dx;
         diff = val - (x*x);
         fdiff = fabs(diff);
         /* @BranchProb(0.05, 1) */
         if ( fdiff <= min_tol )
         {
                 flag = 1;
         }
     }
     else
         {
          x =x;
         }
    }
 }
  return x;
}
int qurt(double a[], double x1[], double x2[])
{
  double d, w1, w2, res;
  int ret=0;
```

```
/* @BranchProb(0.0, 1) */
if (a[0] == 0.0)
{
  ret = (999);
}
d = a[1] * a[1] - 4 * a[0] * a[2];
w1 = 2.0 * a[0];
res = fabs(d);
w2 = sqrt(res);
/* @BranchProb(1.0, 1) */
if (d > 0.0)
  {
    x1[0] = (-a[1] + w2) / w1;
    x1[1] = 0.0;
    x2[0] = (-a[1] - w2) / w1;
    x2[1] = 0.0;
  }
else
{
    /* @BranchProb(0.0, 1) */
    if (d == 0.0)
  {
    x1[0] = -a[1] / w1;
    x1[1] = 0.0;
    x2[0] = x1[0];
    x2[1] = 0.0;
  }
else
  {
    w2 /= w1;
    x1[0] = -a[1] / w1;
    x1[1] = w2;
    x2[0] = x1[0];
    x2[1] = -w2;
```

```
    }
  }
    return (ret);
}
```

ESQUEMA ELÉTRICO DO KIT DESENVOLVIMENTO ARM



Ficha catalográfica elaborada pela Biblioteca Central da UFAM

S586e Silva, Francisco Coelho
Estimador e caracterizador de consumo de energia para software embarcado. / Francisco Coelho da Silva. - Manaus, AM : UFAM, 2011.
96 f.: il. color. ; 30 cm

Inclui referências.
Dissertação (Mestre em Engenharia Elétrica). Universidade Federal do Amazonas. Orientador: Prof. Dr. João Edgar Chaves Filho.

1. Engenharia de software 2. Programação (computadores) 3. Sistemas embarcados (computadores) I. Chaves Filho, João Edgar (Orient.) II. Título

CDU (2007): 004.41(043.3)