UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

CENTRO DE INFORMÁTICA

2010.2

Framework para automação e análise de potência em sistemas embarcados baseados no processador ARM7

Trabalho de Graduação

Aluno : Diogo de Lima Lages (dll@cin.ufpe.br)

Orientador : Abel Guilhermino da Silva Filho (agsf@cin.ufpe.br)

14 de Dezembro de 2010

Orientador :

Abel Guilhermino da Silva Filho

Aluno :

Diogo de Lima Lages

Agradecimentos

Agradeço a Mario Jorge Gracindo Lages, Claudia Marcia de Lima Lages, Herbert Loureiro e a Thiago de Lima Lages, membros da minha família por terem me ajudado e terem sido compreensíveis ao longo desses cinco anos, sempre estando ao meu lado quando eu precisasse.

Agradeço a Thiago Figueredo a Abner Barro e a Daniel Filgueiras por terem me ajudado nesse projeto de graduação fornecendo explicações sobre equipamentos utilizados e estando sempre dispostos a ajudar.

Ao professor Abel Guilhermino por ter confiado esse tema desafiante a minha pessoa e mostrado que é necessário ter calma para alcançar os objetivos.

Enfim, agradeço a todos meus familiares, amigos, colegas e professores que me ajudaram na formação e fortalecimento do meu caráter como pessoa e cidadão.

RESUMO

A necessidade de diminuir consumo de energia elétrica em sistemas embarcados é uma restrição que nasceu com o próprio conceito de sistema embutido. Diversas maneiras para obter o consumo de energia é possível, porém a que fornece mais detalhes é uma análise da netlist, o qual possui informações estruturais sobre o design de determinado IP.

Apesar do modelo citado ser o mais próximo a realidade o mesmo não está acessível para projetos industriais. Em virtude da falta de acessibilidade a detalhes sobre o circuito interno outras técnicas foram elaboradas de forma a ser possível caracterizar o consumo de energia em sistemas embarcados.

Em virtude da característica restritivas de sistemas embarcados e no aumento na demanda por tais sistemas a caracterização de variáveis se torna uma rica fonte de pesquisa a ser explorada.

Nesse contexto, o objetivo desse trabalho é fornecer um framework capaz de caracterizar tempo requisitado, potência necessária e energia elétrica consumida por determinado programa operando em ambiente real, sendo possível também caracterizar a potência utilizada e a quantidade de instruções executadas através de simulações.

Palavras-Chaves : Sistemas embarcados, máquinas virtuais, potência.

ABSTRACT

The need to reduce energy consumption in embedded systems is a constraint that was founded with the concept of embedded system. Several ways to take the power consumption is possible, but that provides more detailed analysis is a netlist, which contains information about the structural design of a particular IP.

Despite the above model is the closest to reality it is not accessible to industrial projects. Because of the lack of accessibility to information about the internal circuitry other techniques were draws up in order to be able to characterize the energy consumption in embedded systems.

Because of the restrictive features of embedded systems and increasing demand for such systems the characterization of variables like time, power and energy becomes a rich source of research to be explored.

In this context, our objective is to provide a framework able to characterize the time required, power required and energy consumed by a particular program running in real environment; it is possible to characterize the power used and the number of instructions too executing a program in virtual environment.

Keywords : Embedded systems, Virtual machines, power.

SUMÁRIO

1 - INTRODUÇÃO 11

2 - O PROBLEMA 13

2.1 DESCRIÇÃO DO PROBLEMA 13

2.2 JUSTIFICATIVA 14

3 - ESTADO DA ARTE 16

3.1 POWER ANALYSIS OF EMBEDDED SOFTWARE: A FIRST STEP TOWARDS SOFTWARE POWER MINIMIZATION 16

3.2 SOFTWARE POWER ESTIMATION AND OPTIMIZATION FOR HIGH PERFORMANCE, 32-BIT EMBEDDED PROCESSORS 18

3.3 MEASUREMENT OF CURRENT VARIATIONS FOR THE ESTIMATION OF SOFTWARE-RELATED POWER CONSUMPTION 19

4 - FERRAMENTAS 20

4.1 SKYEYE 20

4.1.1 Característica 20

4.1.2 Instalação 21

4.1.3 Configuração 21

4.2 OSCILOSCÓPIO DSO3000 24

4.3 ARM7TDMI-S 28

5 - PROPOSTA 32

5.1 OBJETIVOS 32

5.2 FLUXO 34

5.2.1 Fluxo Principal 34

5.2.1.1 Criar arquivos 35

5.2.1.2 Execução Ambiente Virtual 35

5.2.1.3 Execução Ambiente Real 36

5.2.2 Fluxo Criar arquivo 36

5.2.3 36

5.2.3.1 Escrever Arquivos Para Ambiente 37

5.2.3.2 *Escrever scripts para compilação 37*

5.2.3.3 Escrever scripts para gdb 38

5.2.4 Fluxo Ambiente Virtual 40

5.2.4.1 Compilação 40

5.2.4.2 Execução Skyeye 41

5.2.4.3 Depuração 41

5.2.4.4 Processamento 42

5.2.4.5 Formatação 43

5.2.5 Fluxo Ambiente Real 45

5.2.5.1 Compilação 45

5.2.5.2 Setup Uart ... 46
5.2.5.3 Upload Firmware ... 46
5.2.5.4 Setup & Execute Threads ... 47
5.2.5.4.1 Thread Read ... 47
5.2.5.4.2 Thread Write ... 50
5.2.5.5 Processamento ... 50
5.2.5.6 Formatação ... 51
5.2.6 Parser skyeye ... 52
5.2.7 Parser osciloscópio ... 53
5.3 METODOLOGIA ... 54
5.3.1 Método computação de dados usando simulador ... 54
5.3.2 Computação de dados usando osciloscópio e conversão dos dados ... 56
5.3.3 Método sincronia entre Threads ... 59
5.3.4 Metodologia circuito ... 59
5.3.5 Mecanismo para sincronismo entre PC Embarcado e Osciloscópio ... 60
5.3.5.1 Mecanismo de sincronia para sistema embarcado ... 60
5.3.5.1.1 Sincronia – Início ... 61
5.3.5.1.2 Execução ... 62
5.3.5.1.3 Sincronia – Término ... 62
5.3.6 Mecanismo para sincronismo utilizado para sicronizar ambiente virtual ... 64
5.4 RESULTADOS ... 67
5.4.1 Execução em ambiente real ... 68
5.4.1.1 BLIT – Execução ambiente real ... 68
5.4.1.2 ADPCM – Execução ambiente real ... 69
5.4.1.3 BCNT – Execução ambiente real ... 71
5.4.1.4 CRC – Execução ambiente real ... 72
5.4.1.5 DES – Execução ambiente real ... 74
5.4.2 Execução real versus execução Simulada ... 75
5.4.3 Tamanho ARM versus THUMB ... 77
5.4.4 Análsie dos dados ... 79
6 - CONCLUSÃO E TRABALHOS FUTUROS ... 81
7 - MANUAL ... 82
7.1 INSTALAÇÃO ... 82
7.1.1 Pré - Instalação ... 82
7.1.2 Instalação ... 83
7.2 UTILIZAÇÃO ... 86
7.3 ESTRUTURA DE DIRETÓRIOS ... 87
8 - REFERÊNCIAS ... 88

Figuras

Figura 2-1 Plataforma virtual 14
Figura 2-2 Modelo redefinido 15
Figura 4-1 Exemplo configuração skyeye.conf 22
Figura 4-2 Execução skyeye 23
Figura 4-3 Real Time 25
Figura 4-4 Tempo equivalente 25
Figura 4-5 - Buffer com trigger padrão 26
Figura 4-6 - Buffer com trigger deslocado 26
Figura 4-7 Pipeline ARM7TDMI 28
Figura 5-1 Macro Arquitetura do Core da ferramenta Power Analisys Tool 34
Figura 5-2 Fluxo interno criar arquivos 36
Figura 5-3 Diagrama criar arquivo 37
Figura 5-4 Tipos de scripts possíveis 37
Figura 5-5 Tipos de scripts possíveis 38
Figura 5-6 Fluxo simulação 40
Figura 5-7 Fluxo Temporal execução skyeye 41
Figura 5-8 Código rejeitado 42
Figura 5-9 Código útil 42
Figura 5-10 Código filtrado 42
Figura 5-11 Fluxo execução ambiente real 45
Figura 5-12 Chamada para compilação em ambiente real 45
Figura 5-13 Diagrama módulo gerador script 46
Figura 5-14 Fluxo de sincronia entre threads 47
Figura 5-15 Fluxo Thread Leitura 47
Figura 5-16 Thread Read - Sincronismo– Subfluxo para iniciar execução 48
Figura 5-17 Fluxo para sincronia final 49
Figura 5-18 Diagrama da Thread Write 50
Figura 5-19 Diagrama Processamento 50
Figura 5-20 Formatação osciloscópio 51
Figura 5-21 Parser Skyeye 52
Figura 5-22 Fluxo Parser Osciloscópio 53
Figura 5-23 Fluxo Geral Simulação virtual 54
Figura 5-24 Simulação em ambiente real 56
Figura 5-25 Circuito para computação dos dados 59

Figura 5-26 Fluxo embarcado 60
Figura 5-27 Fluxo comunicação 61
Figura 5-28 Processo de execução 62
Figura 5-29 Processo de término 62
Figura 5-30 Mecanismo de sincronia entre PC e embarcado 63
Figura 5-31 Mecanismo de sincronia entre skyeye e gdb 64
Figura 5-32 Power analisys tool framework 66
Figura 5-33 Potência Média 68
Figura 5-34 Tempo de execução 68
Figura 5-35 Energia consumida 69
Figura 5-36 Potência Média 69
Figura 5-37 Tempo de execução 70
Figura 5-38 Energia consumida 70
Figura 5-39 Potência Média 71
Figura 5-40 Tempo de execução 71
Figura 5-41 Energia consumida 72
Figura 5-42 Potência Média 72
Figura 5-43 Tempo de execução 73
Figura 5-44 Energia consumida 73
Figura 5-45 Potência Média 74
Figura 5-46 Tempo de execução 74
Figura 5-47 Energia consumida 75
Figura 5-48 BLIT - Potência com execução real e execução simulada 75
Figura 5-49 ADPCM - Potência com execução real e execução simulada 76
Figura 5-50 BCNT - Potência com execução real e execução simulada 76
Figura 5-51 CRC - Potência com execução real e execução simulada 77
Figura 5-52 BLIT - TAMANHO 77
Figura 5-53 ADPCM - TAMANHO 78
Figura 5-54 BCNT - TAMANHO 78
Figura 5-55 CRC - TAMANHO 79
Figura 5-56 DES - TAMANHO 79

Tabelas

Tabela 4-1 Opções para configuração 22
Tabela 4-2 Descrição Funcionalidade IO 23
Tabela 4-3 Descrição Execução skyeye 23
Tabela 4-4 Fluxo pipeline Ligar pino P0.4 30
Tabela 4-5 Fluxo pipeline desligar pino P0.4 30
Tabela 5-1 SAÍDA FORNECIDA 33
Tabela 5-2 SAIDA PROCESSADA 43
Tabela 5-3 Canal 57
Tabela 5-4 Benchmark PowerStone 67
Tabela 5-5 Método utilizado para resultados 67
Tabela 5-6 Saída dependente da execução 67
Tabela 5-7 Relação entre instruções e instruções executadas entre Modo THUMB E ARM 79
Tabela 5-8 Relação Tempo e energia elétrica entre modo thumb e modo normal 80
Tabela 7-1 Estrutura de diretórios 87

Equações

Equação 3-1 Energia requisitada 19
Equação 4-1 Pipeline 31
Equação 5-1 Potência total para determinada instrução 44
Equação 5-2 Potência total para determinada instrução 44
Equação 5-3 Potência total para determinado módulo 44
Equação 5-4 Potência total para determinado módulo 44
Equação 5-5 Conversão de dados 57
Equação 5-6 Tempo total 58
Equação 5-7 Potência em Watts 58
Equação 5-8 Energia (W*S) 58
Equação 5-9 Potência média 58

1 - Introdução

Sistemas embarcados podem ser descritos como sistemas que estão inseridos em outros maiores, executando de maneira cíclica determinada tarefa. Porém, podemos separá-los em duas classes principais sendo elas : *hard real time systems* ou *soft real time systems*. Cada classe descrita apresenta objetivos específicos, relacionados ao problema que desejamos resolver.

Na última década ocorreu uma grande explosão no desenvolvimento dos sistemas embarcados, visto que a sua utilidade demonstrou-se uma alternativa menos custosa para resolver problemas do cotidiano. Dessa forma tais sistemas podem ser encontrados de televisores a marca-passos.

Sistemas embarcados são mais restritos do que outras alternativas computacionais disponíveis no mercado, no caso do *personal computer* é totalmente visível as diferenças, porém como o valor de produção e custo de manutenção são mais atrativos eles são usados para determinadas aplicações.

As principais restrições quando estamos utilizando um sistema embarcado é o tempo necessário(sistema de tempo real) para computador algo e a potência consumida. Quando nos referimos a tempo, os sistemas embarcados podem ser divididos em duas categorias que sendo elas : tempo real crítico(*hard real time*), onde uma perda de um *deadline* pode causar um erro catastrófico e tempo real não crítico (*soft real time),* onde a perda de determinado *deadline* não gera uma catástrofe. A preocupação com a potência é necessária, pois é preciso garantir que o sistema consiga se manter energizado tempo suficiente para executar determinada tarefa, ou seja, o mecanismo não pode ser interrompido durante a execução da tarefa, caso a tarefa seja crítica a falta de energia pode gerar acidentes graves.

Dispositivos embutidos geralmente são alimentados por bateria, logo como o consumo de energia é uma restrição que todo sistema embarcado deve possuir é necessário calcular a potência requisitada para determinada operação considerando sempre o pior caso, o qual necessitará de mais energia elétrica. Mesmo que a probabilidade de ocorrência do pior caso seja mínima é necessário satisfazer esse requisito quando estramos tratado de um sistema *hard real time*.

Para resolver problemas relacionados ao consumo de energia de um sistemas embarcado várias técnicas foram desenvolvidas, que vão desde : mesclar códigos dos mais diferentes modos de operação[**5**] de determinado processador até alterar modos de operação dos periféricos que fazem parte do sistema embarcado. Isso é extremamente necessário, pois dispositivos de refrigeração consomem energia, e são responsáveis por

manter a temperatura do *chip* em um intervalo onde é possível obter a estabilidade dos componentes.

O ARM7 desenvolvido em 1994 e foi um dos primeiros processadores arm a fornecerem suporte ao modo “thumb”, o qual se mostra útil quando temos limitações no barramento de memória **[3]** . Apesar do modelo citado ser um dos mais simples no mercado ele é bastante utilizado, podemos constatar isso através das licenças concedidas, as quais que superam o valor de 170 licenças e mais de 10 bilhões de unidades colocadas no mercado[**3**]. O ARM-7 é um microcontrolador bastante versátil, visto que possui dois modos de operações sendo eles : Thumb e o ARM, os quais permitem utilizar instruções de 16 e 32 bits respectivamente. A quantidade de fabricantes e modelos do arm tornam esse microcontrolador mais atrativo, trazendo dessa forma liberdade de escolha para o projetista. A filosofia RISC,*reduced instruction set computing*, a qual o arm pertence torna esse microcontrolador mais simples devido ao *set* de instruções de tamanho fixo e pequeno quando comparado com a filosofia CISC,*complex instruction set computing*.

Skyeye é um projeto *opensource*, que é capaz de simular os seguintes processadores : ARM, MIPS, PowePC, BlackFin, ColdFire e Sparc. Novas versões do SkyEye já permitem emulações de alguns periféricos o que permite agilizar o desenvolvimento de determinado sistema, fazendo testes em ambiente virtual. Esse simulador é bastante interessante, pois permite simular programas em ambiente virtual, isso é bastante interessante, pois em um projeto nem sempre o modelo final do hardware está concluído.

No contexto descrito, esse trabalho de graduação vai abordar o consumo de potência necessária para determinada aplicação, sendo possível realizar uma comparação entre o modelo simulado com o hardware real.

O trabalho de graduação foi dividido da seguinte forma : Capítulo 2 apresenta uma descrição detalhada do problema relacionado ao consumo de energia em sistemas embarcados. O capítulo 3 realiza o estado da arte revelando quais são as pesquisas atuais. O capítulo 4 fornece uma descrição detalhada do skyeye, processador arm e o osciloscópio utilizado no projeto. O capítulo 5 fornece o fluxo, limitações e metodologias aplicada a ferramenta desenvolvida e também é fornecido um estudo comparativo entre o modelo simulado e o ambiente real. No capítulo 6 é realizado a conclusão do trabalho de graduação e trabalhos futuros que pode ser desenvolvidos. No capítulo 7 e 8 finalizamos o trabalhado fornecendo o manual da ferramenta e referências utilizadas.

2 - O problema

Sistemas embarcados possuem como restrição a potência consumida, essa restrição deve ser atendida visto que um grande percentual dos dispositivos embutidos são alimentados por baterias. Na última década é visível o aumento na demanda por sistemas embarcados, isso pode ser analisado por pesquisas que revelam que em 2008 92 bilhões de dólares foram aplicado no mercado de sistemas embutidos e a mesma pesquisa ainda revela um crescimento para 112,5 bilhões de dólares em 2013**[6][7]**.

2.1 **Descrição do problema**

Sistemas embutidos possuem como restrição a energia consumida, pois a maioria dos sistemas embarcados são alimentados por bateria, porém a tecnologia para desenvolvimento de baterias não tem sido acompanhada no mesmo ritmo pela capacidade de processamento do microcontroladores, a qual demanda mais energia elétrica**[9]**.

Para resolver o problema citado diversas técnicas foram desenvolvidas e revelam alguns pontos positivos para redução de consumo de energia elétrica. A maneira mais simples de aferir a potência necessária por um *chip* é em fase de projeto o qual é possível ter acesso a *netlist*, dessa forma é possível verificar com alta precisão o quanto de energia é requisitada pelo *chip*. Porém, apesar dessa técnica ser a melhor para aferir a energia necessária é necessário ter acesso a *netlist*, o que não ocorre na prática.

Para resolver esse outro problema e ainda ser possível otimizar a energia elétrica consumida existe outro processo de aferir a energia, o qual é feito em alto-nível, ou seja a partir de dispositivos que permitem a análise de circuitos, como o caso de um osciloscópio.

Munido dos resultados do consumo de energia elétrica o grupo de desenvolvimento pode fazer alterações no próprio código ou ainda na própria gerência dos periféricos que compõe o sistema embutido. Para tanto, existe inúmeras alternativas para aferir consumo de potência de energia as quais serão discutidas em seções futuras.

2.2 **Justificativa**

Sistemas críticos devem garantir estabilidade e confiabilidade, para que isso seja possível é necessário fazer com que os *deadlines* sejam respeitados, pois a perda de um *deadline* pode gerar um erros catastróficos. Dessa forma, quando estamos trabalhando com sistemas críticos é necessário garantir que a fonte de energia consiga fornecer a energia requisitada de forma a satisfazer as necessidades do sistema crítico**[9].**

Portanto, para garantir que as restrições serão satisfeitas é necessário calcular o pior caso. Logo, para garantir a estabilidade é necessário percorrer todo o fluxo do programa com os valores que estejam relacionados ao pior caso.

Para sistemas reais que não sejam críticos a computação do tempo e potência média trazem um maior controle do projeto, permitindo que seja possível otimizações relacionadas a estabilidade e economia.

Essa necessidade de atender os requisitos citados tem um grande impacto no mercado, já que é possível determinar o grau de confiabilidade e estabilidade de determinado equipamento.

Modelos simuláveis de processadores conseguem fornecer um visão geral das restrições a serem satisfeitas, isso é interessante quando não é possível ter acesso ao hardware final. Simulações em ambiente virtual fornecem maior independência entre as equipes de desenvolvimento de hardware e software. Isso, se torna importante quando estamos utilizando a metodologia que usa plataforma virtual descrita na **figura 2.1 [8]**.

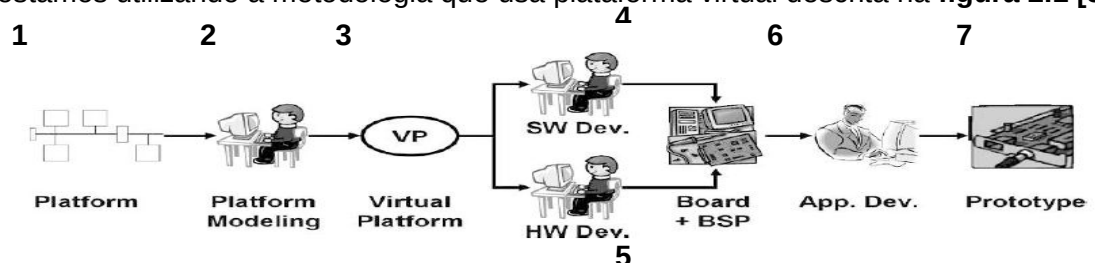

Figura 2-1 Plataforma virtual

1. Fase na qual todos os requisitos são elaborados e toda a plataforma é definida.
2. A plataforma encontra-se definida e seu modelo virtual é construído.
3. Descrição da plataforma real em ambiente virtual.
4. Software básico é desenvolvido baseando-se no modelo virtual.
5. Hardware é desenvolvido em paralelo com o grupo de desenvolvimento de software.
6. Desenvolvedor de aplicações alto-nível
7. Protótipo.

O modelo citado na **figura 2.1** pode ser alterado para um sistema incremental, onde as restrições são testadas ao fim das fases 4 e 5. Onde, é possível redefinir a plataforma no passo 6, fornecendo um modelo da plataforma mais exato.

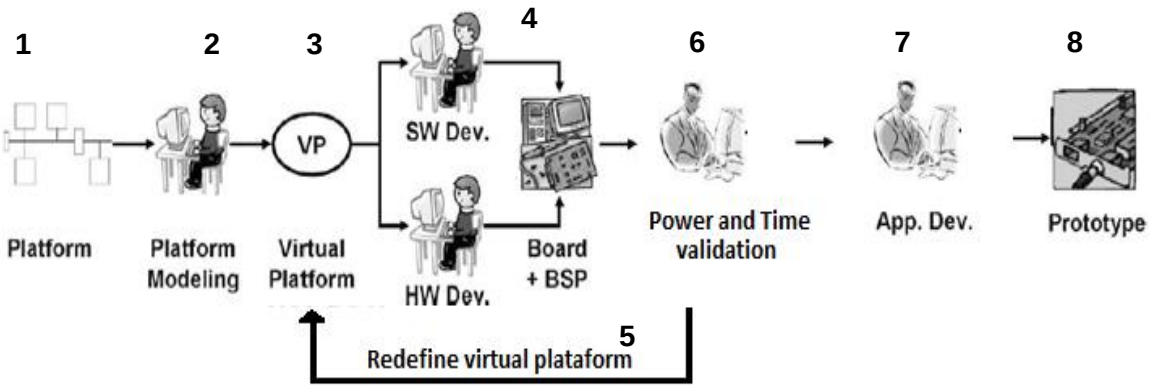

Figura 2-2 Modelo redefinido

Durante a fase 6 descrita na **figura 2.2** , os projetistas podem validar o modelo utilizando informações relacionadas a potência e tempo aferidas de equipamentos de análise de circuitos, com esse resultados em mãos é possível remodelar o ambiente virtual de acordo com dados obtidos, dessa forma é possível que o desenvolvedor do software básico consiga remodelar o sistema para atender as restrições necessárias, enquanto o projetista do hardware trabalha em paralelo. Apesar dessa alternativa permitir uma maior independência entre o grupo de desenvolvimento as simulações feitas no passo 4 ainda serão de alto nível, mas serão mais exatas devido ao modelo incremental proposto.

Logo, para aferir restrições relacionadas a tempo e a potência requisitadas é necessário a criação de mecanismos que consigam computar tais dados. Munido de tais informações é possível abandonar ou mudar determinada estratégia para desenvolvimento de determinado equipamento, minimizando gastos desnecessários durante o projeto.

3 - Estado da arte

Inúmeras são as contribuições para tentar aferir o consumo de potência e energia por um sistemas embarcado. As primeiras tentativas remontam ao ano de 1994 **[15]** , onde é realizada a caracterização de consumo para processadores 486dx2 e fujitsu Sparclite 934.

Outros modelos fazem uso de sistemas mais precisos para caracterização do consumo de energia. Para isso, os desenvolvedores fazem uso de osciloscópios para aferir os dados [**13**] e **[10]**.

Modelos mais recentes de caracterização fazem uso de ferramentas de debug e simuladores para extrair a quantidade de ciclos necessários e quantidade de acessos as memórias de instruções e dados. Munidos com as informações exibidas os autores podem computar o consumo gasto com a memória de instruções e dados[**17**]. Este capítulo tem por fim analisar o que foi feito e quais são os modelos de caracterização de potência mais relevantes para esse trabalho.

3.1 **Power Analysis of Embedded Software: A First Step Towards Software Power Minimization**

Devido a dificuldade de calcular a energia consumida por um processador , o qual não dispomos de informações relacionadas ao nível estrutural inúmeras técnicas foram criadas. A técnica apresentada por [**15**] revelam os primeiros trabalhos nessa área.

O trabalho sugere a hipótese de que podemos calcular o consumo de energia elétrica por determinado conjunto de instruções a partir da execução dessas instruções.

Para isso, os criadores do trabalho usaram dois processadores sendo eles : 486DX2-S e Sparclite. Fazendo uso de um amperímetro digital o autor define a potência média para determinado programa e a quantidade de ciclos necessários. Após, ter realizado a captura de informações relacionadas a tempo e potência é possível descobrir a energia elétrica por determinado programa.

Para aferir os dados é necessário fazer o uso de um *loop* infinito logo após a execução do programa de forma que seja possível obter uma leitura da corrente consumida, o autor faz uso de uma janela de 100ms para aferir os dados. Vemos que os autores já se preocupam com o tamanho da janela, visto que só é possível aferir programas que possuem no máximo 100ms. Para calcular o custo base de cada instrução o autor criou um loop com inúmeras instâncias da mesma instrução e calculou a potência

média, ou seja, o custo base é a potência média de uma instrução. Outra preocupação que o autor revela é a quantidade de instruções que o *loop* pode possuir, pois é necessário garantir que não ocorrerá exceções lançadas pela memória cache de instruções.

O autor define o custo base de um programa como sendo o somatório dos custos básicos de todas as instruções executadas. Apos definido o setup completo o autor verifica que outros fatores alteram a potência consumida e devem ser calculados. Sendo eles : comutação entre instruções e variações de determinada instrução.

O primeiro fator a ser avaliado é a variação de parâmetros de determinada instrução, o autor verificou que alterações com registradores são mínimas, porém isso não ocorre quando estamos trabalhando com operações que usam *immediate*. Foi verificado que quanto maior a quantidade de 1's (binário) no *immediate* menor vai ser a potência consumida. Quando ele se refere a instruções de movimentação é concluído que o alinhamento dos dados pode interferir no consumo da potência.

Outro fator que o autor verifica é o mecanismo a potência consumida pela alternância entre duas instruções. Dessa forma, verifica-se que o consumo de potência por determinado programa não é apenas dado pelo custo base, mas que depende também da comutação entre circuitos internos do core.

Afim de modelar um sistema que seja capaz de sobrepor o problema relacionado a potência entre instruções. Um modelo mais robusto é utilizado, nesse mecanismo é calculado o custo base de duas instruções. Outro fator importante que pode aumentar o consumo de energia são exceções lançadas pela cache de instruções, a qual adiciona mais potência consumida, devido a mudança de contexto.

Para aferir a corrente consumida pela memória o autor verifica que existe um relacionamento entre o modo de operação e energia requisitada. O consumo por parte da memória também depende do programa que está sendo executado, visto que quanto maior o número de acesso a memória maior vai ser o consumo .

3.2 Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors

Trabalho que faz uso do setup utilizado em [**15**] com algumas modificações. Para medir a corrente o grupo usa um resistor em série com vdd núcleo do processador, fizeram uso também de capacitores para desacoplamento, visto que a introdução do resistor insere ruído no caminho.

Para calcular a potência consumida pelo processador é usado a diferença de potencial no resistor, dessa forma, é possível descobrir a corrente passante para o processador. Utilizando a corrente requisitante pelo *core* e a tensão de entrada do processador é possível calcular a potência consumida, enquanto o tempo é calculado utilizando um osciloscópio permitindo dessa forma, calcular a energia elétrica.

Para calcular a potência base e o efeito relativo ao intervalo entre duas instruções consecutivas o grupo coloca as instruções a serem avaliadas dentro de um *loop,* de forma que possam ser computadas, sendo que são utilizadas 100 instruções por vez. A partir dos resultados é obtido a média da potência consumida.

Para computar os dados ao invés de usar a média aritmética o autor verifica que a aplicação da mediana e usando *semi-interqualite* (corresponde a mediana superior subtraído da mediana inferior , após a subtração o resultado é dividido por dois) revelam resultados mais interessantes.

A partir disso os autores propõe modelar a potência como um parâmetro constante utilizando inferência estatística e usando a distribuição *t-student* com coeficiente de confiança de 99%. Porém, com isso não estaremos dizendo que a média cairá 99 % das vezes em determinado intervalo, mas afirmaremos que a esperança estará contida no intervalo, ou seja o intervalo de confiança calculado terá probabilidade de 99% de possuir determinada esperança.

3.3 Measurement of Current Variations for the Estimation of Software-Related Power Consumption

Esse modelo propõe um novo circuito para mensurar a corrente consumida, para isso o grupo desenvolve um circuito capaz de realizar o espelhamento da corrente que vai para para o *core*, ou seja o circuito desenvolvido tem capacidade de realizar uma "cópia idêntica" consumida pelo *core*. Dessa forma, é possível descobrir a potência e energia elétrica consumida pelo processador. Para tanto, a energia requisitada é descrita como sendo o somatório da corrente de base mais custo entre instruções e exceções lançadas pela cache ou *stalls* gerados no pipeline.

$$E_p = \sum_i (B_i * N_i) + \sum_{i,j} O_{i,j} * N_{i,j} + \sum_h E_h$$

Equação 3-1 Energia requisitada

Onde, o primeiro termo representa o somatório do custo base de determinada instrução multiplicada pelo número de ocorrências, já o segundo termo representa o somatório do conjunto de instruções multiplicadas pelo número de ocorrências e o último termo representa a contribuição ocasionada por stalls e faltas geradas pela cache.

Para aferir a corrente consumida os autores utilizam osciloscópio para capturar os dados fornecido pelo circuito desenvolvido dessa forma é possível analisar os dados.

4 - Ferramentas

Neste capítulo vamos abordar as ferramentas utilizadas de forma a fornecer uma visão geral das ferramentas e como elas foram utilizadas. Para isso, apresentaremos como elas são utilizadas nesses projeto.

4.1 **Skyeye**

Nesta seção vamos apresentar características do simulador de processador utilizado para esse projeto e o setup utilizado no projeto.

4.1.1 **Característica**

Skyeye é um simulador *opensource* capaz de simular os seguintes processadores :

1. ARM
2. MIPS
3. PowerPC
4. BlackFin
5. Coldfire
6. SPARC

O Skyeye se destaca ainda mais, pois permite executar sobre o simulador sistemas operacionais como Linux, uCLinux, uc/OS-II, rtemse Ecos com suporte a debug.

Versões mais novas fornecem suporte a periféricos como : Timer, Uart, LCD, TouchScreen, etc.

4.1.2 **Instalação**

Utilizamos a versão *1.2* do skyeye para o *debian lenny 5.0.* Abaixo mostramos o comando para instalar o skyeye :

apt-get install skyeye

O comando acima instalará a versão disponibilizada no repositório da distribuição linux debian. Caso seja necessário instalar o skyeye a partir do source é necessário executar os passos abaixo :

Download :

http://sourceforge.net/projects/skyeye/files/skyeye/skyeye-1.3.2_rc1/skyeye-1.3.2_rc1.tar.gz/download

Descompactar :

tar –xvzf skyeye-1.3.2_rc1.tar.gz

Instalação :

cd skyeye-1.3.2_rc1

./configure

make

make install

4.1.3 **Configuração**

Para que o simulador funcione corretamente é necessário configurá-lo de acordo com as especificações do processador que queremos utilizar. Portanto, abaixo fornecemos uma descrição detalhada da configuração para simular o arm7tdmi.

É necessário criar um arquivo de configuração, por padronização utiliza-se o nome de “skyeye.conf”, esse arquivo conterá a descrição da arquitetura, modelo do processador, descrição sobre memórias e periféricos. Na **tabela 4.1** podemos ver a

descrição detalhada de cada opção e na **tabela 4.2** é realizado a descrição para configuração de dispositivos de IO.

Tabela 4-1 Opções para configuração

Opção	Descrição
arch	Arquitetura a ser usada
cpu	Processador que desejamos usar no projeto
march	Especifica o SOC
membank	Utilizado para descrever o espaço de memória e mapeamento de IO
map	Opção utilizada distinguir entre endereço de dados e IO . Valores com “M” representa memória e valores com “I” representa IO
type	Descreve se o espaço de memória é apenas leitura ou fornece suporte para leitura e escrita. Valores com “RW” permitem a escrita e a leitura e “RO” apenas leitura
addr	Fornece o endereço base de memória.
size	Especifica o tamanho da região da memória em bytes.

```
arch:arm
cpu: arm7tdmi
mach: at91
mem_bank: map=M, type=RW, addr=0x00000000, size=0x00007FFF
mem_bank: map=M, type=RW, addr=0x40000000, size=0x40001FFF
mem_bank: map=I, type=RW, addr=0xF0000000, size=0x0FFFFFFF
```

Figura 4-1 Exemplo configuração skyeye.conf

Neste projeto utilizamos a configuração apresentada no manual e *datasheet* para o processador **[11][12]** . Apesar da região de memória de IO ser definida, ela não é usada no nosso projeto.

Tabela 4-2 Descrição Funcionalidade IO

Opção	Descrição
net	Opção utilizada para configurar uma placa de rede
type	Especifica o tipo de placa de rede. As placas disponíveis são : cs8900a e rtl8019
base	Especifica o endereço base da placa de rede
size	Especifica o tamanho da região de memória ocupada pela placa de rede
int	Especifica a interrupção associada a placa de rede
mac	Especifica o endereço MAC da placa de rede
ethmod	Conexão ponto a ponto. O único valor possível é tuntap
hostip	Especifica o IP da placa de rede
lcd	Opção utilizada para configurar o lcd
type	Especifica o tipo, podendo assumir valores como s3c2410x ou ep7312
mod	Especifica a biblioteca gráfica a ser usada para desenhar a interface do LCD. Suporte apenas para GTK

Execução :

O skyeye permite a execução através de dois modos sendo eles :

- Normal – Usado para opera em máquina local
- Daemon – Usado para depurar remotamente

Tabela 4-3 Descrição Execução skyeye

Opção	Descrição
-d	Executa skyeye em modo daemon
-c	Entra com local da configuração
-e	Executa skyeye

Exemplo:

skyeye –e –c /home/toolchain/skyeye/skyeye.conf –d

Figura 4-2 Execução skyeye

Para maiores informações sobre skyeye consultar o manual do skyeye**[14].**

4.2 **Osciloscópio DSO3000**

A série DSO3000 de osciloscópio possui quatro modelos sendo eles :

- DS03062A
- DSO3102A
- DSO3152A
- DSO3202A

Apesar da série possuir os 4 modelos citados e a programação serem identificas abordaremos apenas o osciloscópio DSO3102A.

O osciloscópio DSO3102A é considerado um osciloscópio de baixo custo, o qual permite até 1Gsa/s (1 Giga Amostras por segundo), a memória que ele possui tem capacidade de armazenar 4000 amostras ou 4000 Bytes, isso é possível pois cada amostra possui 8 bits.

O modelo possui várias características interessantes que são relacionadas a funções matemáticas que permitem adicionar, subtrair e multiplicar sinais, outra função interessante é a função FFT, *fast fourier transform*, a qual permite visualizar sinais no domínio da frequência. Esse osciloscópio também possui filtros como passa baixa, passa faixa, passa alta e rejeita banda, esses filtros são interessantes para eliminar ruídos existentes, ou seja rejeitar determinadas frequências.

O modo de aquisição deste dispositivo pode ser feita através de tempo real, onde as amostras são capturadas em intervalos regulares e tempo equivalente, o qual cria amostras com *delays* variáveis a partir do trigger. Na **Imagem 4.3[2]** podemos visualizar o modo de tempo real e na **imagem 4.4[2]** podemos visualizar o modo equivalente **[2]**.

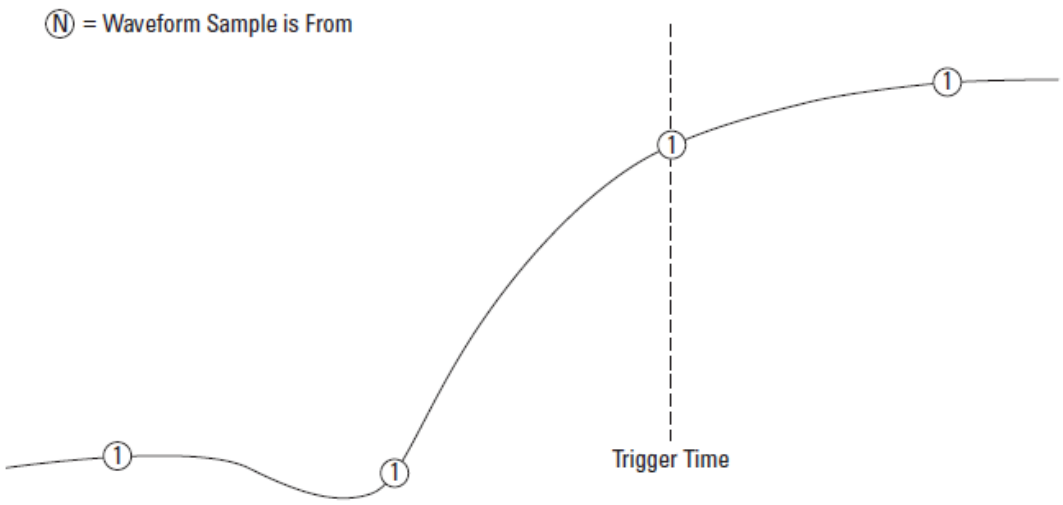

Figura 4-3 Real Time

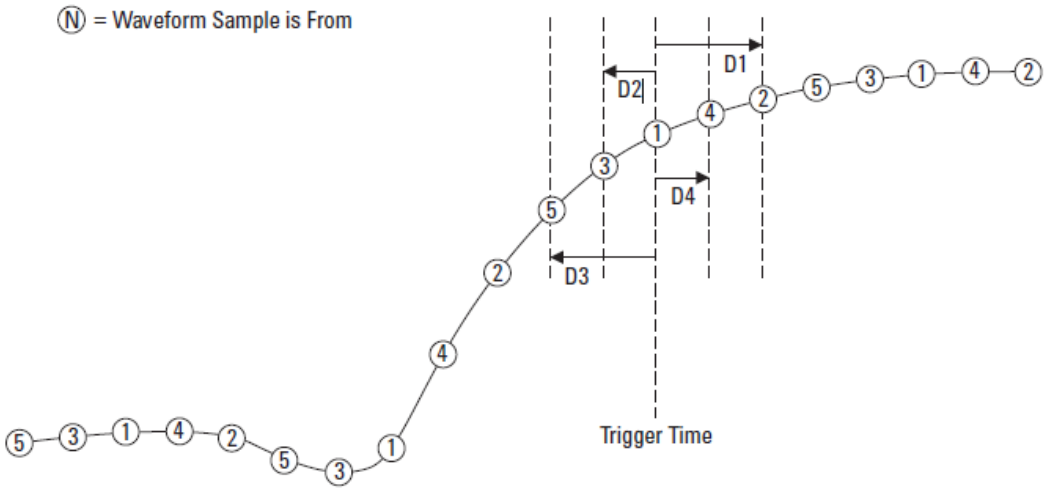

Figura 4-4 Tempo equivalente

Assim, como qualquer outro osciloscópio moderno ele faz uso de *triggers*, ou seja, é possível exibir na tela apenas os dados a partir de determinada condição ser verdadeira. Neste projeto utilizamos o trigger configurado no canal 2, a condição que escolhemos é que a voltagem precisa ser maior ou igual a 3.1 volts. Também, podemos escolher o deslocamento dentro do buffer do osciloscópio a ser utilizado. No projeto deslocamos o *trigger* horizontal para aumentar a janela de dados que podemos capturar.

Nas **figuras 4.5** e **4.6** mostramos os buffers e sua utilização nesse trabalho.

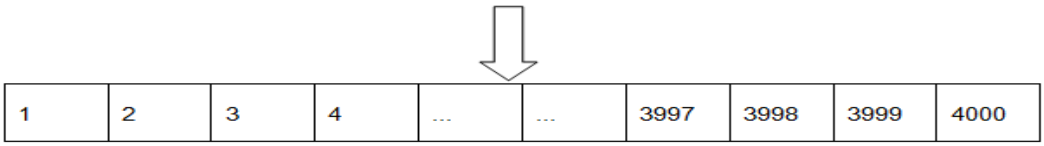

Figura 4-5 - Buffer com trigger padrão

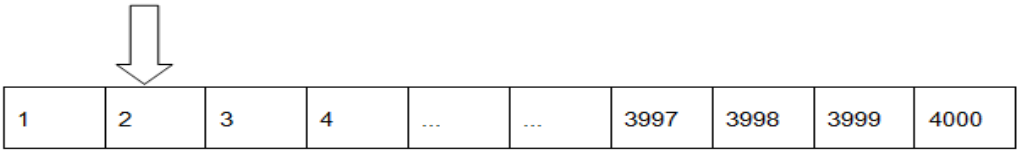

Figura 4-6 - Buffer com trigger deslocado

O *trigger* preenche a memória disponível até metade do *buffer*, ou seja, se possuímos um buffer de 4000 bytes utilizamos apenas 2000 bytes enquanto o *trigger* não ocorrer. Quando ocorre um *trigger*, caso ele não esteja deslocado, é plotado no visor os valores aferidos antes do *trigger* e o valores aferidos depois do trigger. Isso é um problema para o nosso projeto, pois não desejamos saber o comportamento do sinal de ativação antes do trigger, no nosso trabalho só temos interesse dos dados após a geração do trigger. Caso, usassemos a maneira convencional estariamos perdendo dados, ou seja estariamos diminuindo nossa janela de dados.

Para resolver o problema citado, deslocamos a posição do *trigger* na escala temporal para aumentar a quantidade de dados que podemos utilizar. Apesar do trigger ser ativado com 3.1 volts capturamos apenas valores que são maiores 3.1 volts e menores que 3.46 volts, pois dessa forma conseguimos ignorar *glitch* ocasionados pela comutação do circuito interno. Nesse projeto utilizamos a porta, P0.4, do LPC2103 pertencente a placa de desenvolvimento MCB2103 da keil como ativador do *trigger*, ou seja o canal 2 do osciloscópio fica responsável por computar a ativação e desativação do trigger. Os dados são transferidos para o computador usando a porta USB, com isso podemos computar de maneira automática os dados sem ser necessário utilizar o

software disponibilizado pelo fabricante. O modelo DSO3102A pode ser programado para utilizar suas funções a partir do computador, na seção 5.3.2 descrevemos como computar os dados e aferir os dados a partir do osciloscópio.

4.3 **ARM7TDMI-S**

O ARM é um processador de 32 bits de propósito geral que implementa arquitetura RISC, *reduced instruction set computer*, a qual incorpora arquitetura load/store e tamanho fixo de instrução.

O ARM7TDMI possui pipeline de três estágios, os quais são exibidos na **imagem 4.7** e explicados logo em seguida **[4]**.

Figura 4-7 Pipeline ARM7TDMI

- FETCH – Captura próxima instrução da memória
- DECODE – Decodifica a instrução capturada do passo anterior
- EXECUTE – Executa a instrução decodificada. Aciona circuitos internos para processamento da instrução.

O ARM possui suporte a 7 modos de operação são eles :

1. User – Operação Normal
2. FIQ - Usado para operações que precisem de mais agilidade
3. IRQ – Usado para tratar exceções
4. Supervisor – Modo protegido para sistema operacional
5. Abort – Modo responsável pela proteção da memória
6. Undefined – Usado para emulação de coprocessador, quando não existe, ou instrução inexistente.
7. System – Modo privilegiado para gerenciador de tarefas

Assim, como os modos de operação as instruções do ARM são divididas de acordo com as suas funcionalidades, são elas :

1. Instruções de salto
    a. São instruções que conseguem fazer o deslocamento para outras regiões do código
2. Instruções de processamento
    a. Instruções responsáveis por operações matemática e comparações
3. Instruções de movimentação para registradores CPSR e SPSR
    a. Envia ou captura dados do registradores citados
4. Instruções de operação com a memória
    a. Instruções que movimenta dados da memória para registradores
5. Instruções de processamento do coprocessador
    a. Instruções que manipulam coprocessador.
6. Instruções que geram exceções
    a. Instruções que geram exceções para executar determinada funcionalidade como : chamar sistema operacional ou executar programa passo a passo.

Nesta seção vamos abordar o processador ARM7TDMI-S utilizado durante esse projeto. O ARM utilizado faz uso da versão 4 do conjunto de instruções sendo um dos primeiros modelos a ter suporte ao modo THUMB.

O ARM possui registrador especial para *flags* denominado de CPSR, ele possui 32 bits e dependendo de cada estado de suas flags o fluxo de execução pode ser alterado. Para maiores informações consultar manual do ARM**[4][11][12].**

Outro fator inovador agregado ao ARM é a possibilidade de uso de instruções de 16 bits, as quais estão relacionadas ao modo THUMB, o qual possui duas versões, sendo elas : THUMBV1 e THUMBV2. Porém, neste trabalho abordaremos apenas a primeira versão.

O modo THUMB é interessante quando queremos aumentar a densidade do programa, ou seja com o modo que usa 16 bits é possível criar executáveis menores, a redução é tipicamente de 35%**[4]** . Apesar da vantagem citada, esse modo agrega o problema de que a computação é mais custosa, ou seja é necessário executar mais instruções para realizar a mesma tarefa quando comparada com modo normal **[4].**

O modo THUMB apresenta vantagem em relação ao modo ARM quando é utilizado memórias com largura de 16 bits, a diferença em performance chega a 160% em relação a operação em 32 bits.

Com o que foi apresentado podemos apresentar como a mensuração do programa foi realizada. Como descrito na **figura 4.7,** o ARM7TDMI possui um pipeline com 3 estágios, por causa disso inserimos *nop's* para capturar os dados. Isso é feito, pois dessa forma garantimos que o pipeline só vai possuir instruções que conhecemos a **tabela 4.4** exibe o pipeline em função do tempo.

Tabela 4-4 Fluxo pipeline Ligar pino P0.4

Operação	Fetch	Decode	Execute
1	NOP	DC	DC
2	NOP	NOP	DC
3	MOV ##,##	NOP	NOP

Quando a instrução MOV ##,## chega ao estado, *execute,* o pino da porta P0.4 é ligado e o osciloscópio começa a capturar os dados. Para desligar o pino usamos o mecanismo inverso.

Tabela 4-5 Fluxo pipeline desligar pino P0.4

Operação	Fetch	Decode	Execute
1	NOP	DC	DC
2	NOP	NOP	DC
3	NOP	NOP	MOV ##,##

Na operação 3 da **tabela 4.5**, a instrução MOV ##,##, responsável por desligar o pino P0.4, é executada. O pipeline possui duas instruções, as quais também consomem potência visto que :

$$\boldsymbol{E_{pipeline} = E_{fetch} + E_{decode} + E_{execute}}$$

Equação 4-1 Pipeline

Com isso, vemos que o processo de desligar os pinos acarreta a soma do consumo de potência nos estados *Fetch* e *Decode*. Na *seção 5.2* apresentamos o fluxo da ferramenta e na *seção 5.3* apresentamos os métodos usados para computar os dados.

5 - Proposta

Inúmeras técnicas foram demonstradas, porém nenhuma das técnicas exibidas apresenta uma maneira automática para criar arquivos fontes, scripts de compilação e aferir dados como tempo, potência e energia elétrica para diversas iterações. A fim de tentar melhorar a computabilidade das medidas de potência, energia e tempo de execução de determinada funcionalidade foi desenvolvido uma ferramenta que tem como por fim fornecer de maneira automática resultados a partir da execução de programas em ambientes reais ou ambientes virtuais.

5.1 **Objetivos**

Esse trabalho tem como objetivo fornecer um mecanismo automático para computar a potência, energia elétrica e tempo consumido para execução de determinada programa. Para tanto, a comparação é feita usando o processador ARM operando em modo Normal, com instruções de 32 bits, e modo Thumb, o qual opera com instruções de 16 bits.

A fim de computar os dados de maneira correta é necessário que o usuário especifique de maneira explícita o que é desejado. As entradas possíveis são exibidas abaixo.

- Número de iterações desejadas (1,2,3,4 ....)
- Modelo do ARM (arm7tdmi-s ou arm7tdmi-s)
- Modo operação (-marm ou –mthumb)
- Execução desejada (0 – Nenhuma, 1 – Ambiente Virtual, 2 – Ambiente real)

A partir dos resultados fornecidos pela, *power analisys tool*, é possível analisar em tempo de projeto o modo a ser usado por determinada aplicação. Para que seja possível a análise dos resultados a ferramenta fornece saídas, as quais estão descritas na **tabela 5.1** classificadas a partir do ambiente desejado (virtual ou real) .

Tabela 5-1 SAÍDA FORNECIDA

Ambiente Real	Ambiente Virtual
Número de cada amostra	Número de vezes que determinada instrução foi executada
Potência individual de cada amostra	Potência individual de cada instrução
Potência média individual de cada amostra	Potência total de cada instrução individual
Energia individual de cada amostra	Potência total necessária
Tempo de execução de determinada amostra	
Número de pontos para cada amostra	
Potência total	
Potência média total	
Energia elétrica total	
Tempo total de execução	
Número total de amostras	

5.2 **Fluxo**

A ferramenta faz uso de vários fluxos de controle. Para facilitar a explanação será usado sistemas de hierarquias. Portanto, será usado uma abordagem *top-down*, ou seja explicaremos os macro módulos e após isso será explicado os micro módulos.

5.2.1 **Fluxo Principal**

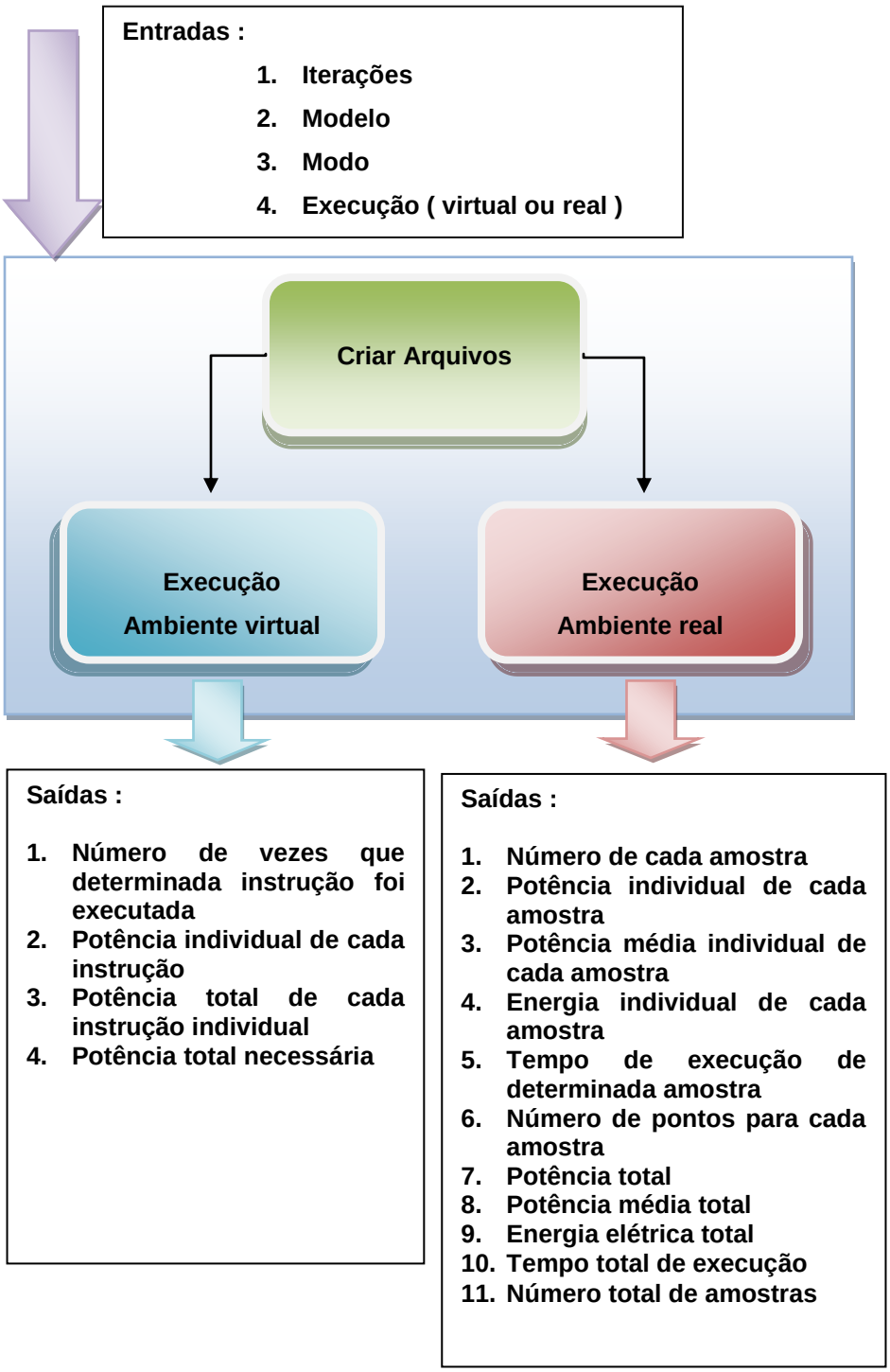

Figura 5-1 Macro Arquitetura do Core da ferramenta Power Analisys Tool

5.2.1.1 Criar arquivos

O módulo, criar arquivos, tem como finalidade criar os *sources* a partir do modo que se deseja executar. Ou seja, nessa fase os *sources* são gerados, tanto para o modo de execução real quanto para o virtual. A criação desses arquivos fontes são diferentes, pois para simulação em ambiente real é necessário a criação de arquivos em formato binário e a chamada para uma função compilada como arm é diferente da chamada para uma função compilada para modo thumb.

Apesar de ser realizado a distinção entre ambiente real e virtual, nessa fase só os arquivos finais são criados, mas os arquivos criados não são compilados e executados, ou seja nessa fase os arquivos finais em formato elf e binário não são criados. Na *seção 5.2.2* fornecemos mais detalhes sobre o módulo criar arquivos.

5.2.1.2 Execução Ambiente Virtual

Sistema responsável pela execução do programa selecionado em modo de simulação. Tem como objetivo fornecer dados a partir de simulações ocorridas em software. Dessa forma, permite que o software seja testado sem a necessidade do hardware real.

Com o modelo virtual do processador a equipe de software pode trabalhar em paralelo com a equipe de hardware. Apesar de ser necessário reajustar a plataforma virtual inicial podemos ter um grau maior de paralelismo relacionado ao trabalho das equipes envolvidas.

A execução de testes é extremamente lenta, mas isso pode ser contornado oferecendo depuração remota. Dessa forma, várias funcionalidades que não possuam dependência entre si podem ser executadas. Isso, fornece um maior grau de agilidade para obtenção dos resultados. Porém, esse mecanismo não foi implementado na versão beta 1.0 da ferramenta *power analisys tool*.

O módulo tem por fim tratar todas as saídas geradas pela depuração em informação útil, para que possa ser analisada posteriormente. Na *seção 5.2.3* fornecemos mais detalhes o processo de execução do programa em ambiente virtual .

5.2.1.3 Execução Ambiente Real

Entidade que tem como objetivo realizar toda configuração desejada para aferir os dados desejados. O resultado de cada iteração executada é analisada e armazenada para posterior processamento. Com isso, podemos criar um grau maior de liberdade entre as amostras. Na *seção 5.2.4* é fornecido mais detalhes sobre esse módulo.

5.2.2 **Fluxo Criar arquivo**

5.2.3

O módulo, criar arquivo, é dividido em três módulos sendo eles :

1. Escrever *sources* para ambiente
2. Escrever scripts de compilação
3. Escrever scripts para gdb

É possível também visualizar a hierarquia do módulo, criar arquivo, através da **Figura 5.2**, o qual será explicado nas próximas seções.

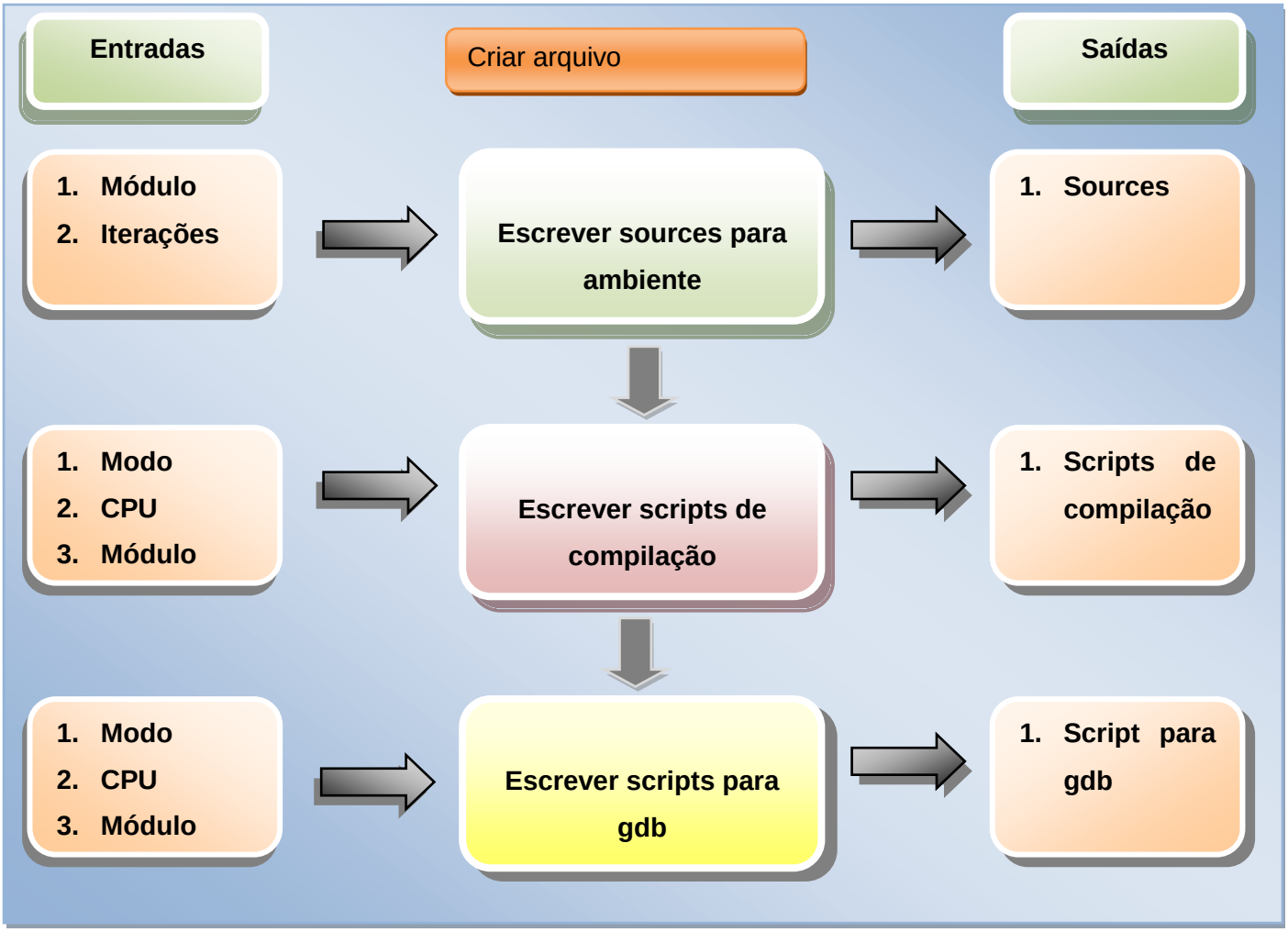

Figura 5-2 Fluxo interno criar arquivos

5.2.3.1 Escrever Arquivos Para Ambiente

Essa fase tem como finalidade criar *sources* para execução em ambiente virtual e execução para ambiente real. O programa gerado para o simulador e dispositivo real possuem ligeiras diferenças, portanto os dois modelos são criados.

Os modelos criados diferem entre si, pois é necessário criar para o ambiente real um mecanismo de sincronia, esse sincronismo é necessário pois a geração de dados é mais ágil do que a transferência de dados do osciloscópio para o PC. A sincronia entre o PC e sistema embarcado é explicado nas *seções 5.3.5 e 5.2.4.4* .

Na **figura 5.3,** pode ser visto o fluxo de criar arquivos.

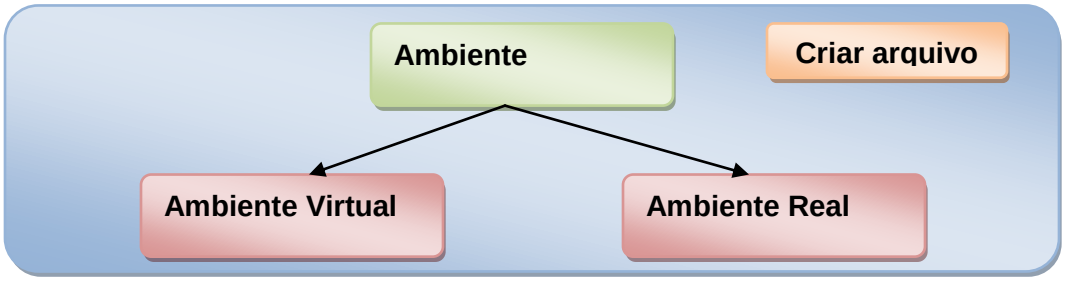

Figura 5-3 Diagrama criar arquivo

5.2.3.2 *Escrever scripts para compilação*

Nessa fase do fluxo “criar arquivo”, os scripts de compilação são criados a partir do que foi solicitado, esses scripts tem como função gerar tanto arquivos em formato ELF como em binário.

Para simplificar a criação dos arquivos para execução foi escolhido separar a criação de script para o sistema embarcado e a simulação em ambiente virtual. Para isso é fornecido o fluxo abaixo do que é possível gerar a partir das entradas.

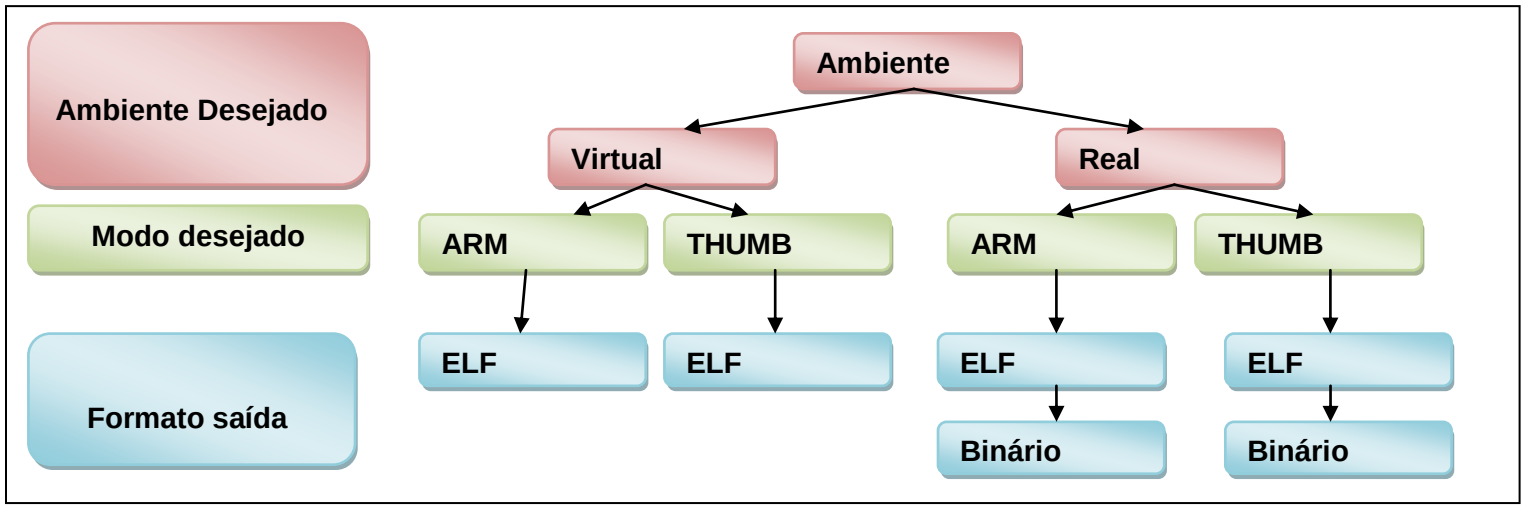

Figura 5-4 Tipos de scripts possíveis

A **figura5.4** revela que é possível gerar diferentes tipos de scripts para compilação dos *sources*. Apesar de ser necessário de forma explícita dizer o modelo do ARM que se deseja a ferramenta só fornece suporte ao arm7tdmi e o arm7tdmi-s.

A partir do modo desejado ARM ou THUMB é possível gerar arquivos no formato ELF que respeitem o modo desejado.

Não é possível fazer o interworking entre thumb e arm, visto que desejamos analisar apenas os modos separados, ou seja não é permitido o módulo desejado ter funções em modo arm e em modo thumb ao mesmo tempo.

Quando desejamos criar arquivos que rodem em ambiente real uma fase a mais é acrescentada, tal fase é responsável por criar o arquivo em formato binário de forma que seja possível fazer o futuro download para o sistema embarcado.

5.2.3.3 Escrever scripts para gdb

Nessa fase do processo de "criar arquivo" é gerado um script para o gdb de forma a automatizar todo processo de captura de instruções executadas, na **figura 5.5** é possível fornecer uma visão geral do que esse script realiza. E sua descrição é realizada em seguida.

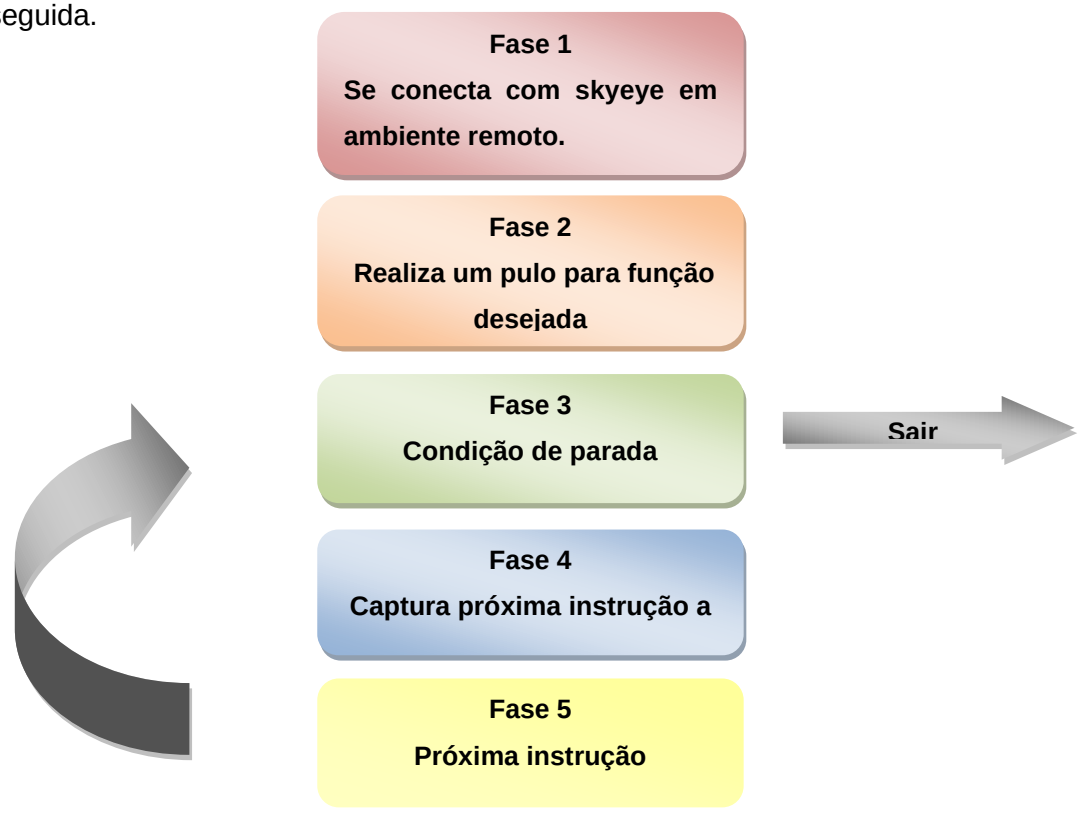

Figura 5-5 Tipos de scripts possíveis

A partir da **figura 5.5** podemos dividir o script em várias seções, cada uma responsável pela execução de determinada tarefa. Abaixo é realizado a descrição de cada fase.

- Fase 1 – A instância do gdbclient se conecta ao servidor, o qual pode estar localizado em outra máquina.
- Fase 2 – É Realizado o *jump,* para função que desejamos verificar quais instruções são executadas
- Fase 3 – É verificado se a condição de parada é satisfeita, caso sim, a instância do servidor do skyeye é retirada da memória e todas as instruções desejadas se encontram em um arquivo chamada “gdb.txt”, o qual será processado futuramente pelo parser descrito na **seção 5.2.5***.* A condição de parada é feita verificando-se qual endereço de memória a execução se encontra.
- Fase 4 – Verifica qual é a próxima instrução a ser executada e grava a instrução no arquivo.
- Fase 5 – Retorna a fase 3.

5.2.4 **Fluxo Ambiente Virtual**

Após ter explicado a fase onde são criados os *sources* e scripts de compilação podemos abordar as fases responsáveis por executar a simulação com skyeye.

A figura baixo descreve o fluxo realizado quando se deseja computar o resultado através de simulações . Detalhes de cada fase é feito nas próximas seções.

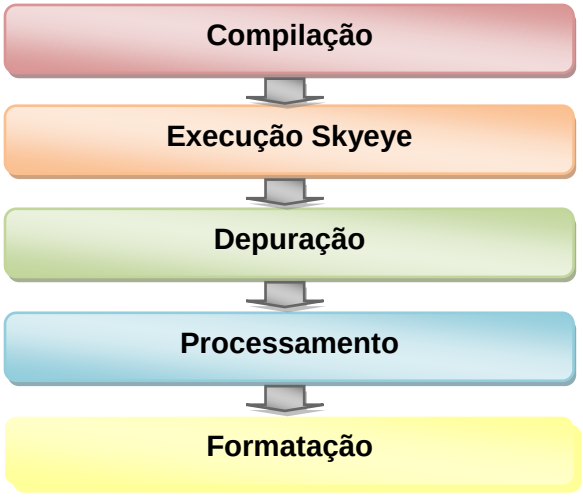

Figura 5-6 Fluxo simulação

5.2.4.1 Compilação

Essa fase tem como finalidade executar o script de compilação. Tal script é gerado de acordo com os argumentos que o usuário fornece. A geração dos arquivos de compilação pode ser encontrada na *seção 5.2.2* deste trabalho.

A execução do script é feita através da chamada do sistema operacional - system. Por questões de padronização a ferramenta, *power analisys tool*, fará uso da função execl em suas futuras versões.

A chamada é realizada da seguinte forma :

```
sprintf(name_script_skyeye,"../output/%s/./script_skyeye",module);
system(name_script_skyeye);
```

Onde module corresponde ao nome do módulo desejado. Caso o módulo se chamasse blit a execução a chamada ao script ocorreria da seguinte forma :

```
sprintf(name_script_skyeye,"../output/%s/./script_skyeye",module);
system("../output/blit/./script_skyeye");
```

A estrutura de diretórios será descrita na *seção 7.3.*

5.2.4.2 Execução Skyeye

Fase na qual o skyeye é invocado . Na versão 1.0 da ferramenta, *power analisys tool*, só é possível executar o skyeye localmente. As próximas versões vão incluir suporte a execução remota. A ferramenta suporta depuração remota, porém não de uma maneira automatizada, portanto consideramos que a versão 1.0 só possui depuração local.

A instância do SkyEye funciona em modo *daemon*, ou seja a execução do programa desejado só é iniciado quando uma instância do gdb se conecta ao gdbserver do skyeye.

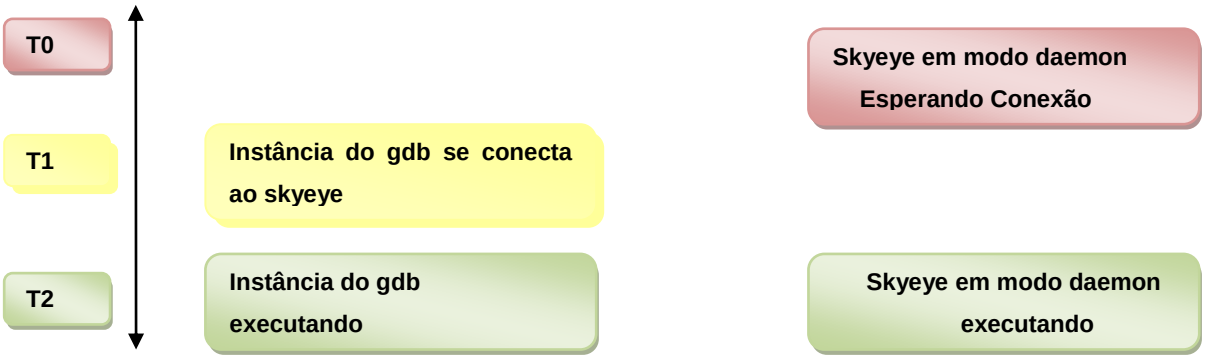

Figura 5-7 Fluxo Temporal execução skyeye

5.2.4.3 Depuração

Fase na qual a instância do gdb se conecta ao skyeye e executa passo a passo a execução do programa desejado no simulador.

Todas instruções executadas são armazenadas para que possam ser analisadas nas próximas fases.

A transparência dessa fase para o usuário só é possível através da geração de um script para o gdb. O qual possui instruções como o gdb deve executar. A descrição da criação do script para o gdb é realizado na *seção 5.2.2.3.*

5.2.4.4 Processamento

Fase na qual é realizado o processamento da saída do gdb. Tem como função isolar apenas o código desejado. Abaixo é demonstrado parte da saída da execução do módulo blit do *benchmark powerstone*.

```
Program received signal SIGHUP, Hangup.
=> 0x14c <label_dont_use_2+2>:    adds    r3, r7, #0
Can't send signals to this remote system.  SIGHUP not sent.
```

Figura 5-8 Código rejeitado

Vemos que no trecho de código acima é possível a visualização do label "label_dont_use_2", o qual instrui ao *parser* não fazer uso das instruções que possuem esse label como modificador. Dessa forma, podemos realizar a filtragem da saída.

Outro exemplo de código que se encaixa no perfil desejado é descrito abaixo.

```
Program received signal SIGHUP, Hangup.
=> 0x30e <blit_6+342>:        adds    r2, #24
Can't send signals to this remote system.  SIGHUP not sent.
```

Figura 5-9 Código útil

O perfil da instrução acima se encaixa nas instruções que são capturadas pelo parser, visto que não possui o modificador "label_dont_use_X", onde X representa um numeral.

Após o processamento de todo arquivo gerado pelo gdb o arquivo resultante possui apenas as instruções úteis. Abaixo é dado um exemplo.

```
adds
str
ldr
```

Figura 5-10 Código filtrado

É possível visualizar no código acima que apenas as instruções que não possuem o modificador "label_dont_use_X" são utilizadas. Na *seção 5.2.5* é exibido o fluxo do parser utilizado e a metodologia utilizada para computar os dados é abordada na *seção 5.3.1.*

5.2.4.5 Formatação

Nessa fase é realizada a formatação do arquivo gerado na fase de processamento descrito na *seção 5.2.3.4.* Esse módulo tem como finalidade contabilizar os seguintes dados :

1. Número de instruções executadas
2. Instruções que foram executadas
3. Potência individual das instruções
4. Potência total das instruções de um tipo
5. Potência de todo o módulo

Um exemplo de saída pode ser descrito na tabela abaixo e sua caracterização é feita logo em seguida.

Tabela 5-2 SAIDA PROCESSADA

Instruction	Potency(Ws)	Number Of Instructions	Total of potency(Ws)
add	6,20E-02	4017	2,49E+02
and	6,13E-02	10	6,13E-01
b	1,69E-01	4	6,75E-01
bcs	1,69E-01	1	1,69E-01
beq	1,69E-01	1	1,69E-01
bhi	1,69E-01	2002	3,38E+02
bl	1,67E-01	3	5,01E-01
bls	1,67E-01	2	3,34E-01
bne	1,69E-01	2	3,37E-01
cmp	6,18E-02	2008	1,24E+02
ldmia	9,69E-02	3	2,91E-01
ldr	7,06E-02	24067	1,70E+03
mov	6,01E-02	4038	2,43E+02
mvn	6,18E-02	6	3,71E-01
nop	5,84E-02	8	4,67E-01
orr	5,84E-02	2005	1,17E+02
rsb	5,84E-02	8	4,67E-01
stmdb	8,78E-02	3	2,63E-01
str	6,52E-02	12034	7,84E+02
sub	6,18E-02	2008	1,24E+02
Total potency : 3.683325E-03 Watts			
Number of instructions : 52230			
Can't find : 0 instructions			

- Na primeira coluna é explicitado quais instruções foram executadas
- A segunda coluna informa a potência média de cada instrução
- A terceira coluna reflete o número de vezes que determinada instrução foi executada
- A quarta coluna fornece a potência total para cada instrução, para isso usamos a seguinte fórmula :

$$Pt = \sum_{1}^{n} pi$$

Equação 5-1 Potência total para determinada instrução

Ou

$$Pt = n * pi$$

Equação 5-2 Potência total para determinada instrução

  - n representa o número de vezes que determinada instrução foi executada
  - pi é a potência individual para determinada instrução

Na linha 23 da **tabela 5.2** é explicitado a potência total do módulo, para isso usamos a fórmula descrita abaixo.

$$PT = \sum Pt \ , onde\ Pt\ é\ potência\ total\ encontrada\ para\ cada\ instrução$$

Equação 5-3 Potência total para determinado módulo

Ou

$$PT = \sum \sum_{i=0}^{n} pi\ , onde\ pi\ é\ a\ potência\ de\ cada\ instrução$$

Equação 5-4 Potência total para determinado módulo

A partir das fórmulas citadas podemos computar a potência que determinado módulo consome, na versão 1.0 não damos suporte a energia consumida, visto que o tempo para executar determinada instrução não foi calculada.

5.2.5 **Fluxo Ambiente Real**

Após o fluxo do ambiente virtual ter sido exibido e descrito podemos abordar o fluxo necessário para utilizar o dispositivo em ambiente real. Na **figura 5.11** é fornecido uma visão de alto-nível desta fase .

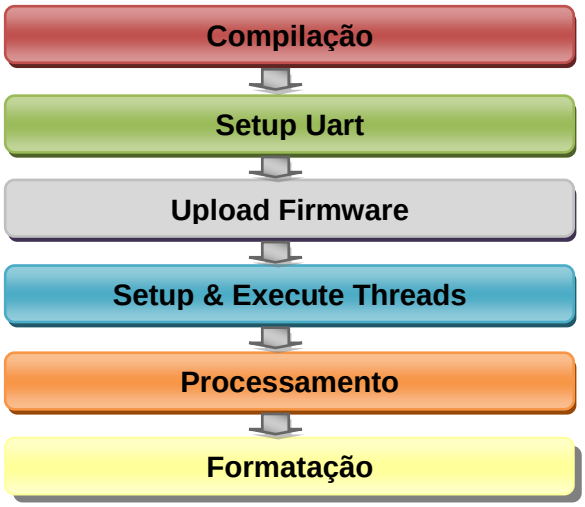

Figura 5-11 Fluxo execução ambiente real

Cada parte do módulo exibido acima possui peculiaridades que serão abordadas nas próximas seções .

5.2.5.1 Compilação

Esse módulo possui extrema semelhança com o descrito na *seção 5.2.3.1* . Nessa fase ocorre a compilação do script responsável pela tradução do código fonte para código ELF e binário . A criação desses script's para o ambiente real pode ser encontrado na seção *5.2.2.2* deste trabalho

A execução do script é feito através da função do sistema operacional - system. Por questões de padronização a ferramenta, *power analisys tool*, fará uso da função execl em suas futuras versões.

A chamada é realizada da seguinte forma :

```
sprintf(name_script_embedded,"../output/%s/./script_embedded",module);
system(name_script_embedded);
```

Figura 5-12 Chamada para compilação em ambiente real

Onde "module" apresentado na **figura 5.12** corresponde ao nome do módulo desejado. Comparando com o módulo de compilação apresentado na seção 5.2.3.1 podemos visualizar que os *script's* possuem nomes diferentes, isso é realizado para criar

uma estrutura mais organizada para o projeto e a compilação do módulo para ambiente real necessita da tradução do formato ELF para o formato binário.

A estrutura de diretórios será descrita no capitulo *7 seção 3* .

5.2.5.2 Setup Uart

Fase a qual tem como responsabilidade de iniciar e configurar a porta serial do PC para que seja possível a sincronia entre o sistema embarcado e o PC . Essa fase é abordada nas *seções 5.2.4.4* .

5.2.5.3 Upload Firmware

Fase responsável por invocar aplicativo para fazer o download do arquivo desejado em formato binário para o sistema embarcado. Na *figura 5.13* podemos visualizar o fluxo desta fase.

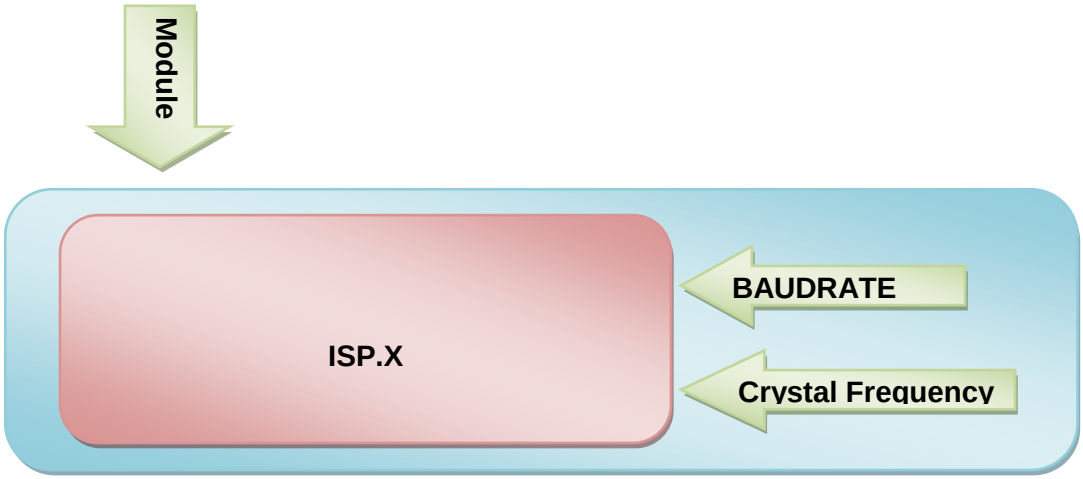

Figura 5-13 Diagrama módulo gerador script

Através da imagem acima é possível visualizar três entradas, onde duas são transparentes para o usuário. A única entrada externa para esse módulo é o nome do programa que se deseja testar.

Os parâmetros BAUDRATE e *Crystal Frequency* são valores estáticos utilizados para configurar o hardware disponível para esse trabalho. Essas entradas são responsáveis por configurar o programa responsável por fazer o download do módulo em formato binário para a placa MCB2103.

5.2.5.4 Setup & Execute Threads

Essa parte é responsável pela criação e sincronia das threads por parte do PC . A *seção 5.3.5.1* aborda como é realizado a sincronia e computação dos dados. Na **figura 5.14** podemos visualizar o fluxo executado por esse módulo.

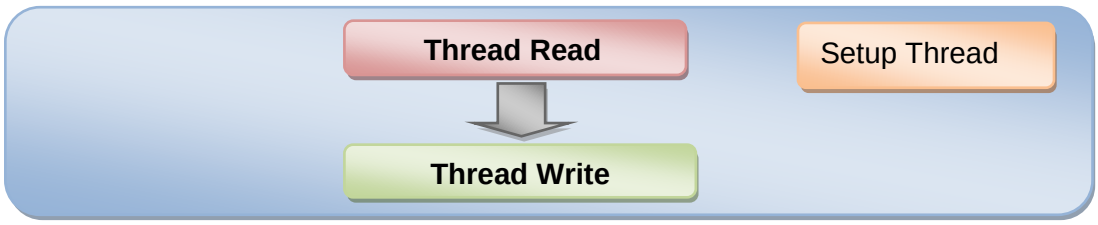

Figura 5-14 Fluxo de sincronia entre threads

O módulo setup é responsável pela criação das threads e configuração da uart. O módulo thread read está relacionada com as leituras a partir da porta serial, enquanto o módulo thread write é responsável pela escrita de dados na porta serial. Para fornecer uma explicação mais detalhada de cada módulo acima descrito os módulos são abordados nas próximas duas seções.

5.2.5.4.1 Thread Read

Thread que tem como objetivo gerenciar a entrada dos dados a partir da serial .O fluxo dessa thread pode ser sumarizado na **figura 5.15.**

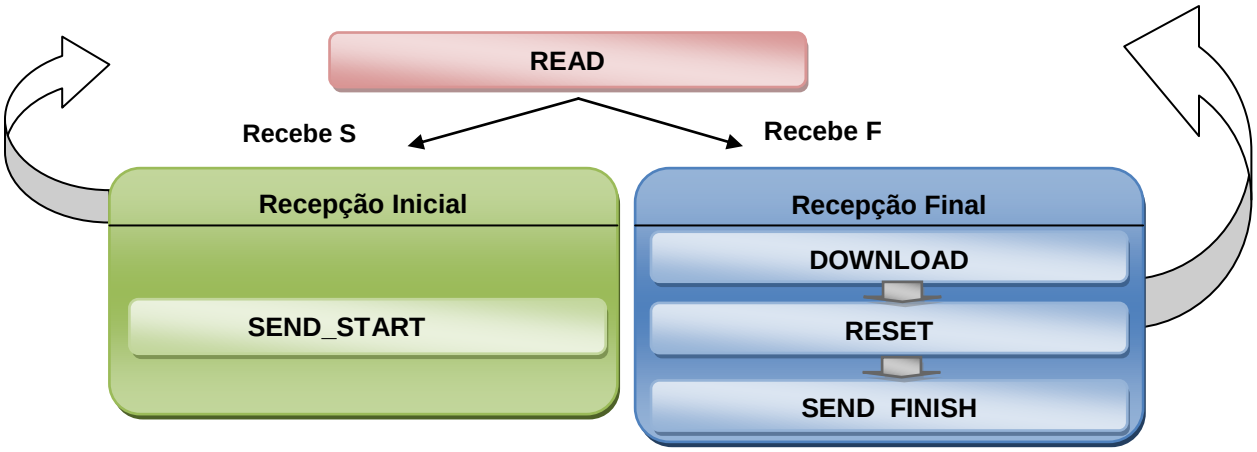

Figura 5-15 Fluxo Thread Leitura

A thread read é responsável pela leitura da porta serial e interpretação dos dados que são enviados pelo sistemas embarcado. Logo, essa thread tem como função controlar o sincronismo entre pc, osciloscópio e sistema embarcado. Devido a importância dessa thread abaixo explicamos o processo de sincronia por parte do PC, na *seção 5.3.5* é apresentado o processo de sincronia completo.

5.2.5.4.1.1 Recepção Inicial

Quando é enviado para o PC o caractere “S” significa que o sistema embarcado encontra-se no macro estado início descrito na *seção 5.3.5.1.1* e que o PC enviou um caractere, o qual não corresponde ao requisitado pelo sistemas embarcado. O caractere “S” serve para alertar ao PC que o sistema embarcado está esperando esse caractere para iniciar a execução. Na **figura 5.16** é possível visualizar essa situação.

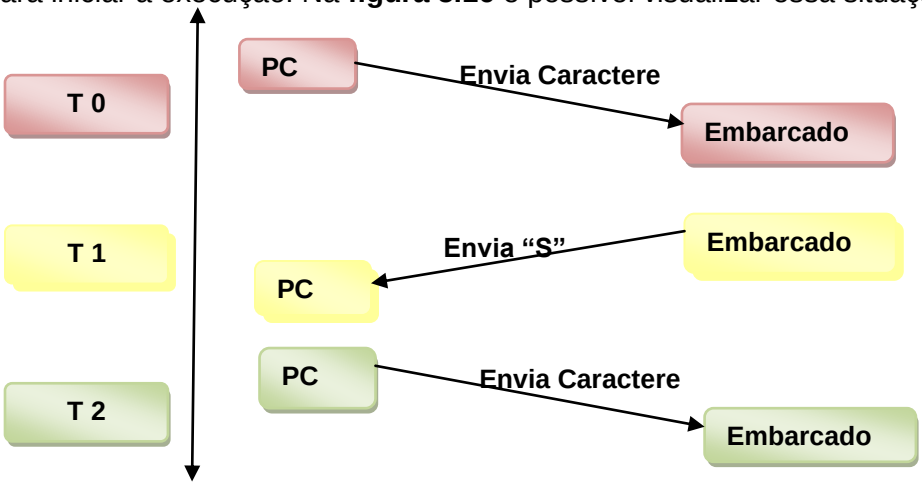

Figura 5-16 Thread Read - Sincronismo– Subfluxo para iniciar execução

Através da figura 5.16 visualizamos que em T0 o PC envia um caractere diferente de “S”. Logo, em T1 o embarcado sinaliza para o PC que ele está esperando “S”, porém o PC enviou outro caractere. Assim, que a Thread Read recebe o caractere desejado é feito mudanças na variável “flag_control_switch”, a qual sinaliza o caractere que deve ser enviado. O estado dessa variável é alterado para “SEND_START” que representa o caractere “S”.

5.2.5.4.1.2 Recepção Final

Esse mecanismo está relacionado com a captura e reconfiguração do osciloscópio. Abaixo exibimos o fluxo para sincronia desse subfluxo da thread read.

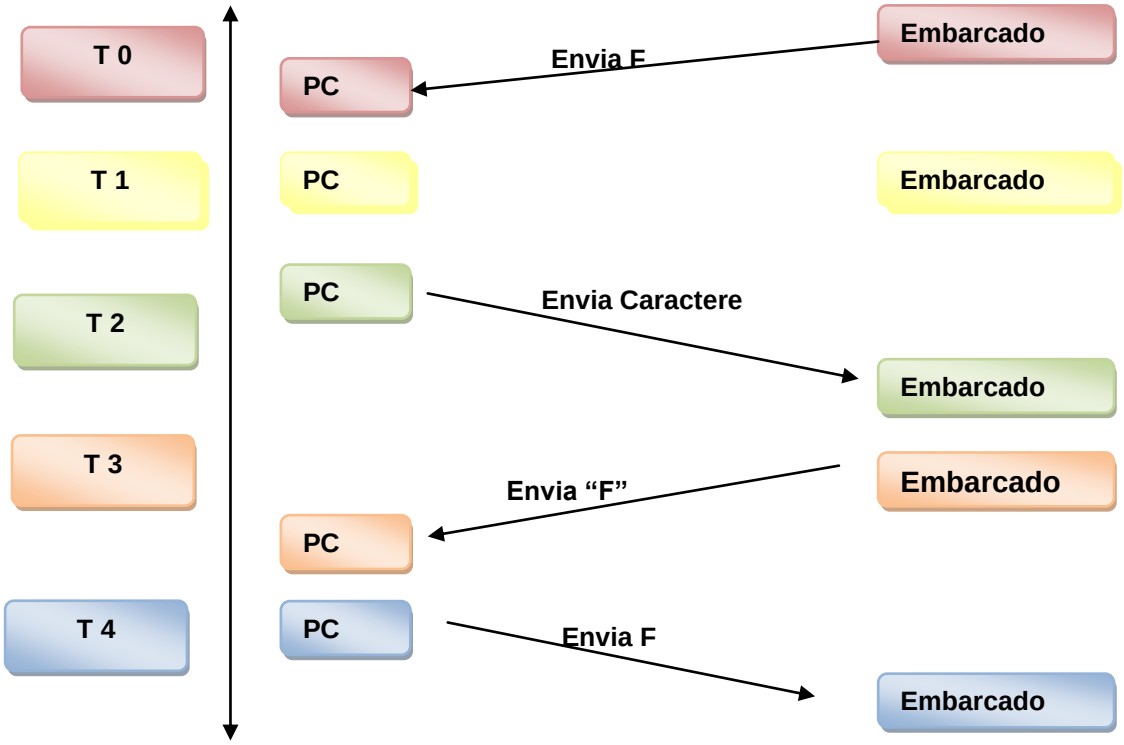

Figura 5-17 Fluxo para sincronia final

- Em T0 o sistema embarcado inicia a comunicação com o PC a fim de sinalizar o final da execução do módulo.
- Em T1 o PC recebeu o caractere “F” e realiza o download das informações capturadas pelo osciloscópio e reconfigura-o para captura de uma nova amostra.
- A fim de ilustrar possíveis problemas de comunicação. Em T2 o embarcado encontra-se no macro estado nomeado como término, o qual é explicado na *seção 5.3.5.1.3.* Nesse estado o embarcado espera o caractere “F”, porém recebe um caractere diferente.
- Em T3 é feito a verificação da validade do caractere. No entanto, o sistema embarcado interpreta que o valor recebido não corresponde ao requisitado, logo o caractere desejado é enviado novamente para o PC.
- Em T4 o PC envia para o dispositivo o caractere desejado sinalizando que a amostra do osciloscópio já foi capturada e o osciloscópio foi reconfigurado para nova amostra.

5.2.5.4.2 Thread Write

Essa thread tem como finalidade mandar para o embarcado sinais de sincronismo ela é controlado pela thread de leitura. A figura 5.18 fornece uma visão do funcionamento da thread write.

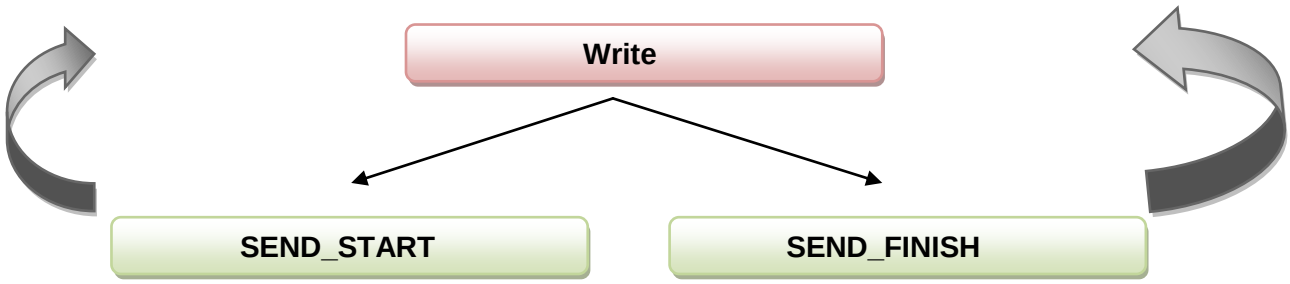

Figura 5-18 Diagrama da Thread Write

. O método de sincronia entre essas duas threads será abordado na seção *5.3.3.*

5.2.5.5 Processamento

Após ter explicado o mecanismo de sincronia podemos passar para o módulo de processamento responsável por tratar a saída do osciloscópio. Esse módulo executa a chamada de um parser que realiza a transformação de dados que estão no formato hexadecimal para decimal.

Na **figura 5-19** temos a descrição do fluxo utilizado para o processamento dos dados.

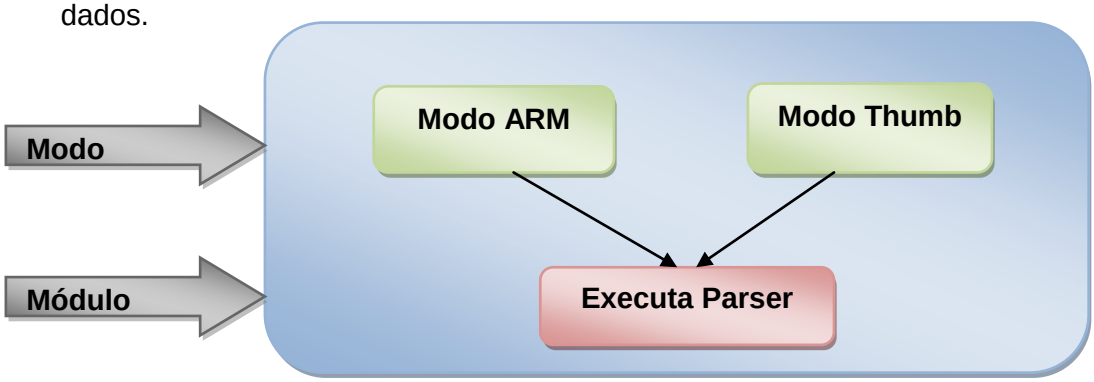

Figura 5-19 Diagrama Processamento

O módulo possui como entrada o Modo, o qual pode ser ARM ou Thumb e o nome do Módulo. Isso é apenas necessário para criar uma estrutura organizada dentro dos diretórios de forma que seja possível uma análise mais detalhada por parte do usuário da saída do osciloscópio. O fluxo do parser é descrito na seção 5.2.6 .

5.2.5.6 Formatação

Essa fase tem como finalidade tratar as informações geradas pelo parser anterior. Durante essa fase valores para potência, tempo e energia elétrica são calculado. A metodologia aplicada para filtrar os dados é descrita na seção 5.**3.2.** A figura abaixo fornece uma visão geral do fluxo executado.

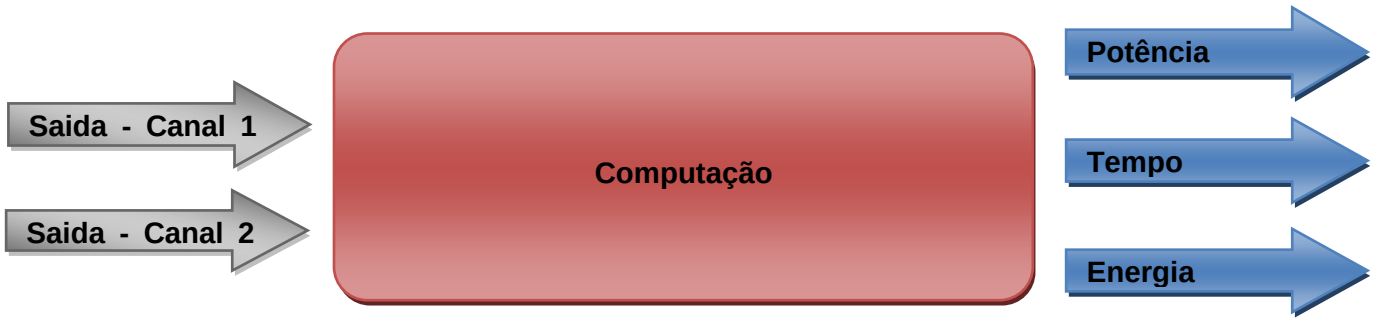

Figura 5-20 Formatação osciloscópio

O módulo recebe como entrada as saídas do osciloscópio processada pelo modo de processamento descrito na seção anterior. A saída gerada é estabelecida usando-se a metodologia descrita na seção 5.3.2. A partir da saídas o time de desenvolvimento pode fazer análises em tempo de projeto, que permite alterar ou ainda desistir do projeto em fase de desenvolvimento diminuindo os custos associados a correções.

5.2.6 **Parser skyeye**

Na figura abaixo sumarizamos o fluxo do *parser* utilizado pelo skyeye para computação dos dados. Na *seção 5.2.3.4* explicamos o que o módulo de processamento realiza. Nesta seção vamos mostrar o fluxo do *parser* criado para tratar a saída. O método utilizado para computar os dados da saída do *parser* é explicado na *seção 5.3.2 .*

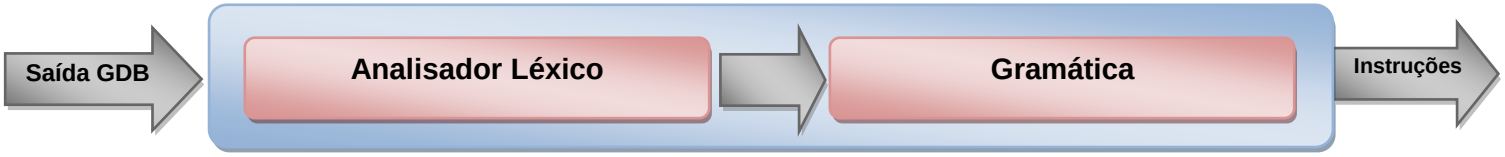

Figura 5-21 Parser Skyeye

A saída do gdb possui informações que não desejamos, a criação desse *parser* se faz necessário por esse motivo. A ideia é filtrar a saída do gdb a fim de criar um arquivo com instruções isoladas.

O analisador léxico tem como função encontrar os padrões desejados, dessa forma podemos identificar os *tokens* que desejamos.

A gramática é bem simples ela possui apenas a seguinte regra :

HEAD TAB INST

Quando a regra dessa gramática estiver satisfeita filtramos novamente e utilizamos apenas as instruções que são importantes. Dessa forma podemos varrer todas as instruções executadas. Esse mecanismo é interessante, pois podemos filtrar a saída do gdb de uma maneira mais rápida, descartando informações não desejadas, ou seja descartando instruções que não fazem parte da execução do módulo.

5.2.7 **Parser osciloscópio**

A fim de facilitar o processamento da saída do osciloscópio utilizamos o mesmo processo descrito para o *parser* realizado para saída do gdb. Apesar do fluxo ser o mesmo, o alfabeto e a gramática usadas neste *parser* são completamente diferente.

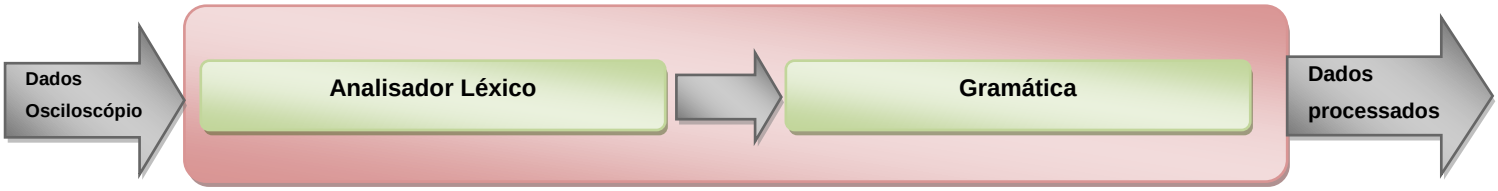

Figura 5-22 Fluxo Parser Osciloscópio

Os dados do osciloscópio são informações que necessitam ser transformadas para dados reais, logo o *parser* criado tem essa finalidade.

O analisador léxico tem como função encontrar os *tokens* desejados e a gramática tem como função encontrar sequências que a satisfazem, dessa forma podemos realizar a conversão dos dados desejados. Abaixo é fornecido um trecho da saída do osciloscópio.

```
YORI:8.00e-04
YINC:8.00e-04
XORI:2.700000e-01
XINC:1.000000e-04
0x7D 0x7D 0x7D 0x7D ...
```

Os *tokens* identificados pelo analisador léxico são :

- YORI – Representa a posição do cursor para coordenada Y
- YINC – Representa a diferença entre duas amostras
- XORI – Representa a posição do cursor para coordenada X

Os valores hexadecimais são reconhecidos, mas são convertidos em regras específicas da gramática. A gramática utilizada faz uso de basicamente cinco regras são elas :

- YORI COLON DOUBLE_VALUE
  - Construção para origem em Y .
- YINC COLON DOUBLE_VALUE
  - Construção para incremento em Y .
- XORI COLON DOUBLE_VALUE
  - Construção para origem em X .
- XINC COLON DOUBLE_VALUE
  - Construção para incremento em X .
- HEX_NUMBER
  - Construção responsável por realizar a conversão de número hexadecimais para decimais .

5.3 **Metodologia**

Essa seção apresenta as metodologias aplicadas para o projeto. Dividimos a apresentação dos métodos de forma a fornecer uma visão mais clara de cada parte do projeto.

5.3.1 **Método computação de dados usando simulador**

Na **figura 5.23** abaixo descrevemos como a computação dos dados são realizadas quando estamos trabalhando com skyeye.

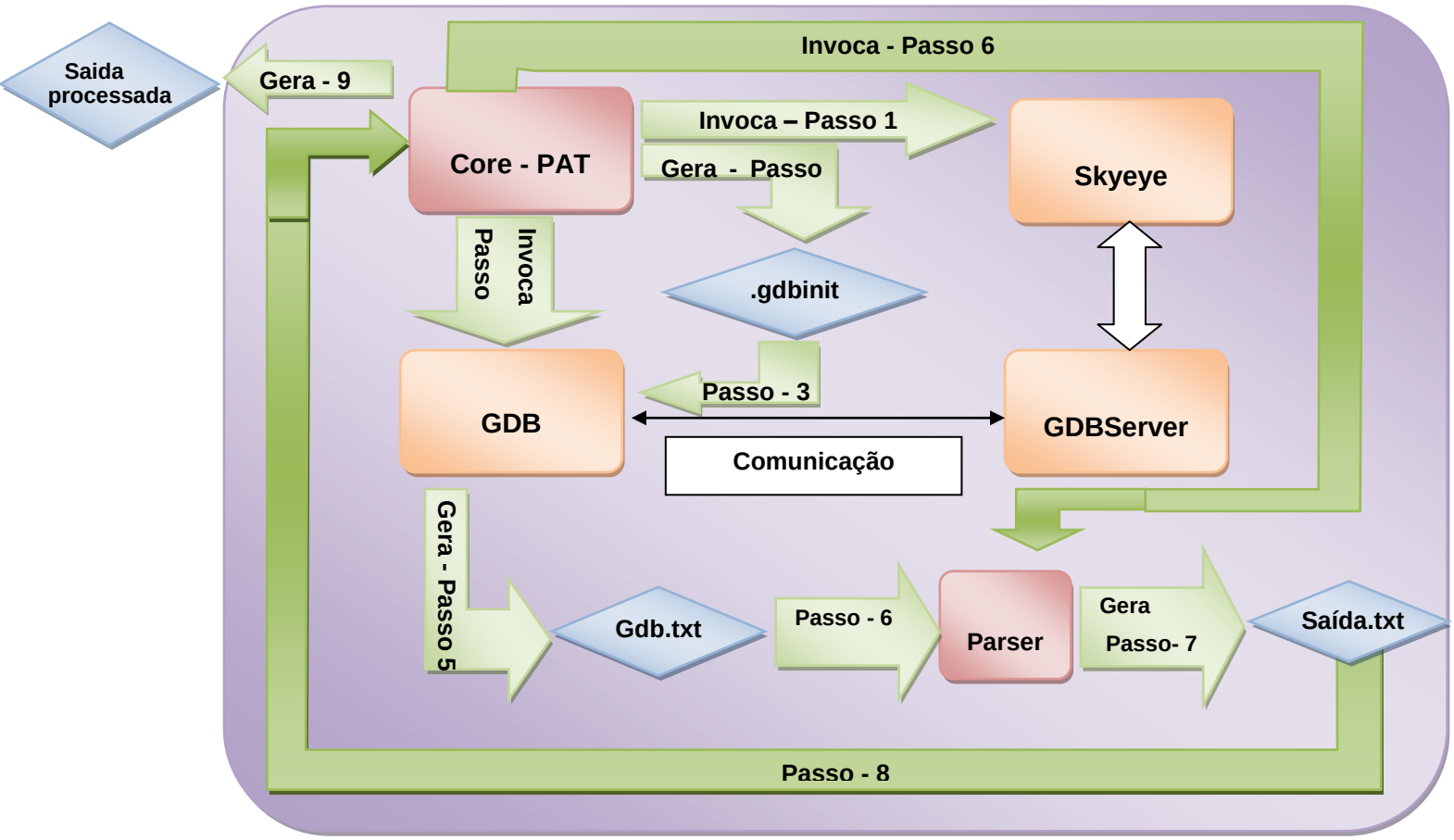

Figura 5-23 Fluxo Geral Simulação virtual

O fluxo realizado por , Core – PAT, relacionado ao ambiente virtual é descrito nas *seções 5.2.3 e 5.2.4* . Para computar os dados relacionados a potência em ambiente virtual o core-pat funciona como um controlador dizendo em sequência o que deve ser feito. Os passos 1, 2, 3 e 4 são responsáveis pela preparação do executável para execução em ambiente virtual. O passo 5 está relacionado com a criação de um arquivo texto que contem todas as instruções executadas, a partir do arquivo gerado é necessário realizar filtragem para eliminar as informações não desejadas.

O *parser* atua como filtro, isolando as instruções executadas, durante esse processo apenas a instrução é capturada e os seus possíveis parâmetros são descartados. Ou seja, esse projeto não considera a energia gasta pelo problema relacionado ao mecanismo de chaveamento entre instruções e variações dos parâmetros.

Após todo o arquivo tiver sido filtrado resultando no passo 7, podemos calcular a potência consumida. Para que isso seja possível uma tabela contendo as médias relacionadas a potência foram computadas, considerando o mecanismo da potência relacionada ao chaveamento de instruções e variação dos parâmetros. Porém, apenas a potência média de cada instrução foi utilizada.

Cada instrução é pesquisada na tabela e somamos esse valor base a potência já computada. Com isso, podemos identificar a quantidade de vezes que determinada instrução foi executada e a potência consumida pelo conjunto.

Da forma descrita acima podemos computar a potência consumida por um programa executando em ambiente virtual, fazendo assim, que não seja necessário o uso de um hardware especial.

Apesar da tabela ter sido criada considerando o problema relacionado ao chaveamento de instruções e variando os parâmetros é necessário reconstruir essas tabelas contendo essas informações organizadas de uma forma atômica considerando os parâmetros e instruções consecutivas. Somente assim poderemos ter um valor mais detalhado para computação da potência.

5.3.2 **Computação de dados usando osciloscópio e conversão dos dados**

Nesta seção abordaremos o método usado para computar e converter os dados oriundos do osciloscópio. A **figura 5.24** fornece uma visão clara dos passos realizados para computação dos dados.

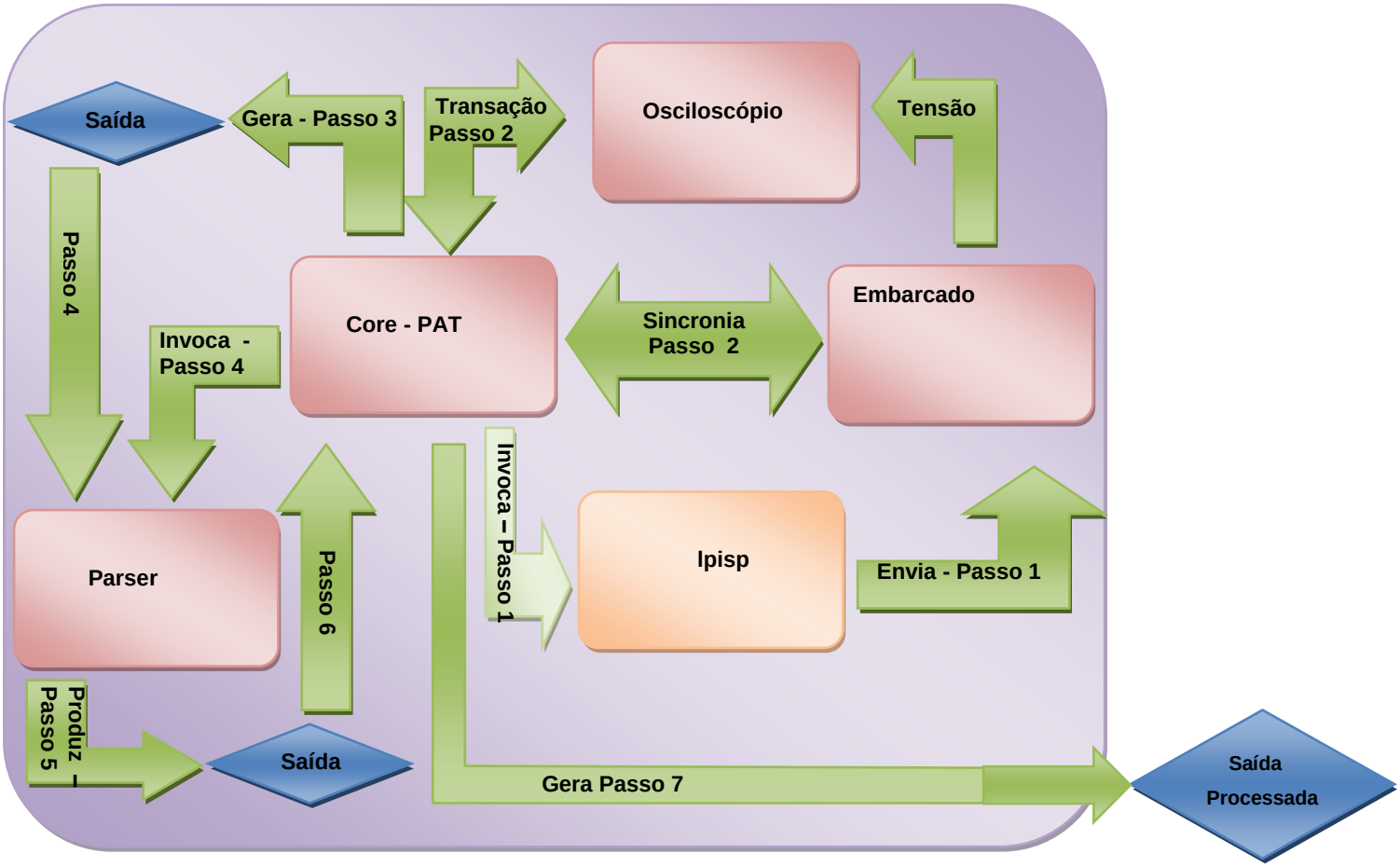

Figura 5-24 Simulação em ambiente real

Para capturar os dados obtidos pelo osciloscópio utilizamos a porta USB para comunicação, para que isso fosse possível utilizamos a biblioteca *libusb,* a qual fornece um modo mais simples para comunicação através da porta USB.

Todas as mensagens enviadas são mensagens de controle, as *strings* enviadas e recebidas são processadas de acordo com o manual do programador **[1]**.

O funcionamento do core-pat relacionado ao ambiente real é descrito nas *seções 5.2.3 e 5.2.4* . O passo 1 está relacionado ao download do executável para o ambiente real. O passo 2 está relacionado a sincronia necessária para capturar os dados gerados pela execução do programa e é descrito nas *seções 5.2.4.4.1 e 5.3.5.1.* Para capturar os dados do osciloscópio relacionados ao passo 2 são usados dois canais do osciloscópio,

fazendo uso desses canais é possível isolar a amostra de dados.O canal 1 é utilizado para capturar a tensão sobre o resistor de shunt, enquanto o canal 2 é responsável por capturar um sinal gerado pelo sistema embarcado. Esse sinal é gerado quando vamos executar a função desejada e desligado assim que a execução da função terminar. Esse sinal é produzido através da Porta 0 e pino 4 do arm utilizando o LPC2103 e a placa da keil MCB2103.

A saída produzida no passo 3 está relacionada a informações dos canais um e dois. A saída produzida no passo 5 está relacionada a transformações necessárias dos dados aferidos do osciloscópio, visto que o osciloscópio retorna valores hexadecimais. A conversão é feita através da seguinte fórmula :

$$(\mathbf{125} - \boldsymbol{VALOR}) * \boldsymbol{YINC} - \boldsymbol{YOR}$$

Equação 5-5 Conversão de dados

- YORI – Representa a posição do cursor para coordenada Y
- YINC – Representa a diferença entre duas amostras em Y
- VALOR – Representa o valor capturado a partir do osciloscópio convertido para base decimal .

Logo, os dados gerados pelo *parser* representam os valores capturados, porém, em valores decimais.

No passo 6, que representa a saída do *parser* no passo 5, são isolados as informações de forma a capturar apenas os dados que desejamos. Para que isso seja possível necessitamos das informações capturadas pelo canal 2. Como o canal 1 e canal 2 apresentam a mesma quantidade de dados gerados, capturamos os dados que representam um valor maior que 3.0 volts . Abaixo apresentamos um exemplo.

Tabela 5-3 Canal

Amostra	Canal 1	Canal 2
1	0.5E+00	0.1 E+00
2	0.55E+00	0.7 E+00
3	0.45E+00	3.3 E+00
4	0.70E+00	3.33 E+00
5	0.55E+00	3.2 E+00
6	0.45E+00	0.3 E+00
7	0.70E+00	0.4 E+00

Na tabela acima podemos ver que as amostras de número 3,4 e 5 relacionadas ao canal 2 possuem um valor maior ou igual a 3.1 volts, com isso, isolamos as amostras ja citadas e descartamos as restantes . O processo de ligar e desligar os pinos é descrito na seção *5.3.5.1.2*, a qual é relacionada a execução.

Após o isolamento das informações desejadas podemos computar o tempo de execução do programa, o calculo é feito através da **fórmula 5.6.**

$$Tempo\ total = Pontos * XINC, onde$$

Equação 5-6 Tempo total

- Pontos - representa o número total de pontos (amostras) capturados
- XINC representa o incremento no eixo X, ou seja no tempo.

Para calcular a potência consumida utilizamos a tensão aferida no resistor de *shunt*, com isso encontramos a corrente necessária para alimentar o core do lpc2103, utilizando a lei de ohm. Como o regulador fornece uma tensão fixa de 1.8 volts ao *core* podemos calcular a potência consumida utilizando a **fórmula 5.7.**

$$P = U * I$$

Equação 5-7 Potência em Watts

- U é a tensão em volts
- I é a corrente em ampére

Para calcular a energia utilizada utilizamos a fórmula abaixo.

$$E = P * T$$

Equação 5-8 Energia (W*S)

- P é a potência computada pela **equação 5-7** em watts**.**
- T é o tempo em segundos calcula pela **fórmula 5.6**

As fórmulas acima são utilizadas para todas as amostras. A potência média pode ser calculada usando a fórmula abaixo.

$$P_{avg} = \frac{1}{n} * \sum_{i} Pi\ , onde$$

Equação 5-9 Potência média

- n – É o número das amostras
- Pi – É a potência de determinada amostra.

Através das fórmulas citadas podemos obter os dados relacionados a potência, energia e tempo de execução de determinado programa.

5.3.3 **Método sincronia entre Threads**

Para realizar o mecanismo de sincronia entre as threads read e write descritas nas *seções 5.2.4.4*, as quais são responsáveis pela sincronia com o embarcado e o osciloscópio é feito a utilização de MUTEX.

Com esse mecanismo podemos fazer com que o PC e o embarcado se sincronizem. A utilização do *mutex* é necessária, pois dessa forma garantimos que a thread *write* enviará o caractere correto e que a ferramenta conseguirá capturar os dados do osciloscópio antes que o programa no sistema embarcado volte a executar.

5.3.4 **Metodologia circuito**

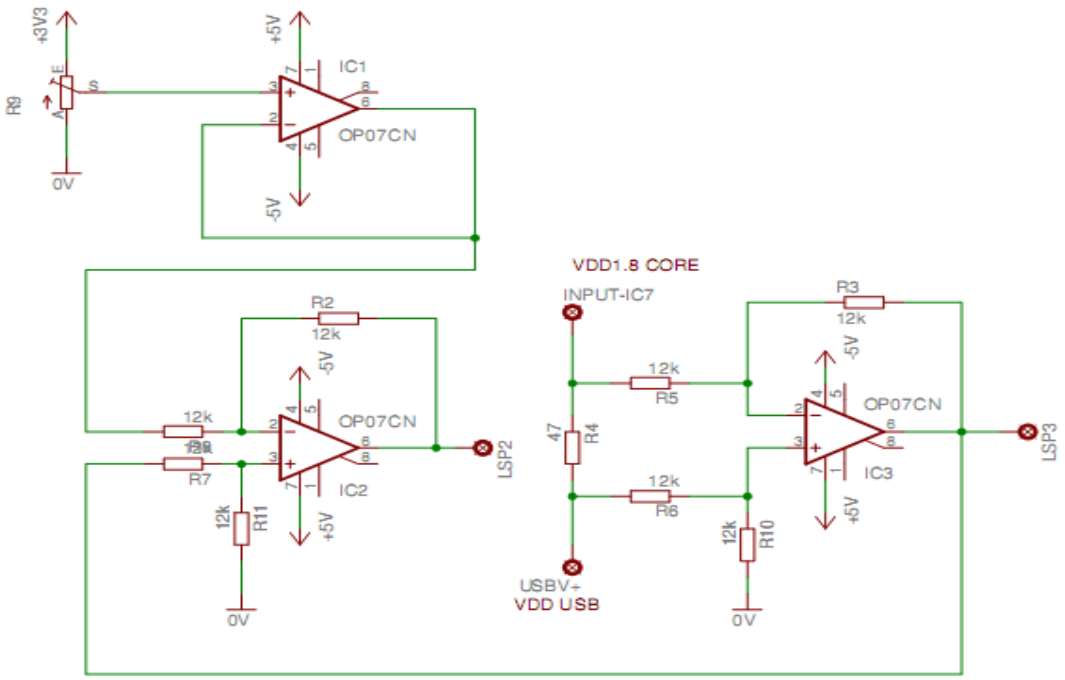

Figura 5-25 Circuito para computação dos dados

O circuito acima é responsável por capturar a corrente utilizada pelo *core* do processador ARM, para isso é necessário calcular a tensão sobre o resistor R4, o qual é resistor de *shunt*. Porém, é necessário retirar o nível necessário para sustentar o circuito interno do *core,* essa subtração é feita através do amplificador operacional que gera a saída LSP2. Com a tensão observada na saída, LSP2, podemos calcular a corrente sobre o resistor de *shunt,* que representa a corrente necessária pelo core. Com a corrente requisitada pelo *core*, podemos calcular a potência necessária que cada instrução necessita para ativação do circuitos internos, já que sabemos a tensão proporcionada pelo regulador ao *core*.

5.3.5 **Mecanismo para sincronismo entre PC Embarcado e Osciloscópio**

Nesta seção abordaremos como foi realizado todo o processo de sincronia entre o PC, sistema embarcado e osciloscópio.

5.3.5.1 Mecanismo de sincronia para sistema embarcado

O mecanismo de sincronia entre o PC e o embarcado é responsável por realizar todas as operações relacionadas a transações necessárias entre o osciloscópio e o PC. Tal mecanismo se torna necessário, pois a velocidade de captura dos dados do osciloscópio é inferior a geração dos das informações. Portanto, esse mecanismo se torna necessário para que os dados possam ser computados de maneira correta.

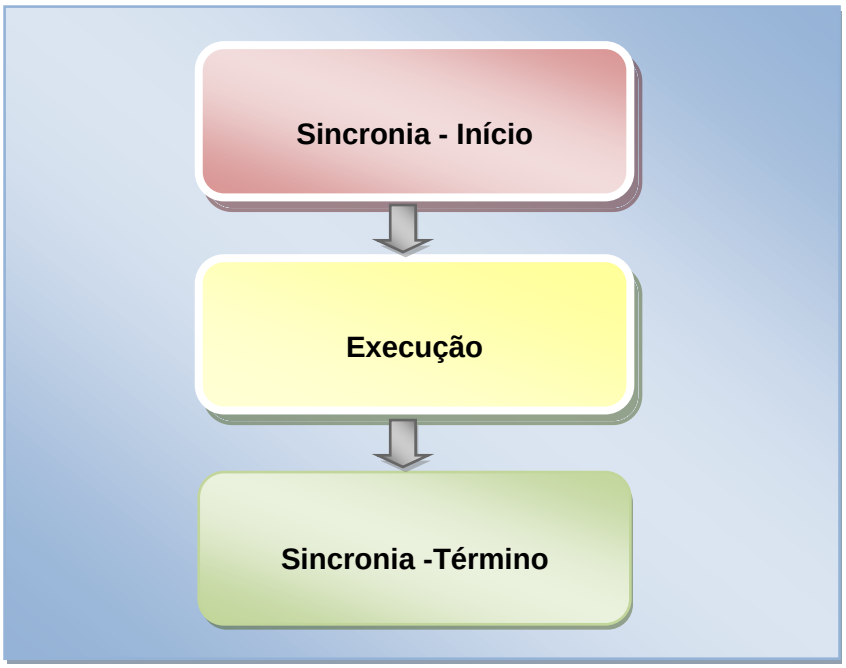

Figura 5-26 Fluxo embarcado

Nas próximas seções detalhamos os módulos exibidos na **figura 5.26,** fornecendo uma visão clara e objetiva de sua funcionalidade.

5.3.5.1.1 Sincronia – Início

Essa parte do programa tem como função iniciar a sincronia entre o sistema embarcado e o PC. O PC é escolhido como dispositivo *master* e a execução da função desejada só é realizada quando o dispositivo escravo recebe uma mensagem para liberação, essa mensagem é descrita com o caractere “S”. A figura abaixo exemplifica o mecanismo de sincronia inicial.

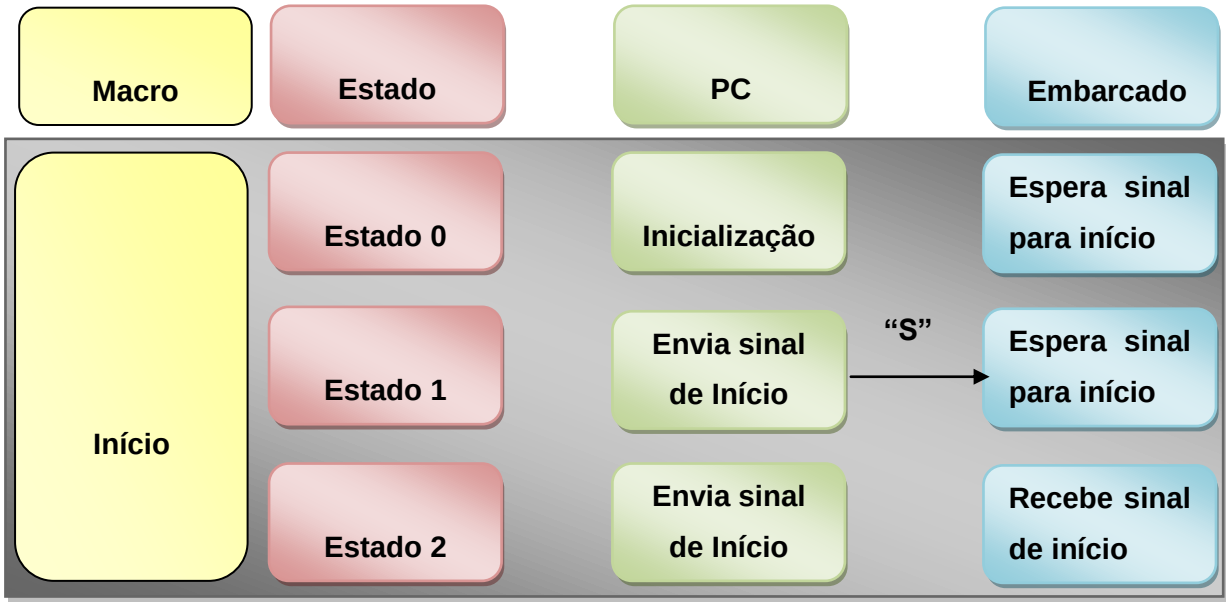

Figura 5-27 Fluxo comunicação

O macroestado tem como objetivo realizar a sincronia inicial entre o PC e o sistema embarcado. Dessa forma, o PC age como dispositivo *master* iniciando a comunicação com o sistema embarcado liberando-o para execução. Abaixo é feita a descrição de cada estado.

- Estado 0 – É a fase na qual o PC faz a inicialização do que é essencial para realizar sincronia.
- Estado 1 – Mecanismo que inicia a comunicação com sistema embarcado usando o caractere “S”.
- Estado 2 – O sistema embarcado recebeu o sinal de sincronia e pula para a macro estado de execução.

5.3.5.1.2 Execução

Abaixo é possível ter uma visão mais detalhada do macroestado utilizado.

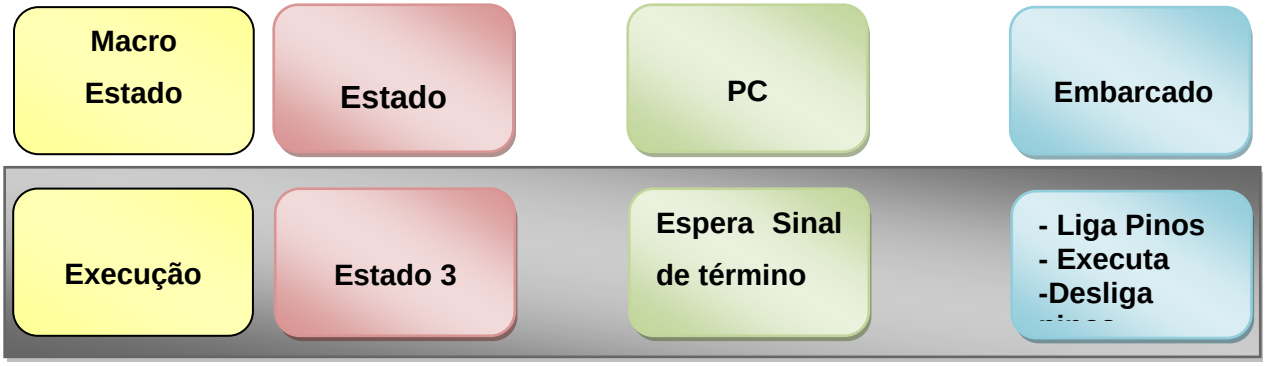

Figura 5-28 Processo de execução

- Estado 3 – Tem como função ligar e desligar o pino P0.4 para que seja possível filtrar os dados relacionados apenas a função solicitada.

5.3.5.1.3 Sincronia – Término

Fase na qual ocorre a sincronização final, para que os dados aferidos pelo osciloscópio possam ser transferidos para o PC, pois a velocidade de transferência é inferior a velocidade de geração de dados. Esse mecanismo se torna necessário para evitar perda de informações. Abaixo é possível ter uma visão mais detalhada do processo de sincronia.

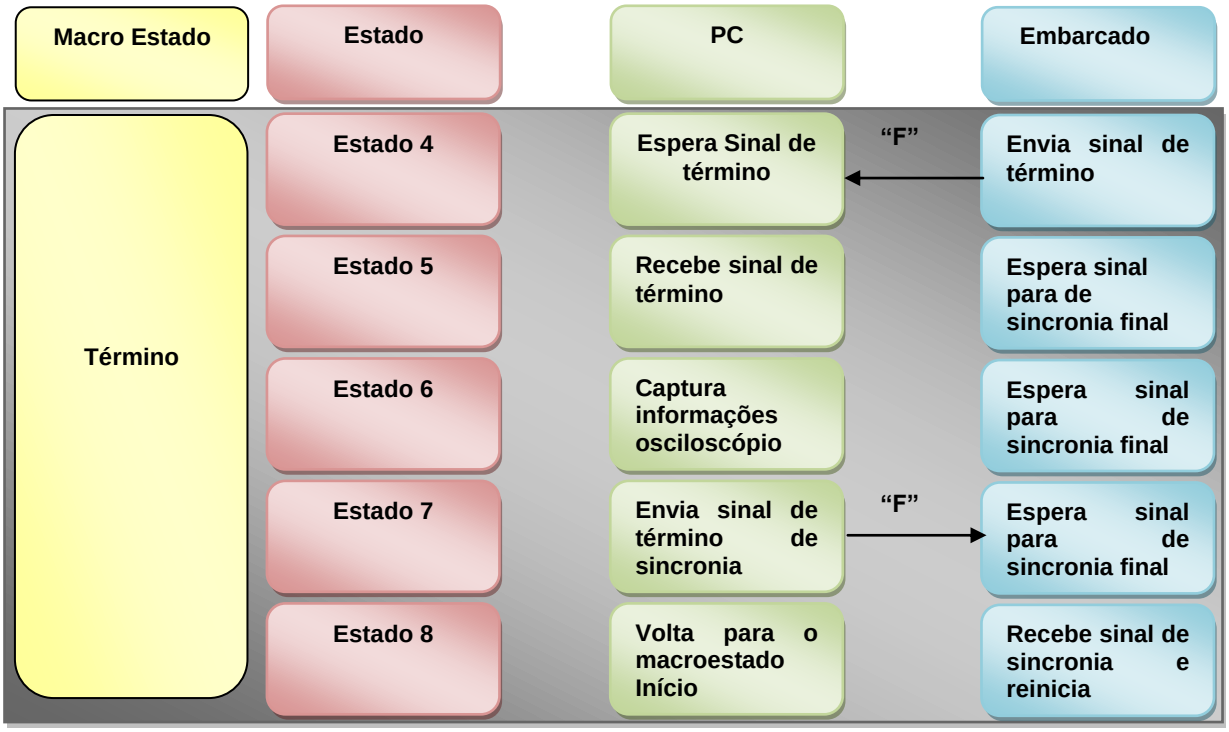

Figura 5-29 Processo de término

Essa fase representa a sincronia final entre o PC e o sistema embarcado, como dito antes é extremamente necessário para que não ocorra perda de dados. Abaixo é feito a descrição de cada estado.

- Estado 4 – O PC espera pelo término da execução do sistema embarcado. Assim, que ocorre o final da execução e o sistema embarcado entra no macroestado término é sinalizado para o computador o término da execução.
- Estado 5 – O PC recebe o sinal de término.
- Estado 6 – O PC captura informações do osciloscópio
- Estado 7 – PC sinaliza para informar que os dados já foram capturados e reconfigura o osciloscópio para capturar uma nova amostra
- Estado 8 - O PC e o embarcado voltam para o macroestado, início.

Abaixo é possível visualizar todos os macroestados e estados que o sistema embarcado e o PC executam quando desejam aferir informações.

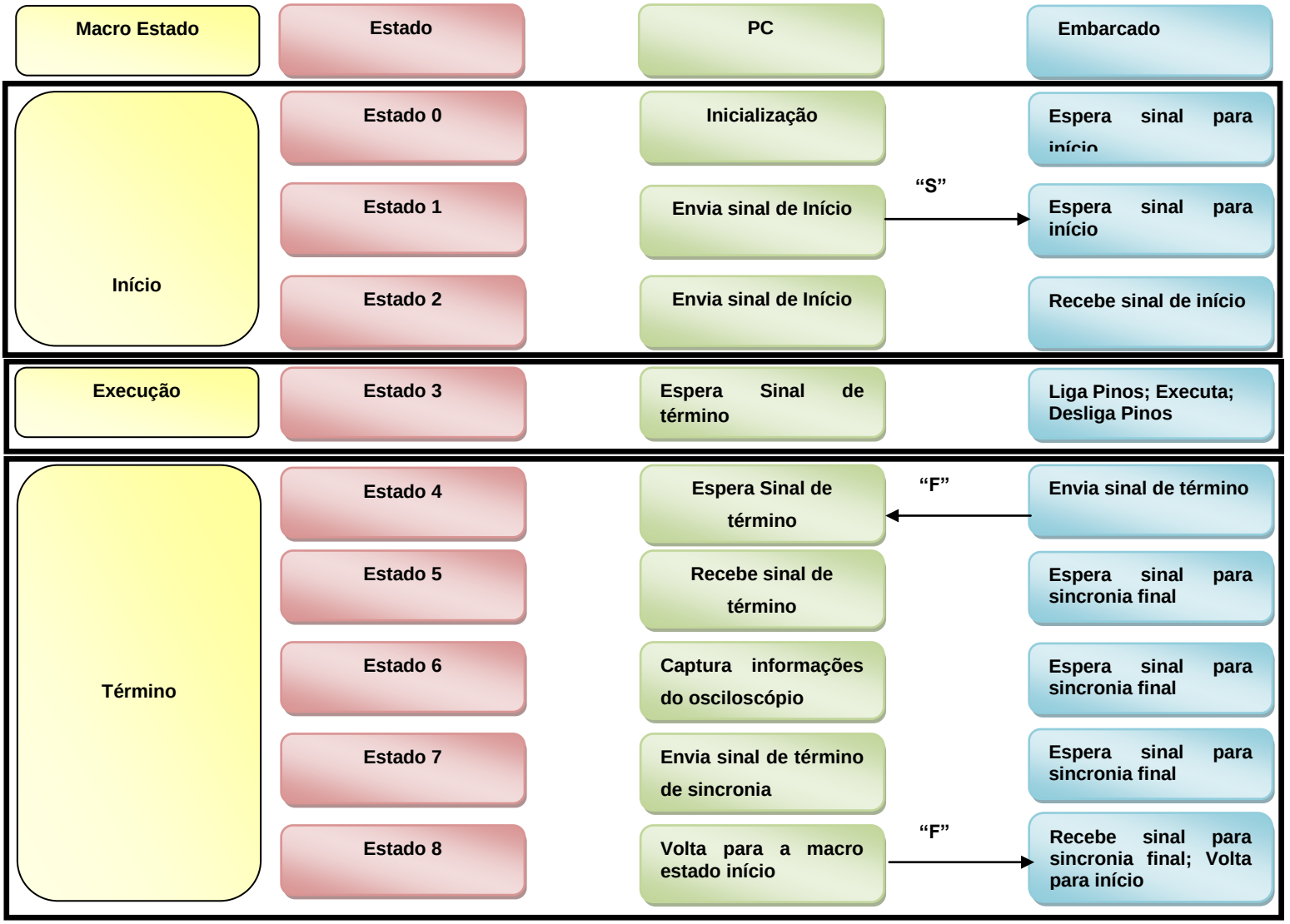

Figura 5-30 Mecanismo de sincronia entre PC e embarcado

5.3.6 **Mecanismo para sincronismo utilizado para sincronizar ambiente virtual**

Abaixo abordamos a sincronia necessária para computar dados oriundos da simulação virtual, a qual faz uso do simulador skyeye.

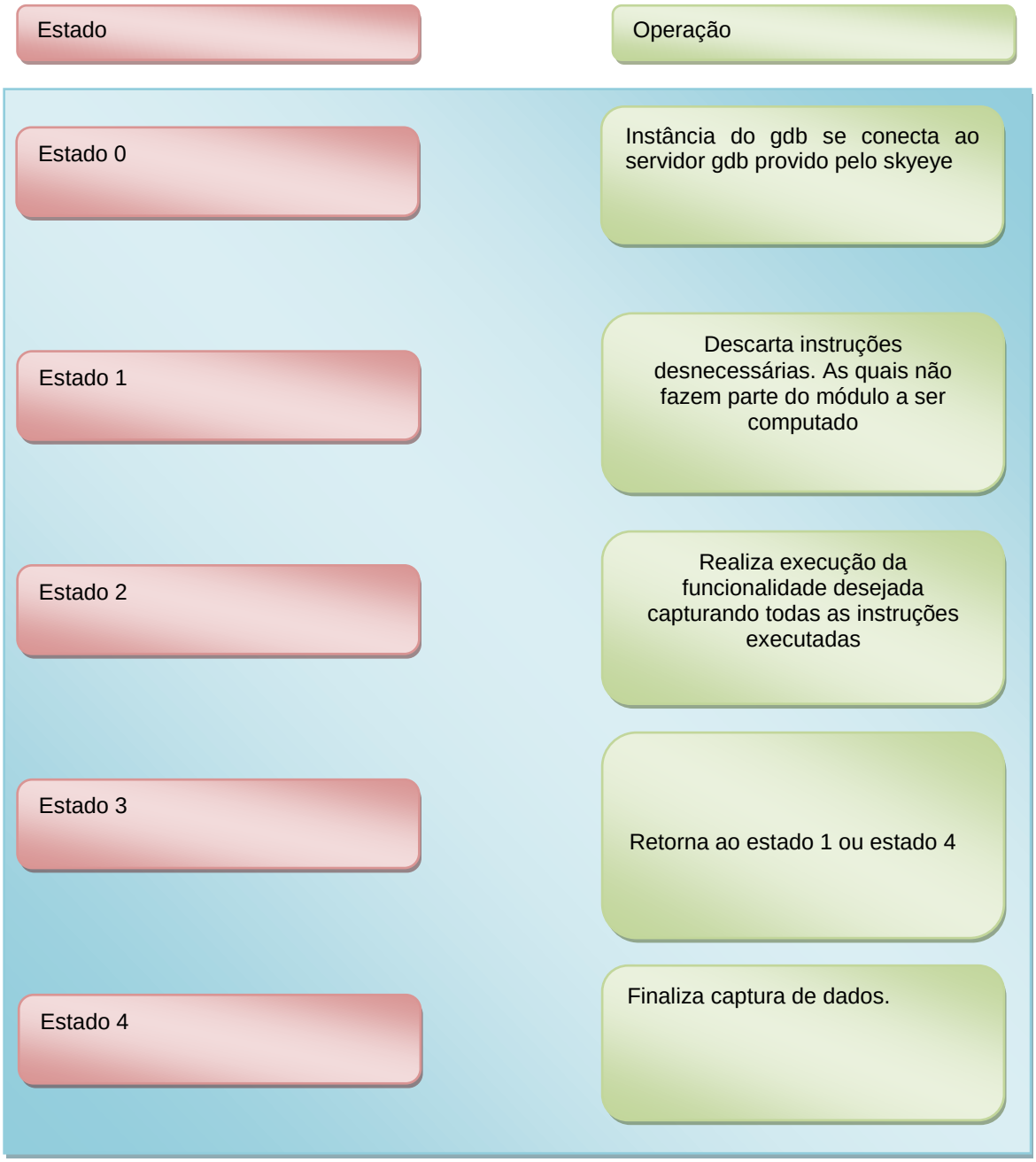

Figura 5-31 Mecanismo de sincronia entre skyeye e gdb

Esses estados são mais simples, do que o mecanismo elaborado para PC e o sistema embarcado, já que é possível em tempo de execução do simulador capturar os dados sem que eles sejam perdidos. o arquivo fonte gerado pela execução em ambiente virtual fornece 5 estados, os quais são descritos abaixo.

1. Estado 0 – Uma instância do gdb local se conecta através do stub do gdb ao servidor gdb proporcionado pelo skyeye.
2. Estado 1 – Assim como o sistema embarcado faz uso de pinos para isolar os dados desejados o sistema virtual faz uso de marcadores para isolar o código não desejado do desejado
3. Estado 2 – Nessa fase todas as instruções que são executadas são armazenadas em um arquivo para futuro processamento
4. Estado 3 – Nesta região é possível pular para o último estágio ou retornar para o estado 1, isso depende da quantidade de iterações solicitadas. Apesar de ser redundante executar o mesmo módulo novamente em ambiente virtual isso é feito para que futuramente seja possível alterar dados desejados em tempo de execução. Dessa forma poderemos explorar a variação de parâmetros por mecanismos externos .
5. Estado 4 – Estado no qual a conexão entre o *gdbserver* e o *gdbclient* é fechada

Para finalizar a seção de metodologia a **figura 5.32** apresenta o esquema geral da ferramenta, *power analisys tool*.

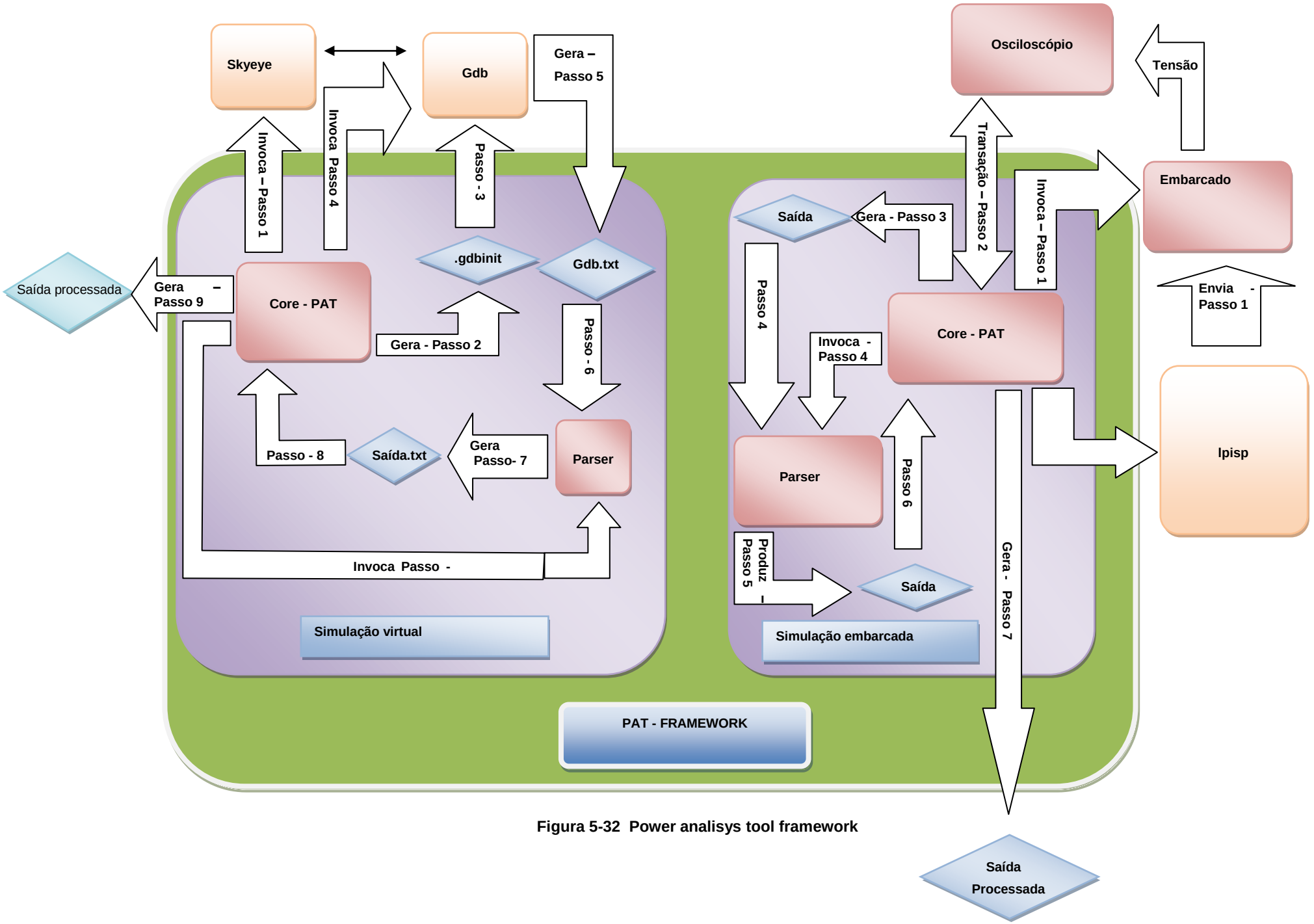

Figura 5-32 Power analisys tool framework

5.4 **Resultados**

Nesta seção vamos abordar os resultados obtidos a partir da ferramenta *power analisys tool* para benchmark powerstone.

Testamos cinco aplicações do benchmark, algumas modificações foram necessárias para adequar o código para a plataforma que possuíamos. Abaixo sumarizamos o conjunto de aplicações.

Tabela 5-4 Benchmark PowerStone

Aplicação	Finalidade
BLIT	Processamento de imagens
DES	Encriptação de dados
CRC	Checagem de dados
ADPCM	Codificador de voz
BCNT	Operações bit a bit em matriz

Fizemos dezenas de simulações a fim de encontrar um padrão no consumo de energia e caracterizar a diferença entre energia consumida no modo ARM e modo THUMB. As Tabelas abaixo sumariza os dados encontrados e os teste realizados.

Tabela 5-5 Método utilizado para resultados

Aplicações	Iterações	Modo	Execução
BLIT	2,4,8,16,32	ARM/THUMB	REAL/SIMULADO
DES	2,4,8,16,32	ARM/THUMB	REAL/SIMULADO
CRC	2,4,8,16,32	ARM/THUMB	REAL/SIMULADO
ADPCM	2,4,8,16,32	ARM/THUMB	REAL/SIMULADO
BCNT	2,4,8,16,32	ARM/THUMB	REAL/SIMULADO

Tabela 5-6 Saída dependente da execução

Aplicação	Execução real	Execução simulada
BLIT	Potência / Tempo /Energia	Número de instruções / Potência
DES	Potência / Tempo /Energia	Número de instruções / Potência
CRC	Potência / Tempo /Energia	Número de instruções / Potência
ADPCM	Potência / Tempo /Energia	Número de instruções / Potência
BCNT	Potência / Tempo /Energia	Número de instruções / Potência

5.4.1 **Execução em ambiente real**

Nesta seção apresentamos os resultados obtidos e geramos um modelo para caracterizar a diferença entre o modo thumb e modo real.

5.4.1.1 BLIT – Execução ambiente real

Abaixo fornecemos resultados obtidos a partir da execução da aplicação blit para execução em ambiente real.

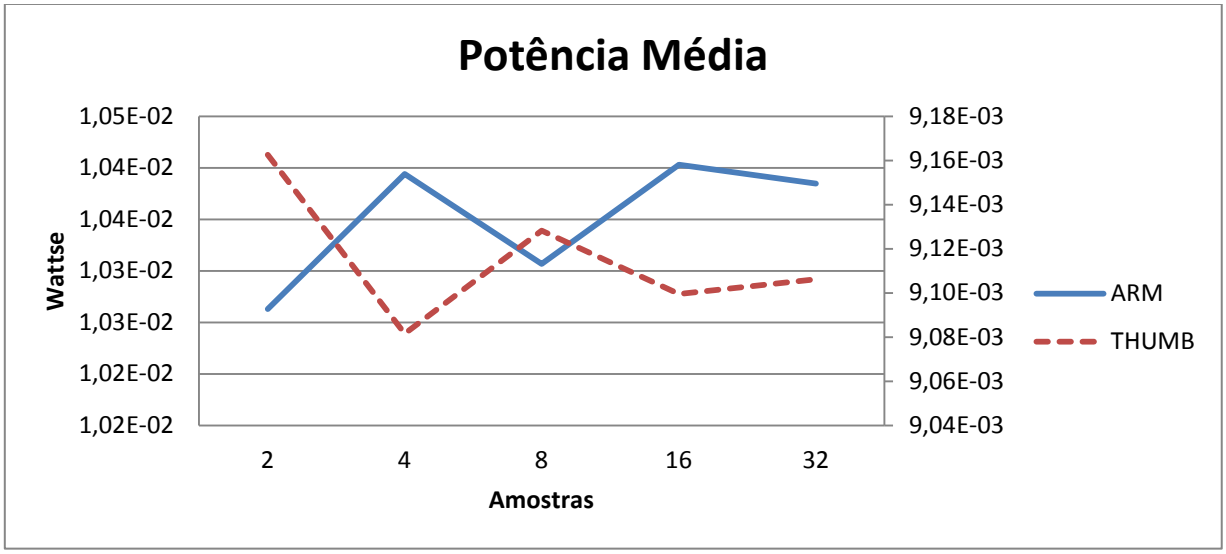

Figura 5-33 Potência Média

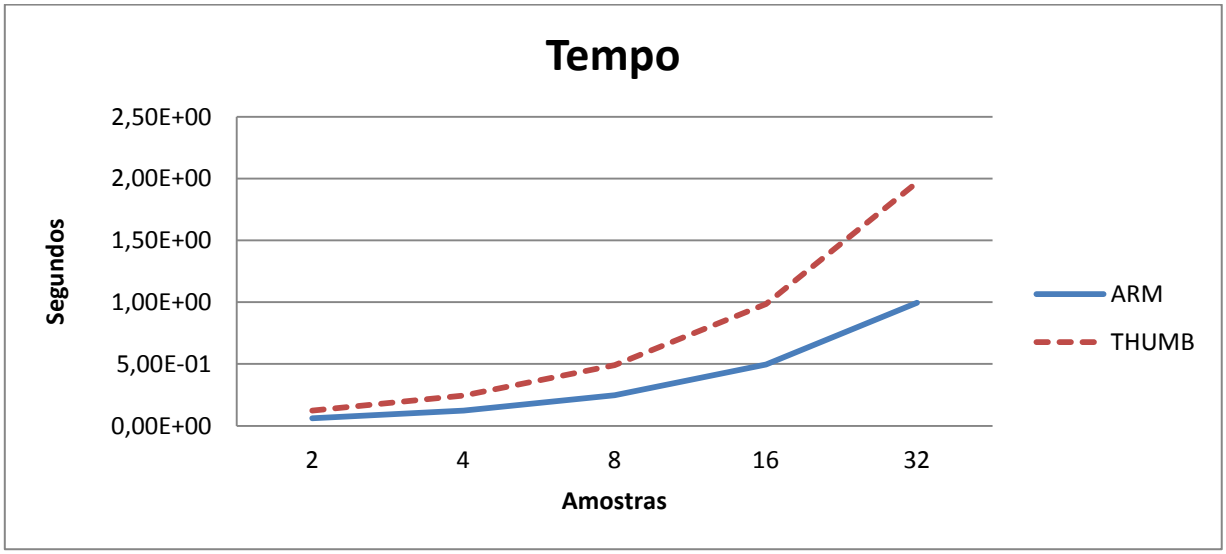

Figura 5-34 Tempo de execução

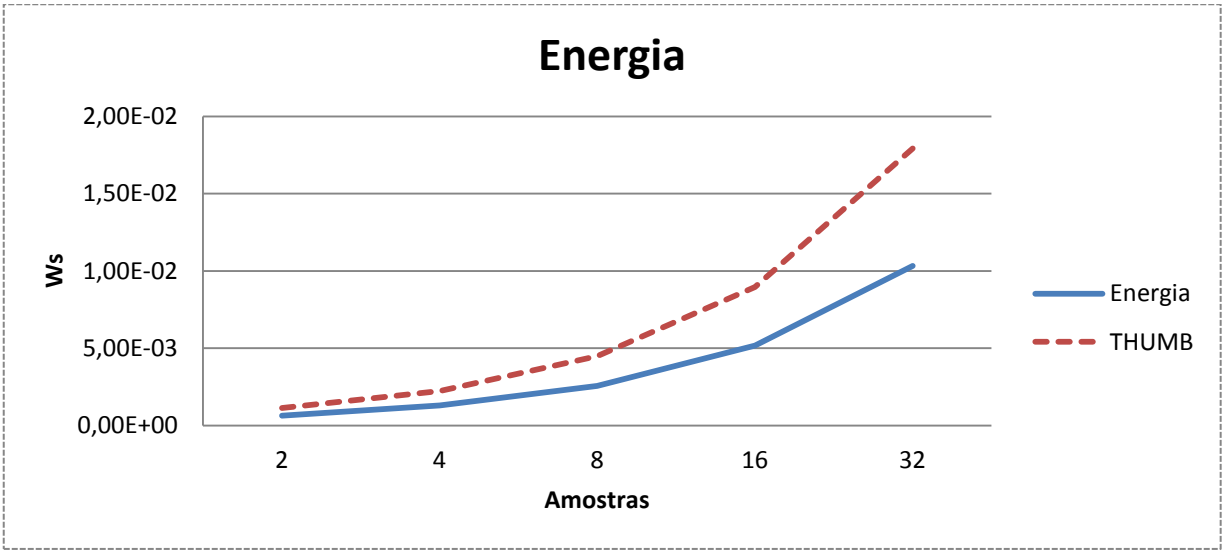

Figura 5-35 Energia consumida

5.4.1.2 ADPCM – Execução ambiente real

Abaixo fornecemos resultados obtidos a partir da execução da aplicação adpcm para execução em ambiente real.

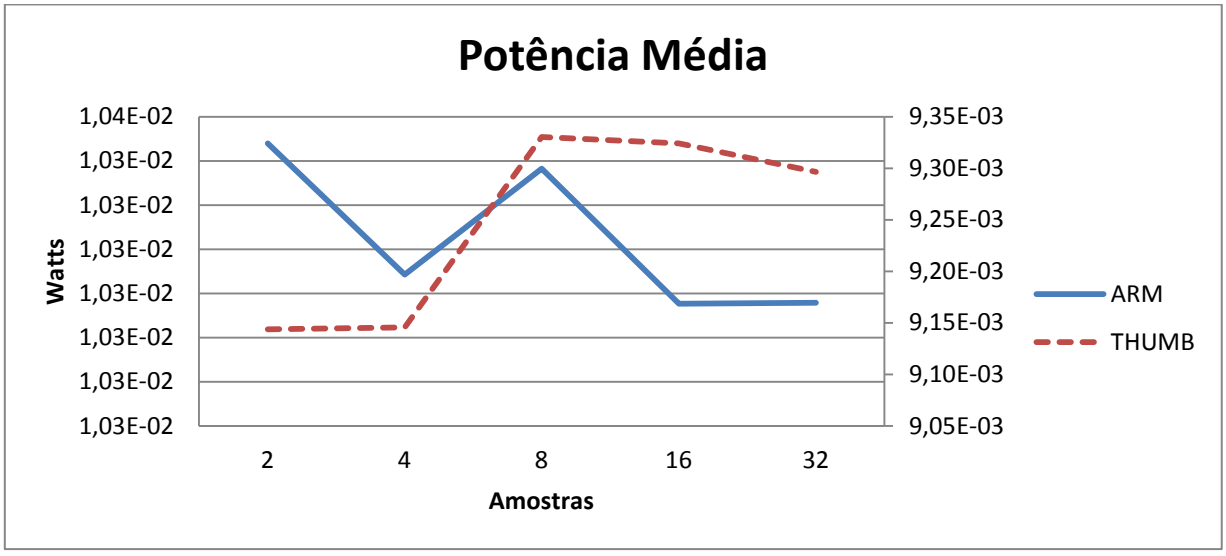

Figura 5-36 Potência Média

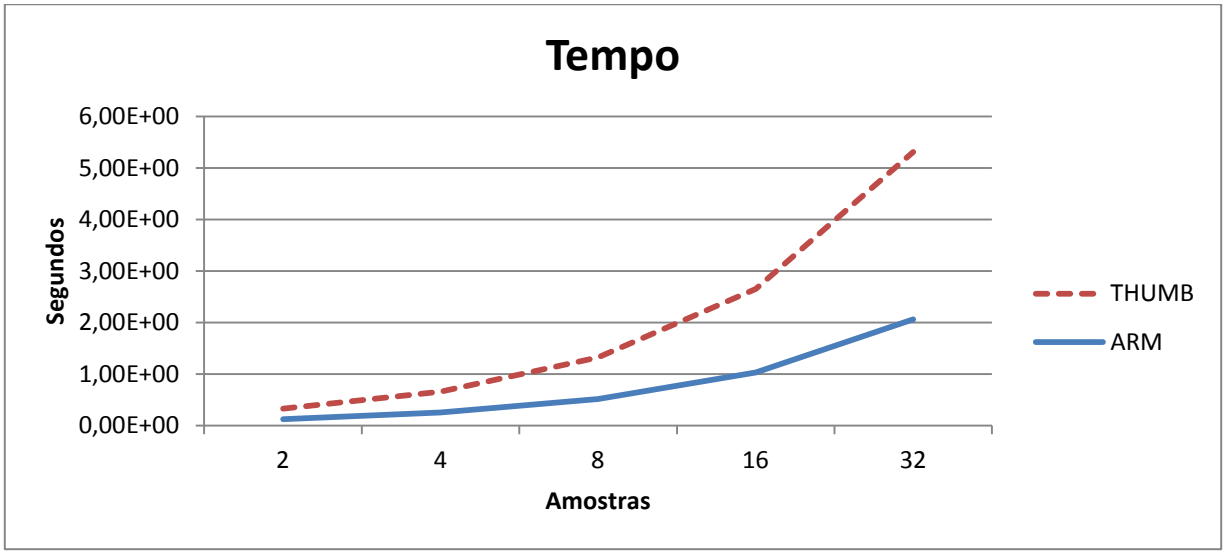

Figura 5-37 Tempo de execução

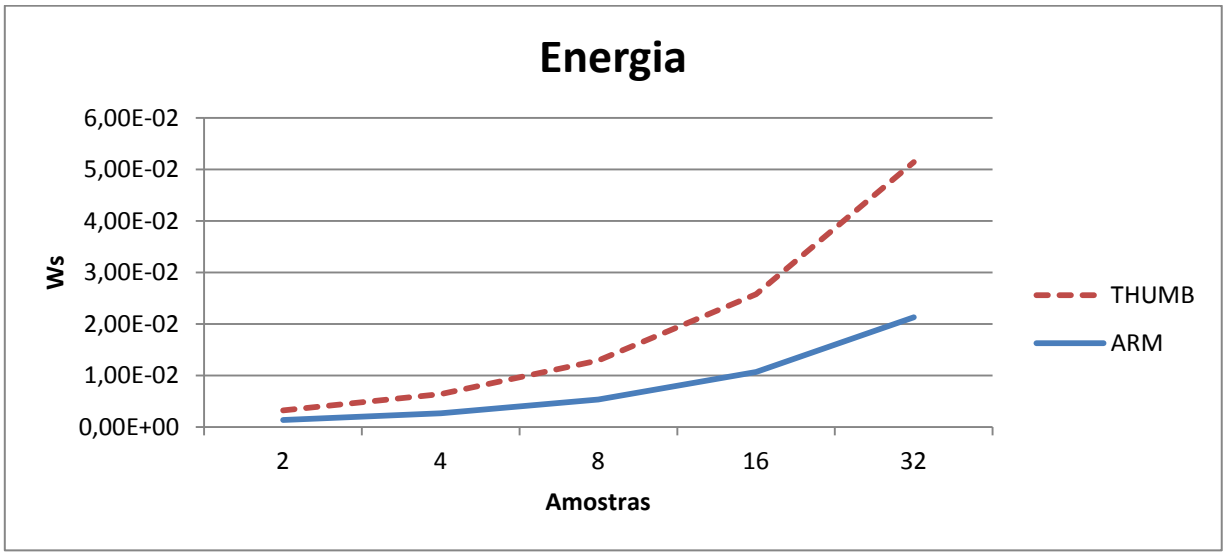

Figura 5-38 Energia consumida

5.4.1.3 BCNT – Execução ambiente real

Abaixo fornecemos resultados obtidos a partir da execução da aplicação bcnt para execução em ambiente real.

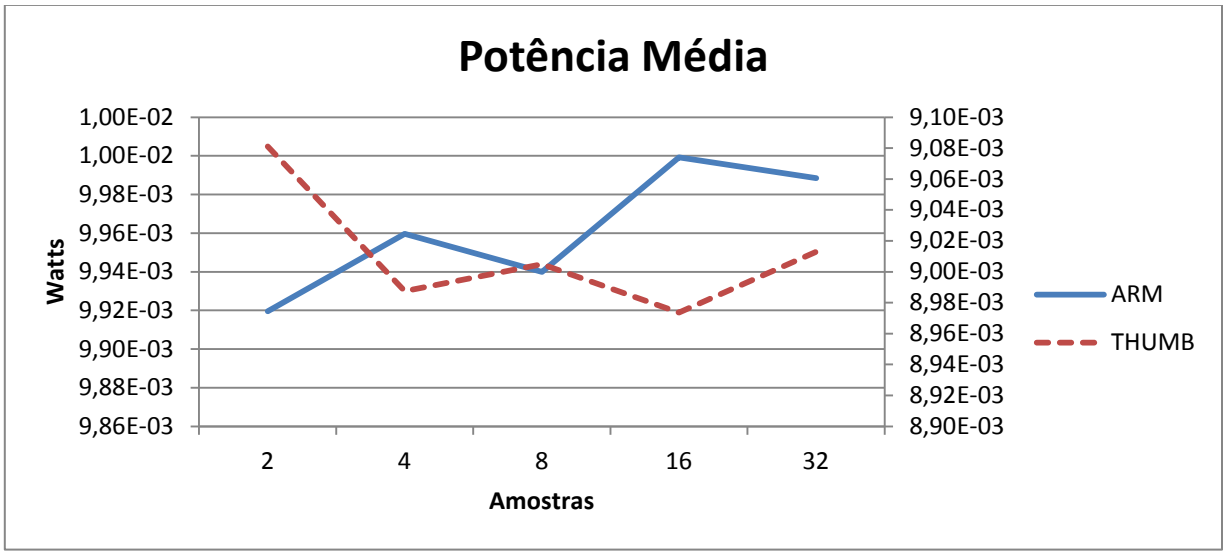

Figura 5-39 Potência Média

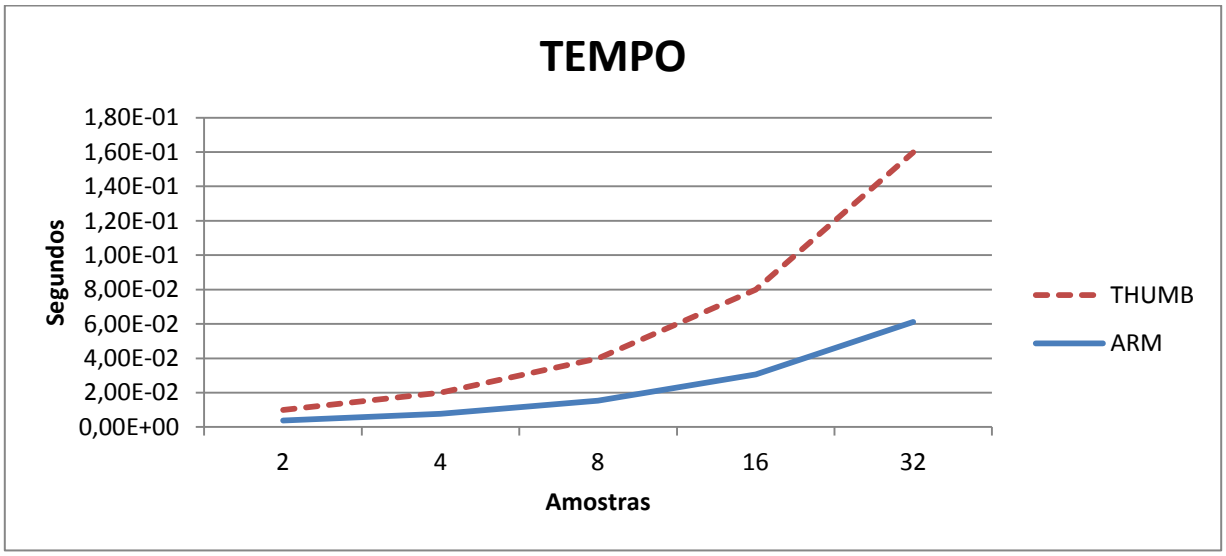

Figura 5-40 Tempo de execução

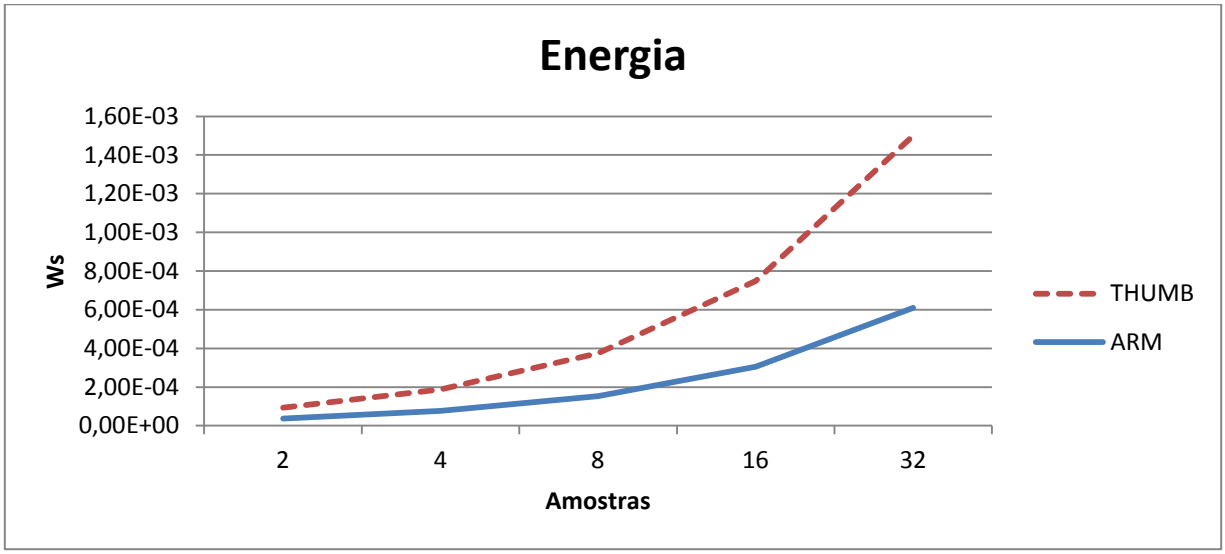

Figura 5-41 Energia consumida

5.4.1.4 CRC – Execução ambiente real

Abaixo fornecemos resultados obtidos a partir da execução da aplicação crc para execução em ambiente real.

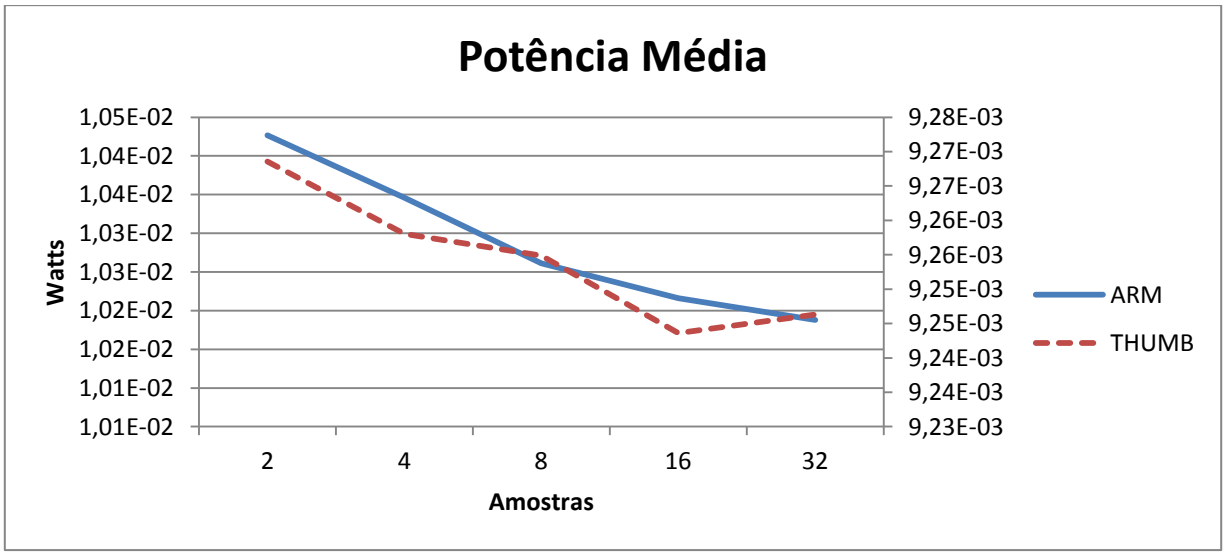

Figura 5-42 Potência Média

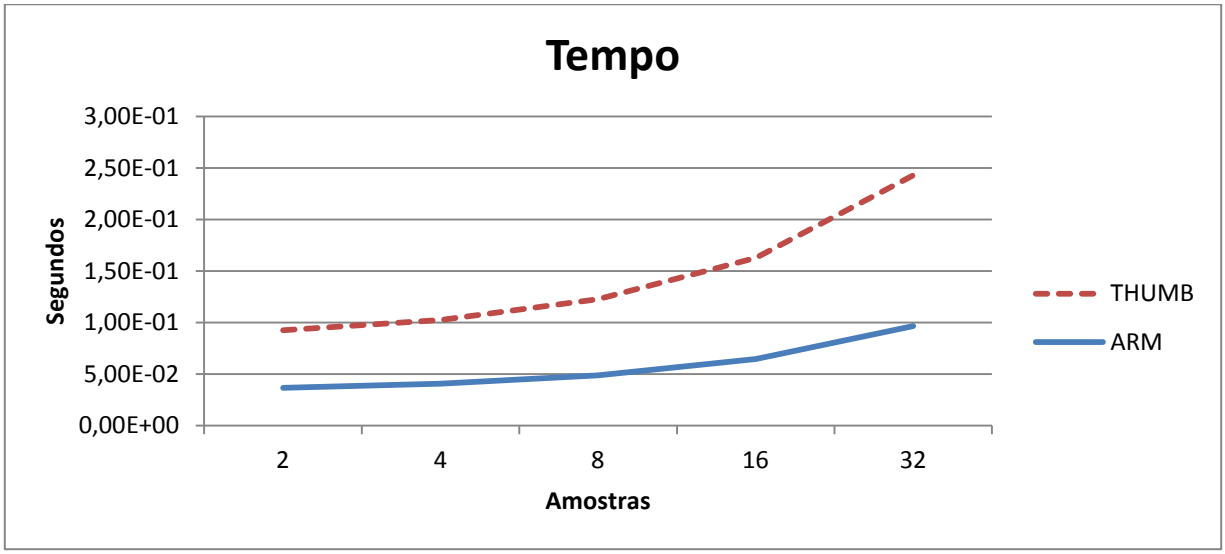

Figura 5-43 Tempo de execução

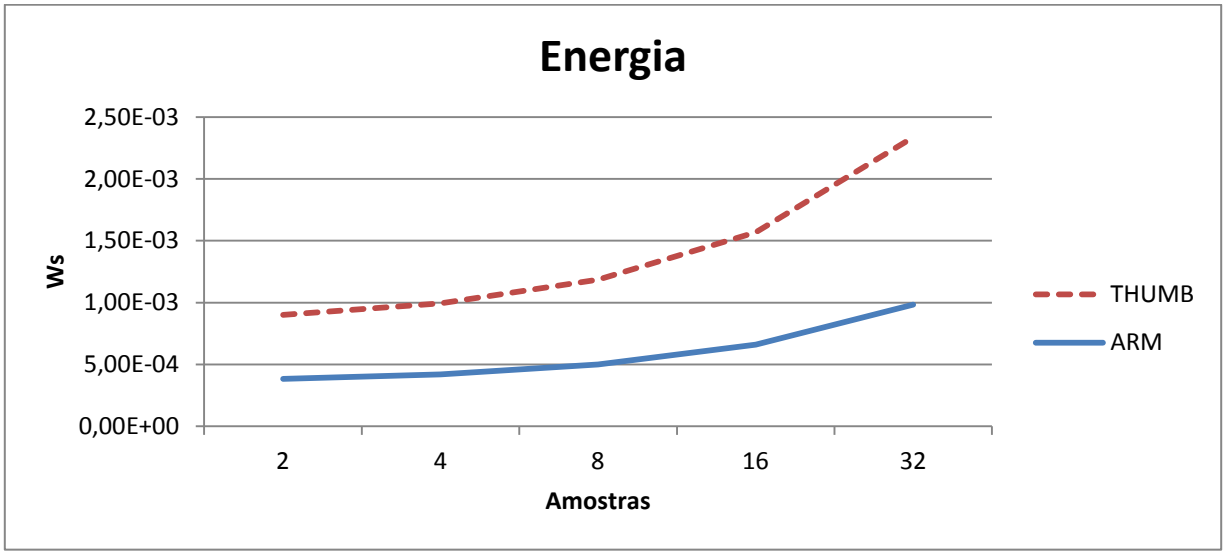

Figura 5-44 Energia consumida

5.4.1.5 DES – Execução ambiente real

Abaixo fornecemos resultados obtidos a partir da execução da aplicação blit para execução em ambiente real.

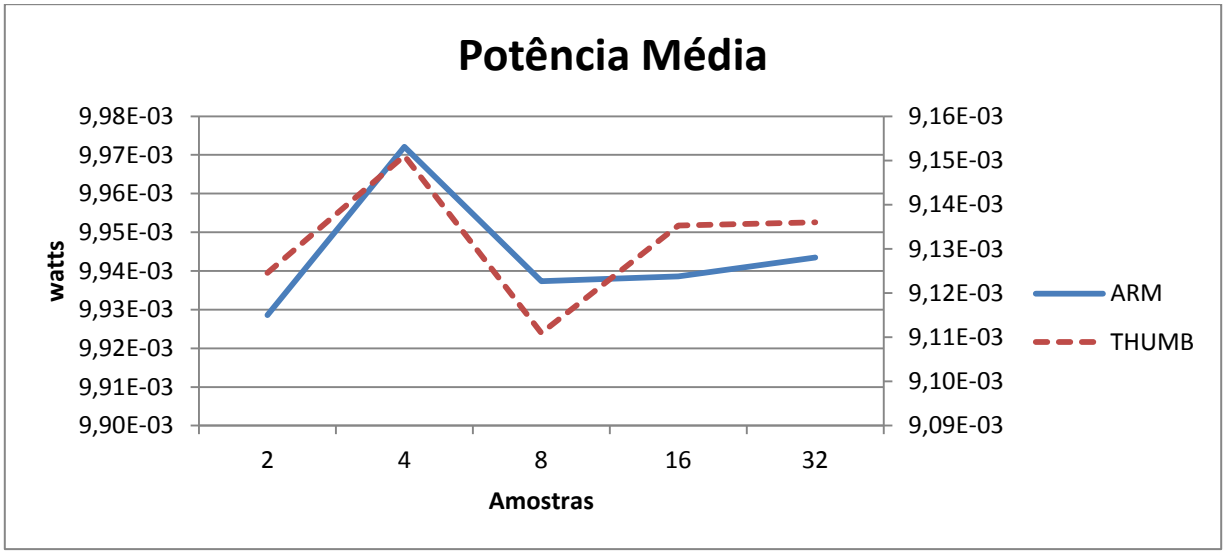

Figura 5-45 Potência Média

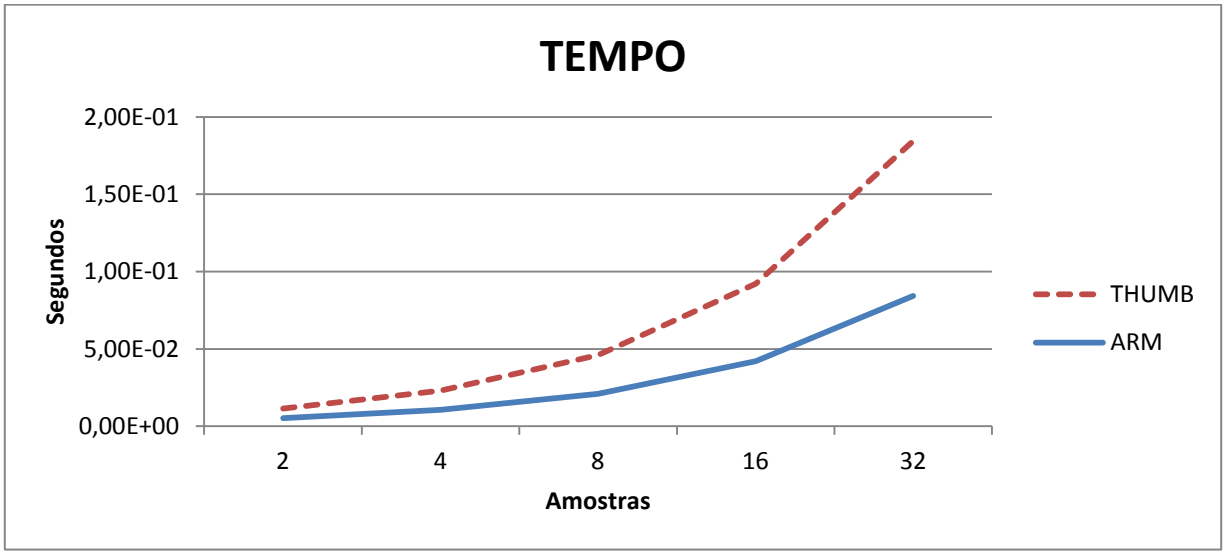

Figura 5-46 Tempo de execução

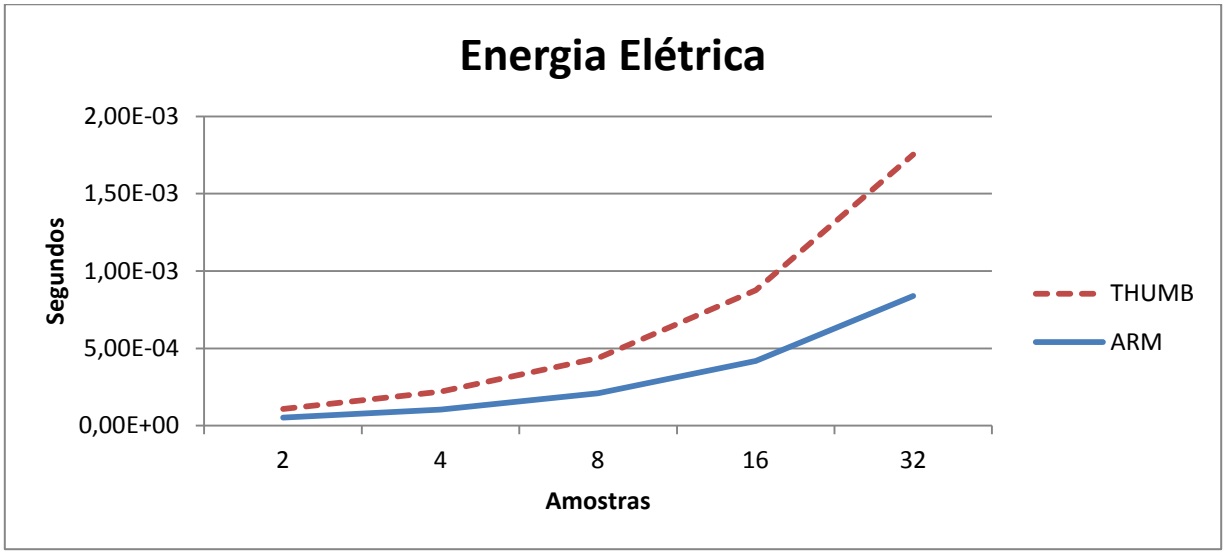

Figura 5-47 Energia consumida

5.4.2 **Execução real versus execução Simulada**

Abaixo fornecemos resultados obtidos a partir da execução da aplicações do bencharmark de forma a comparar as potências.

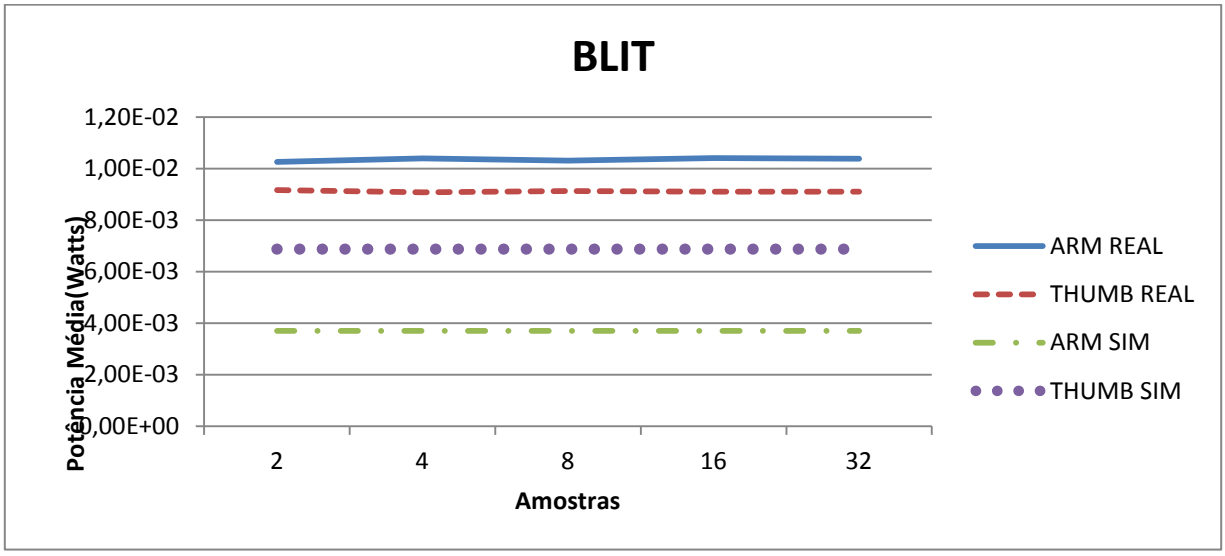

Figura 5-48 BLIT - Potência com execução real e execução simulada

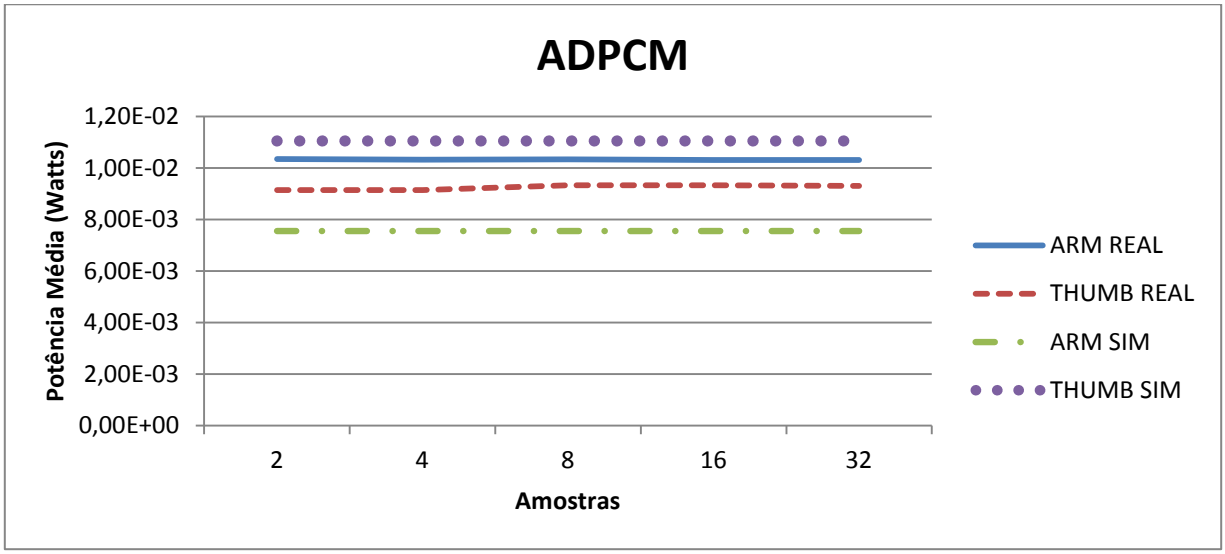

Figura 5-49 ADPCM - Potência com execução real e execução simulada

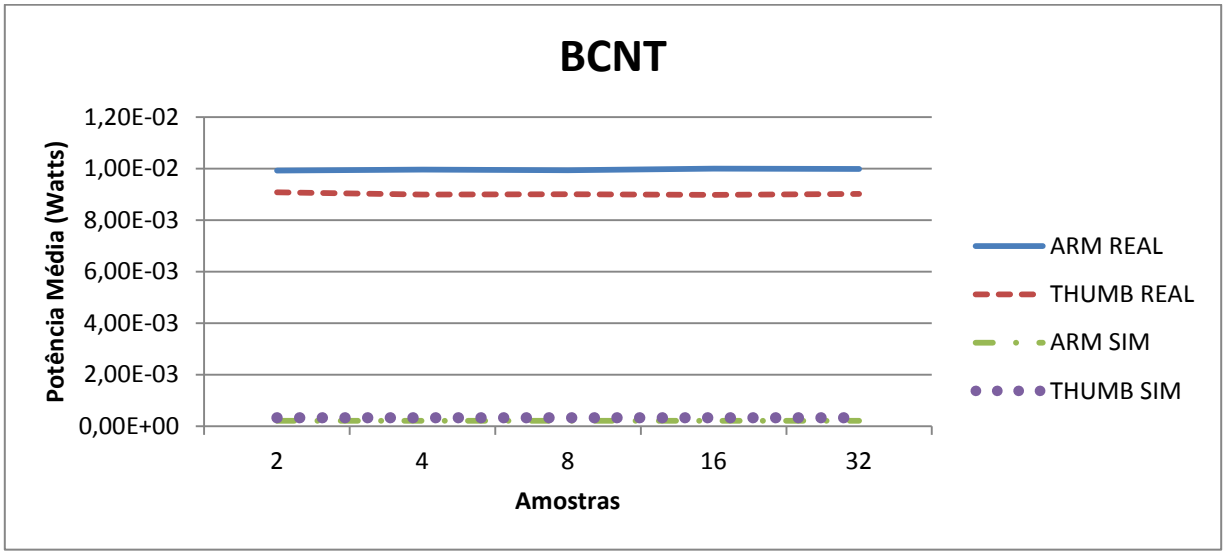

Figura 5-50 BCNT - Potência com execução real e execução simulada

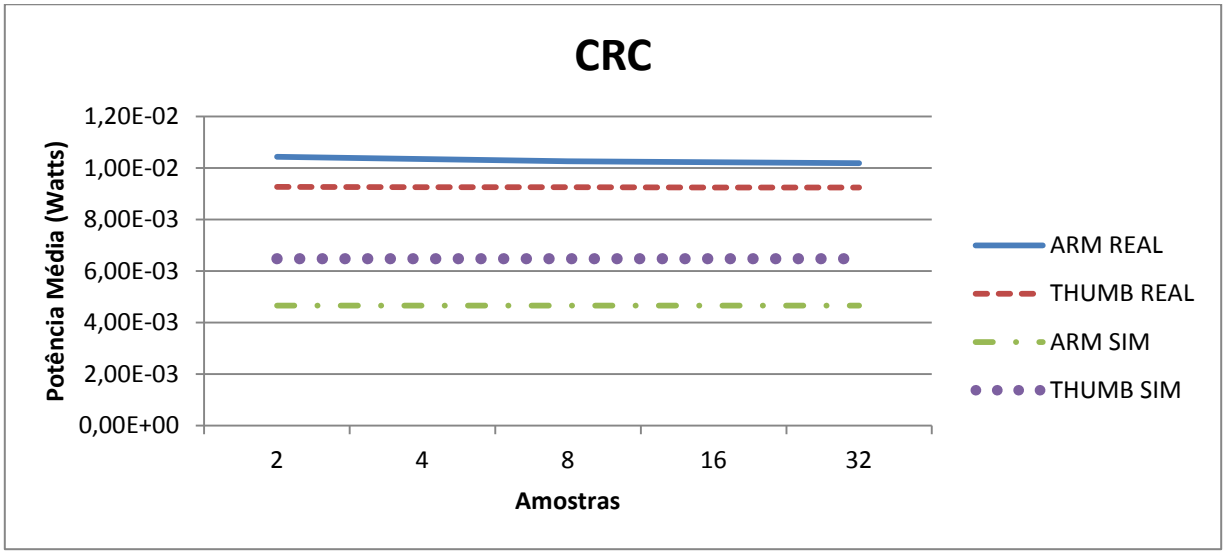

Figura 5-51 CRC - Potência com execução real e execução simulada

5.4.3 **Tamanho ARM versus THUMB**

Nesta seção abordamos o tamanho dos executáveis de forma a comparar a capacidade de densidade do modo THUMB em relação ARM nesta seção avaliamos apenas os executáveis fornecidos para o ambiente real.

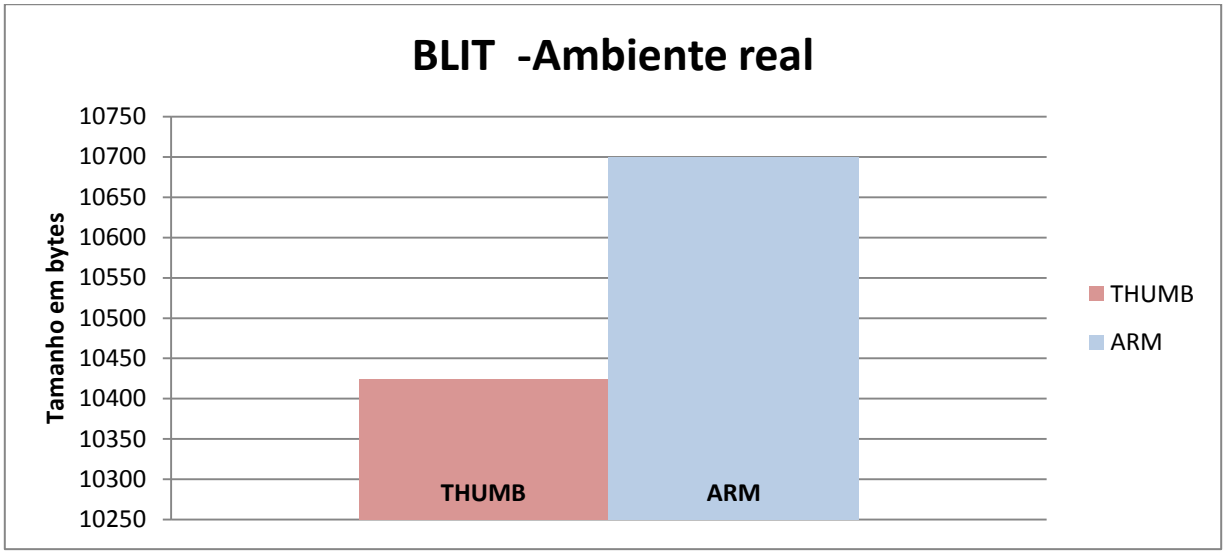

Figura 5-52 BLIT - TAMANHO

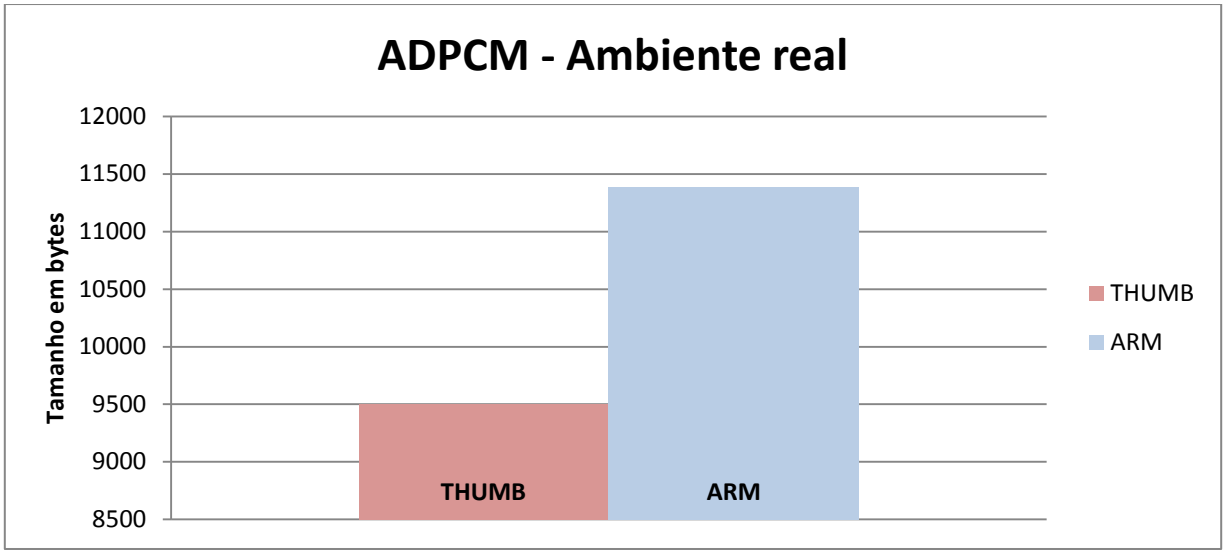

Figura 5-53 ADPCM - TAMANHO

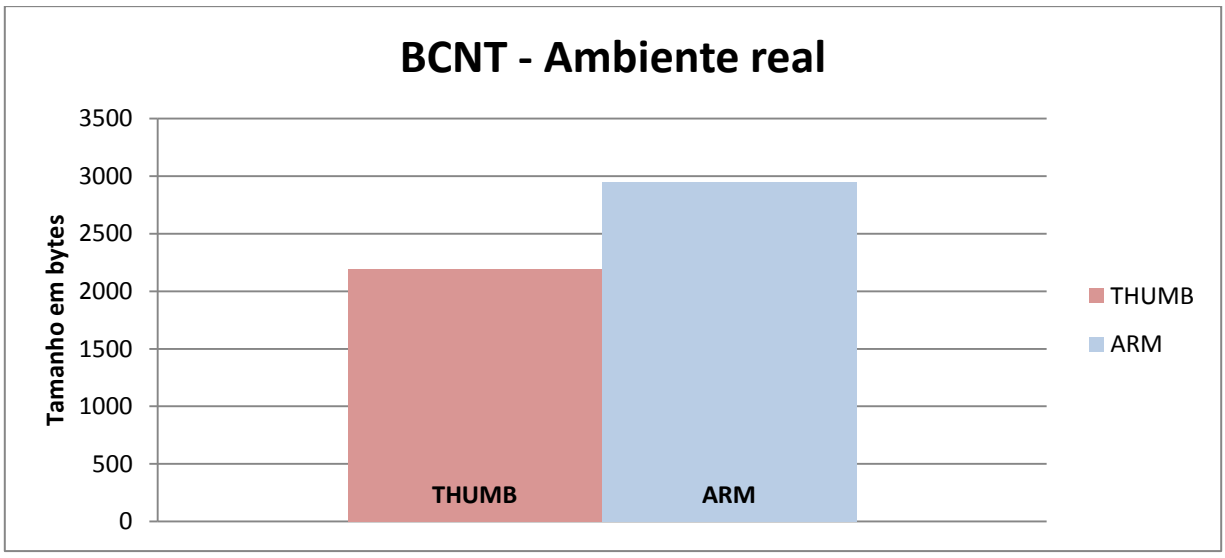

Figura 5-54 BCNT - TAMANHO

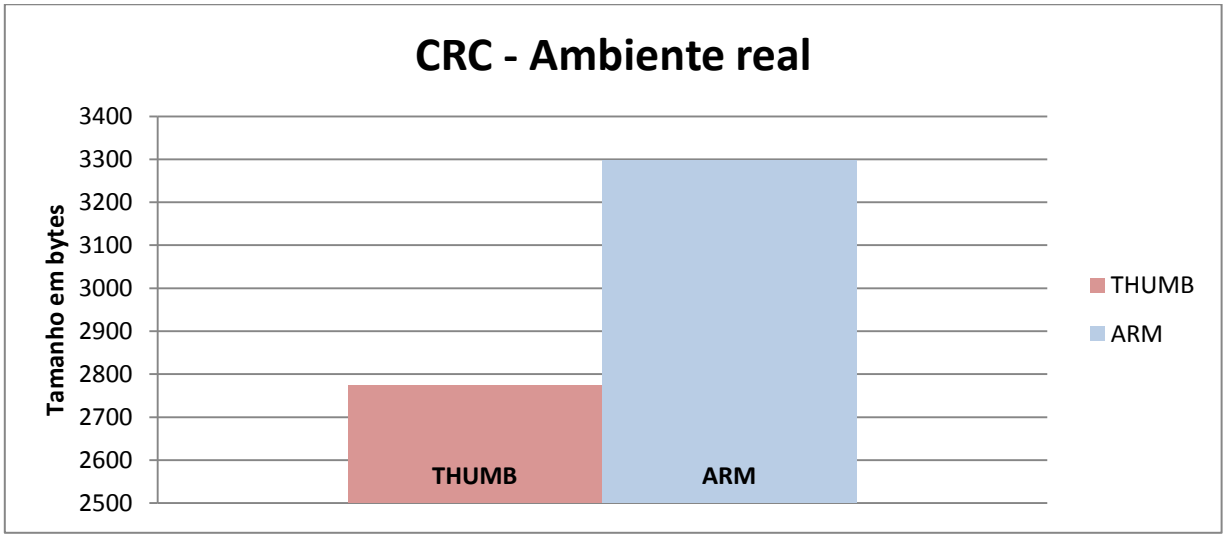

Figura 5-55 CRC - TAMANHO

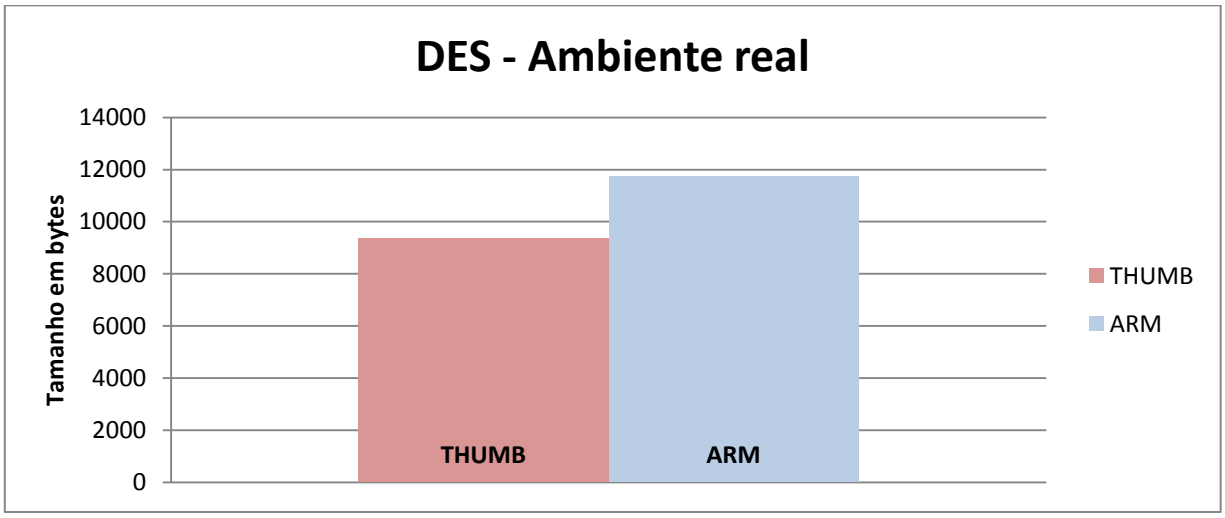

Figura 5-56 DES - TAMANHO

5.4.4 Análsie dos dados

Tabela 5-7 Relação entre instruções e instruções executadas entre Modo THUMB E ARM

Aplicação	Relação ARM/THUMB (Tamanho)	Relação THUMB / ARM Número de instruções executadas
BLIT	2,64¨%	122,46%
ADPCM	19,83%	74,86%
BCNT	34,42%	76,56%
CRC	18,88%	66,91%
DES	25,34%	21,15%

Tabela 5-8 Relação Tempo e energia elétrica entre modo thumb e modo normal

Aplicação	TEMPO THUMB/ARM * 100 = %					Energia Elétrica THUMB/ARM * 100 = %				
Iterações	2	4	8	16	32	2	4	8	16	32
BLIT	97,84	98,14	97,94	97,98	98,00	76,66	73,12	75,31	73,17	73,62
ADPCM	57,03	56,90	56,86	56,90	56,87	38,81	39,13	41,56	41,93	41,47
BCNT	60,70	61,65	61,06	61,86	61,58	47,12	45,87	45,91	45,25	45,80
CRC	52,10	51,96	51,60	51,87	51,39	35,21	35,99	36,73	37,41	37,40
DES	18,40	18,61	18,68	18,56	18,61	8,81	8,85	8,81	8,98	8,98

A partir dos dados avaliados, podemos verificar que o modo thumb consegue condensar mais código de execução e para nossos experimentos encontramos que para o módulo BLIT quando operado em modo ARM apresenta um acréscimo de 25,34% em relação ao modo THUMB. Também verificamos que de fato o modo thumb precisa executar mais instruções para realizar o mesmo trabalho que o modo arm. Na tabela 5.8 podemos visualizar que o percentual entre tempo e energia sem matem constantes ao longo das amostras analisadas. Essas informações são interessantes, pois é possível fazer uma análise sobre as variáveis que temos mais restrições. Por exemplo, se tivéssemos como restrição o tempo sobre o módulo BLIT seria melhor utilizar o sistema no modo ARM, pois o tempo para executar a mesma tarefa no modo thumb demora praticamente o dobro do modo de 32 bits.

6 - Conclusão e trabalhos futuros

Esse trabalho de graduação envolveu vários problemas interdisciplinares relacionados a eletrônica e a computação . Vários métodos dispostos principalmente em **[15] e [17]** foram utilizados. Com os métodos aplicados verificamos que é possível analisar o consumo de potência por determinado programa através de modelos de simulação sem ter a necessidade do hardware real.

Com os modelos que foram exibidos é possível para o desenvolvedor dimensionar características relacionadas a potência, energia e a tempo para execução de determinada tarefas fornecendo um grau maior de confiabilidade para o sistema .

O modelo proposto pode ser melhorado considerando atomicamente cada instrução ou seja considerando todas as variantes possíveis, apesar dessa ser a melhor proposta, ela é impraticável, devido a quantidade de modos possíveis para cada instrução e a quantidade de combinações possíveis entre duas instruções.

Outra forma de melhorar o modelo de forma que o desenvolvedor possa escolher melhor o modo a ser utilizado é o sistema dar suporte ao mecanismo que permite alternar o sistema entre o modo thumb e o modo arm. Dessa forma, podemos tentar todas as variantes possíveis e fornecer ao usuário um diagrama de pareto relacionando tempo, energia elétrica e tamanho do executável.

A ferramenta, *power analisys tool*, apresenta limitações relacionadas ao comprimento da janela, podemos contornar esse problema utilizando interrupções e calculando a potência necessária para cada tratamento de exceção no ARM. Dessa forma podemos variar a janela dinamicamente e explorar janelas que possam gerar mais dados.

A ferramenta também possui limitações quando estamos simulando determinado programa, pois o gdb utilizado considera a existência da instrução *push* no modo arm o que não é verdade, para contornar isso futuras versões vão tratar tal problema substituindo a instrução *push* pelas correspondentes no modo arm.

7 - Manual

Nessa seção descrevemos os passos necessários para instalação e configuração necessárias para utilização da ferramenta power analisys tool. Também, fornecemos a estrutura de diretórios utilizada, de forma a fornecer um nível maior de organização.

7.1 Instalação

7.1.1 Pré - Instalação

Para construir a *toolchain* é necessário baixar as ferramenta abaixo é disponibilizado as instruções para download.

Binutils Versão 2.19

Download :

ftp://ftp.gnu.org/gnu/binutils/binutils-2.19.tar.bz2

Descompactar :

tar –xvjf binutils-2.19.tar.bz2

Gcc versão 4.2.3

Download :

ftp://ftp.gnu.org/gnu/gcc/gcc-4.2.3/gcc-4.2.3.tar.bz2

Descompactar :

tar –xvjf gcc-4.2.3.tar.bz2

Newlib 1.16

Download :

ftp://sources.redhat.com/pub/newlib/newlib-1.16.0.tar.gz

Descompactar :

tar –xvzf newlib-1.16.0.tar.gz

Gdb versão 7.1

Download :

ftp://ftp.gnu.org/gnu/gdb/gdb-7.1.tar.bz2

Descompactar :

tar –xvjf gdb-7.1.tar.bz2

7.1.2 **Instalação**

Binutils Versão 2.19

Instalação :

./configure --prefix=/home/toolchain --target=arm-elf --program-prefix=arm-elf- --enable-multilib --enable-interwork

make

make install

Gcc versão 4.2.3

Pré-Instalação :

É necessário habilitar determinadas funções da multilib abaixo estão listadas as opções necessárias, isso é feito editando o arquivo t-arm-elf

```
MULTILIB_OPTIONS     = marm/mthumb
MULTILIB_DIRNAMES    = arm thumb
MULTILIB_OPTIONS     += mlittle-endian/mbig-endian
MULTILIB_DIRNAMES    += le be
MULTILIB_MATCHES     += mbig-endian=mbe mlittle-endian=mle
MULTILIB_OPTIONS    += mhard-float/msoft-float
MULTILIB_DIRNAMES   += fpu soft
MULTILIB_OPTIONS    += mno-thumb-interwork/mthumb-interwork
MULTILIB_DIRNAMES   += normal interwork
MULTILIB_OPTIONS    += mcpu=arm7
```

MULTILIB_MATCHES += mcpu?arm7=mcpu?arm7d
MULTILIB_MATCHES += mcpu?arm7=mcpu?arm7di
MULTILIB_MATCHES += mcpu?arm7=mcpu?arm70
MULTILIB_MATCHES += mcpu?arm7=mcpu?arm700
MULTILIB_MATCHES += mcpu?arm7=mcpu?arm700i
MULTILIB_MATCHES += mcpu?arm7=mcpu?arm710
MULTILIB_MATCHES += mcpu?arm7=mcpu?arm710c
MULTILIB_MATCHES += mcpu?arm7=mcpu?arm7100
MULTILIB_MATCHES += mcpu?arm7=mcpu?arm7500
MULTILIB_MATCHES += mcpu?arm7=mcpu?arm7500fe

Instalação :

./configure --prefix=/home/ toolchain --target=arm-elf --enable-interwork --program-prefix=arm-elf- --enable-multilib --with-newlib --with-headers=/home/ / newlib-1.16.0/newlib/libc

make all-gcc
make install-gcc

Gdb versão 7.1 :

Instalação :

./configure --prefix=/home /toolchain --target=arm-elf --program-prefix=arm-elf- ; make ; make install

Skyeye :

Instalação :

apt-get install skyeye

Libusb versa 0.1-4

Instalação :

apt-get install libusb-0.1-4

Power analisys Tool 1.0 Beta

Download :

www.cin.ufpe.be/~dll/pat/pat-1.0.tar.gz

Descompactar :

tar –xvzf pat-1.0.tar.gz

Instalação :

make

7.2 **Utilização**

Passos:

1. Para utilizar a ferramenta é necessário plugar o canal 2 do osciloscópio DSO3000 na porta P0.4 do lpc2103 .

2. Conectar conversor USB-RS232

Após realizado todos os passos podemos executar a ferramenta que possui os seguintes parâmetros :

3. ./pat.x Iterações Modo Processador Módulo Tipo

- Iterações
    - Representa a quantidade de iterações que desejamos. Esse valor de ser um inteiro positivo . Ex. 1,2,3 ...
- Modo
    - –marm – Modo arm instruções 32 bits
    - –mthumb – Modo Thumb instruções 16 bits
- Processador
    - arm7tdmi-s – Processador suportado
    - arm7tdmi – Processador suportado
- Tipo
    - 0 – Gera apenas arquivos
    - 1 – Gera apenas arquivos e executa ambiente virtual
    - 2 – Gera apenas arquivos e executa ambiente virtual
- Módulo
    - Nome do arquivo que contem o código fonte do módulo a ser testado

7.3 **Estrutura de diretórios**

Para criar uma estrutura mais organizada foi estabelecido uma estrutura de diretórios sendo eles :

Tabela 7-1 Estrutura de diretórios

Diretório	Descrição
exe	Contêm os executáveis
output	Contêm a saída gerada pelo modo virtual e o modo embarcado. Os diretórios filhos são criados de acordo com o nome dos módulos executados
utils	Possuem arquivos úteis para caracterização dos arquivos gerados para o ambiente virtual e ambiente real
pat	Sources do *power analisys tool*
parser_gdb_output	*Sources* do *parser* que trata informações resultantes do gdb
parser_osc_output	*Sources* do *parser* que trata informações resultantes do osciloscópio

8 - Referências

[1] Agilent 3000 Series Oscilloscopes - Programmer's Reference,Online, acesso 12/12/2010 na url http://cp.literature.agilent.com/litweb/pdf/D3000-97016.pdf

[2] Agilent 3000 Series Oscilloscopes - User's and service guide,Online, acesso 12/12/2010 na url http://cp.literature.agilent.com/litweb/pdf/D3000-97018.pdf

[3] ARM7TDMI Processor - Online, acesso em 12/12/2010 na url http://www.arm.com/support/arm7tdmi.php

[4] ARM7TDMI-S Revision: r4p3 Technical Reference Manual,Online, acesso 12/12/2010 na url http://infocenter.arm.com/help/topic/com.arm.doc.ddi0234b/DDI0234.pdf

[5] Arvind Krishnaswamy and Rajiv Gupta. 2002. Profile guided selection of ARM and thumb instructions. In Proceedings of the joint conference on Languages, compilers and tools for embedded systems: software and compilers for embedded systems (LCTES/SCOPES '02). ACM, New York, NY, USA, 56-64. DOI=10.1145/513829.513840

[6] Embedded Systems: Technologies and Markets - Online, acesso 12/12/2010 na url http://www.electronics.ca/publications/products/Embedded-Systems:-Technologies-and-Markets.html

[7] Embedded Systems: Technologies and Markets - Online, acesso 12/12/2010 na url http://www.bccresearch.com/report/IFT016C.html

[8] Gajski D. D., Abdi S., Gerstlauer A., and Schirner G., **"**Embedded System Design: Modeling, Synthesis, Verification**"** Springer, ISBN 978-1-4419-0503-1, July 2009.

[9] Jayaseelan, R.; Mitra, T.; Xianfeng Li; , "Estimating the Worst-Case Energy Consumption of Embedded Software," Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE , vol., no., pp. 81- 90, 04-07 April 2006 doi: 10.1109/RTAS.2006.17

[10] Laopoulos, T.; Neofotistos, P.; Kosmatopoulos, C.A.; Nikolaidis, S.; , "Measurement of current variations for the estimation of software-related power consumption [embedded processing circuits]," Instrumentation and Measurement, IEEE Transactions on , vol.52, no.4, pp. 1206- 1212, Aug. 2003 doi: 10.1109/TIM.2003.816837

[11] LPC2101/02/03 User manual, Online, acesso 12/12/2010 na url http://www.nxp.com/documents/user_manual/UM10161.pdf

[12] LPC2101/02/03 Datasheet, Online, acesso 12/12/2010 na url http://www.nxp.com/documents/data_sheet/LPC2101_02_03.pdf

[13] Russell, J.T.; Jacome, M.F.; , "Software power estimation and optimization for high performance, 32-bit embedded processors," Computer Design: VLSI in Computers and Processors, 1998. ICCD '98. Proceedings. International Conference on , vol., no., pp.328-333, 5-7 Oct 1998 doi: 10.1109/ICCD.1998.727070

[14] Skyeye User's Manual version 0.1,Online, acesso 12/12/2010 na url http://skyeye-simulator.googlegroups.com/web/skyeye_user_manual_eng_v1.pdf?gda=iLUrZE8AAACBTQZ6sMQYY15u77tF0if6Mq9-jByS5WLE8LH46Ucv_o_vyd3n3Y04VhwUEYOBLVeEwLoGEKRtZWEhYGDy2WnnHMhSp_qzSgvndaTPyHVdA

[15] Tiwari, V.; Malik, S.; Wolfe, A.; , "Power Analysis Of Embedded Software: A First Step Towards Software Power Minimization," Computer-Aided Design, 1994., IEEE/ACM International Conference on , vol., no., pp.384-390, 6-10 Nov 1994 doi: 10.1109/ICCAD.1994.629825

[16] VAHID, F., and GIVARGIS, T. Embedded System Design. A Unified Hardware/Software Introduction. John Wiley & Sons, Inc., United States of America, 17 de Outubro 2001.

[17] Zotos, K.; LITKE, A.; Chatzigeorgiou, E.; Nikolaidis, S.; Stephanides, G.; , "Energy complexity of software in embedded system", ACIT - Automation, Control, and Applications,Journal ,2005.