UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

PAULO ROBERTO CEMIN

PLATAFORMA DE MEDIÇÃO DE CONSUMO PARA COMPARAÇÃO ENTRE SOFTWARE E HARDWARE EM PROJETOS ENERGETICAMENTE EFICIENTES

DISSERTAÇÃO

CURITIBA

2015

PAULO ROBERTO CEMIN

PLATAFORMA DE MEDIÇÃO DE CONSUMO PARA COMPARAÇÃO ENTRE SOFTWARE E HARDWARE EM PROJETOS ENERGETICAMENTE EFICIENTES

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de "Mestre em Ciências" – Área de Concentração: Informática Industrial.

Orientador: Volnei Antonio Pedroni

CURITIBA

2015

Dados Internacionais de Catalogação na Publicação

C394p 2015

Cemin, Paulo Roberto
Plataforma de medição de consumo para comparação entre software e hardware em projetos energeticamente eficientes / Paulo Roberto Cemin.-- 2015.
99 f.: il.; 30 cm

Texto em português, com resumo em inglês.
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Curitiba, 2015.
Bibliografia: f. 95-98.

1. Energia - Consumo. 2. Medição. 3. Coprocessadores. 4. Arranjos de lógica programável em campo. 5. Sistemas de detecção de intrusão (Segurança do computador). 6. Software - Desenvolvimento. 7. Métodos de simulação. 8. Engenharia elétrica - Dissertações. I. Pedroni, Volnei A. (Volnei Antonio), orient. II. Universidade Tecnológica Federal do Paraná - Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD 22 -- 621.3

Biblioteca Central da UTFPR, Câmpus Curitiba

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba

Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

Título da Dissertação Nº. ____

PLATAFORMA DE MEDIÇÃO DE CONSUMO PARA COMPARAÇÃO ENTRE SOFTWARE E HARDWARE EM PROJETOS ENERGETICAMENTE EFICIENTES

Por

PAULO ROBERTO CEMIN

Orientador: Prof. Dr. VOLNEI ANTONIO PEDRONI

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM CIÊNCIAS – Área de Concentração: ENGENHARIA DE AUTOMAÇÃO E SISTEMAS do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR, às 09:h00 do dia 24 de fevereiro de 2015. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores doutores:

Prof. Dr. VOLNEI ANTONIO PEDRONI
(Presidente – UTFPR)

Prof. Dr. ALTAIR OLIVO SANTIN
(PUC-PR)

Prof. Dr. LUCIANO SCANDELARI
(UTFPR)

Visto da coordenação:

Prof. Dr Emilio Carlos Gomes Wille
(Coordenador do CPGEI)

"A Folha de Aprovação assinada encontra-se na Coordenação do Programa de Pós Graduação de Engenharia Elétrica e Infomática Industrial."

Dedico este trabalho a Cristiane Regina Melz, que me incentivou a buscar um crescimento acadêmico e, sobretudo, pela sua dedicação a fim de viabilizar a concretização deste trabalho.

AGRADECIMENTOS

Agradeço ao professor Dr. Volnei Antonio Pedroni pela orientação e por compreender a inviabilidade de me dedicar exclusivamente às atividades acadêmicas. Sou grato também pelos aperfeiçoamentos que me foram oportunizados em diversas áreas da minha formação.

Agradeço ao André França e ao Dr. Ricardo Pereira Jasinski, amigos e companheiros de pesquisa, pelo grande esforço e dedicação para a implementação da aplicação apresentada no capítulo 4, além das grandes contribuições provenientes das exaustivas revisões dos artigos elaborados no decorrer desta pesquisa.

Agradeço aos colegas da Pontifícia Universidade Católica do Paraná, Dr. Altair Olivo Santin e Eduardo Viegas, pelas inúmeras contribuições a este trabalho, além de serem os responsáveis pelo desenvolvimento dos algoritmos de extração de características e de classificação descritos na seção 4.2.1.

Devido à oportunidade de estudar profundamente a plataforma LEAP e de receber-me no laboratório Actuated Sensing and Coordinated Embedded Networked Technologies (ASCENT) da Universidade da Califórnia em Los Angeles (UCLA), agradeço ao Dr. William Keiser e ao Dr. Digvijay Singh.

E por fim, mas não menos importante, agradeço aos amigos da Perkons S.A.: M.Sc. Rodrigo Trebien, M.Sc. Leandro Guerra Becker e M.Sc. Adriel Bilharva da Silva, por terem me disponibilizado o tempo necessário para dedicar-me aos estudos relacionados a esta pesquisa.

“A menos que modifiquemos a nossa maneira de pensar, não seremos capazes de resolver os problemas causados pela forma como nos acostumamos a ver o mundo”. (Albert Einstein)

RESUMO

CEMIN, Paulo Roberto. PLATAFORMA DE MEDIÇÃO DE CONSUMO PARA COMPARAÇÃO ENTRE SOFTWARE E HARDWARE EM PROJETOS ENERGETICAMENTE EFICIENTES. 99 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

A popularização dos dispositivos móveis impulsionou a pesquisa e o desenvolvimento de soluções de baixo consumo. A evolução destas aplicações demanda ferramentas que permitam avaliar diferentes alternativas de implementação, fornecendo, aos desenvolvedores, informações valiosas para a criação de soluções energeticamente eficientes. Este trabalho desenvolveu uma nova plataforma de medição de consumo que permite comparar o consumo de energia de diferentes algoritmos implementados em software e em hardware. A plataforma é capaz de medir o consumo energético de um processo específico em execução em um processador de propósito geral com um sistema operacional padrão, além de comparar o resultado obtido com algoritmos equivalentes implementados em uma FPGA. Isto permite ao desenvolvedor dividir o processamento da aplicação entre software e hardware de forma a obter a solução mais energeticamente eficiente. Comparada com o estado da arte, a plataforma de medição criada possui quatro característica inovadoras: suporte à medição de consumo de software e hardware; medição de trechos de código específicos executados pelo processador; suporte à alteração dinâmica do clock; e aumento da precisão da medida. Também é mostrado neste trabalho como a plataforma desenvolvida tem sido utilizada para analisar o consumo energético de algoritmos de detecção de intrusão de rede para ataques do tipo *probing*.

Palavras-chave: Eficiência Energética. Medição de Consumo. Coprocessador em FPGA. Detecção de Intrusão de Rede. PCIe.

ABSTRACT

CEMIN, Paulo Roberto. POWER MEASUREMENT PLATFORM FOR SOFTWARE VS. HARDWARE COMPARISON IN LOW-POWER DESIGNS. 99 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

The large number of mobile devices increased the interest in low-power designs. Tools that allow the evaluation of alternative implementations give the designer actionable information to create energy-efficient designs. This paper presents a new power measurement platform able to compare the energy consumption of different algorithms implemented in software and in hardware. The proposed platform is able to measure the energy consumption of a specific process running in a general-purpose CPU with a standard operating system, and to compare the results with equivalent algorithms running in an FPGA. This allows the designer to choose the most energy-efficient software vs. hardware partitioning for a given application. Compared with the current state-of-the-art, the presented platform has four distinguishing features: (i) support for both software and hardware power measurements, (ii) measurement of individual code sections in the CPU, (iii) support for dynamic clock frequencies, and (iv) improvement of measurement precision. We also demonstrate how the developed platform has been used to analyze the energy consumption of network intrusion detection algorithms aimed at detecting probing attacks.

Keywords: Energy Efficiency. Low-power Design. Power Measurement. FPGA Coprocessor. Network Intrusion Detection. PCIe.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

API	Application Programming Interface
APM	Aplication Power Management
ASCENT	Actuated Sensing and Coordinated Embedded Networked Technologies
ASICs	Application Specific Integrated Circuits
CFS	Completely Fair Scheduler
CPLDs	Complex Programmable Logic Devices
CPU	Central Processing Unit
DAQ	Data Aquisition Tool
DMA	Direct Memory Access
DT	Decision Tree
FPGA	Field Programmable Gate Array
GAL	Generic Array Logic
GPU	Graphics Processing Unit
HD	Hard Disk
HPC	High-Performance Computing
IDS	Intrusion Detection System
IPs	Intelectual Property Functions
ISRA	Intel Strategic Research Alliance
KNN	K-Nearest Neighbors
LE	Logical Element
LEAP	Low-power Energy Aware Platform
LUT	Look-Up Table
ML	Machine Larning
PAL	Programmable Array Logic
PC	Program Counter
PCIe	PCI Express
PID	Process ID
PLA	Programmable Logic Array
PLDs	Programmable Logic Devices
RAPL	Running Average Power Limiting
SO	Sistema Operacional
SoCs	Systems On Chip
SPLDs	Simple Programmable Logic Devices
SVM	Support Vector Machine
TGID	Thread Group ID
TSC	Time Stamp Counter
UCLA	University of California, Los Angeles

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

O número de dispositivos eletrônicos alimentados por bateria tem aumentado consideravelmente nos últimos anos. A principal causa deste aumento deve-se à popularização dos dispositivos móveis com acesso à internet, representado em sua maioria pelos *Smartphones* e *Tablets*. Em 2013, foi alcançada a marca de 7 bilhões de aparelhos móveis, igualando-se ao número de pessoas, resultado de um aumento de mais de meio bilhão de dispositivos (526 milhões) em relação a 2012. Evidências claras apontam que este crescimento ainda se acelera, como pode ser observado através da figura 1, que apresenta o constante aumento da proporção de dispositivos móveis que acessam a internet em relação às demais plataformas. Nos últimos cinco anos, o acesso móvel à internet, que representava apenas 1,8% em 2010, subiu para 30,9% em 2014 (CISCO, 2014).

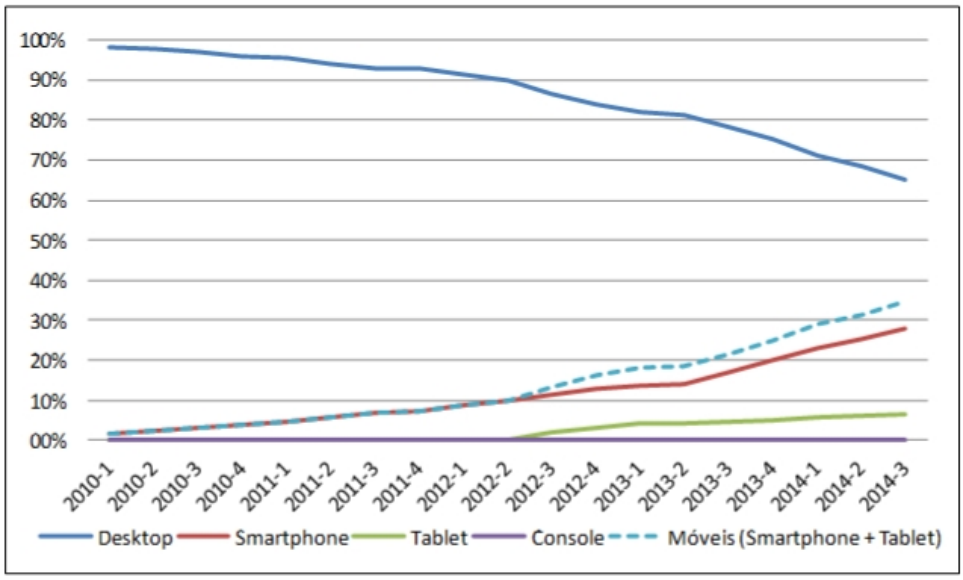


Figura 1: Comparação por plataforma dos dispositivos conectados à internet nos últimos 5 anos indicando um acelerado aumento da proporção de dispositivos móveis.

Fonte: (STATCOUNTER, 2014).

Dispositivos móveis possuem requisitos específicos, dentre os quais se destaca a necessidade de ser o mais eficiente energeticamente possível, uma vez que são alimentados através de baterias. Outro requisito relevante é a necessidade de proteção contra os mais variados tipos de ameaça à segurança da informação, pois estes dispositivos frequentemente estão conectados a redes não confiáveis, como é o caso da internet. Devido a estes dois fatores, e também ao grande número de dispositivos existentes, a busca pela eficiência energética das aplicações móveis, principalmente relacionadas à segurança contra ataques de rede, tem recebido grande atenção das empresas e instituições de pesquisa. Um exemplo é o programa *Strategic Research Alliance* (ISRA) da Intel, que fomentou este projeto à procura de soluções energeticamente eficientes para a segurança da informação em Sistemas em um Chip (SoCs - *Systems On Chip*).

Um método promissor para reduzir o consumo energético de uma aplicação é mover tarefas computacionais do software para hardwares dedicados. Devido ao fato do hardware poder ser customizado para uma tarefa específica, geralmente consome menos recursos e energia que um processador genérico.

Apesar da atual demanda por sistemas de baixo consumo, existem poucas ferramentas para medir e avaliar o consumo energético de aplicações em software. Comparações entre o consumo de implementações em software e hardware ainda são imprecisas e não fornecem informações suficientemente detalhadas para identificar partes da aplicação que poderiam ser otimizadas. Não se encontra na literatura ou no mercado uma plataforma capaz de fornecer informações suficientes para o desenvolvimento de projetos de baixo consumo, em que se busca avaliar o ganho energético com o emprego de hardwares dedicados ou encontrar a relação ótima entre a parcela da aplicação a ser implementada em software e em hardware.

Apesar das ferramentas de análise e estimativa do consumo baseadas em modelos teóricos fornecerem informações relevantes, apontando possíveis gargalos para a eficiência do sistema, apenas a medição é capaz de indicar a significante parcela do consumo energético causada por atividades assíncronas, como interrupções provenientes da rede ou de periféricos externos e disputa por recursos entre diferentes aplicações.

A maior dificuldade para a medição do consumo energético em sistemas que possuem aplicação em software deve-se à presença de um Sistema Operacional (SO) de propósito geral com escalonamento preemptivo de tarefas, como é o caso do Linux, Windows, Android, entre outros. Nestes SOs, múltiplas tarefas compartilham os recursos do hardware através de uma distribuição no tempo. Uma vez que cada tarefa possui características particulares quanto à sua execução, apresentam consumos distintos entre si, como mostrado na figura 2. Desta forma, a

correta atribuição do consumo energético de uma tarefa específica executada por um SO não é trivial e exige o desenvolvimento de técnicas de sincronismo entre as amostras de consumo e a execução do algoritmo a ser medido.

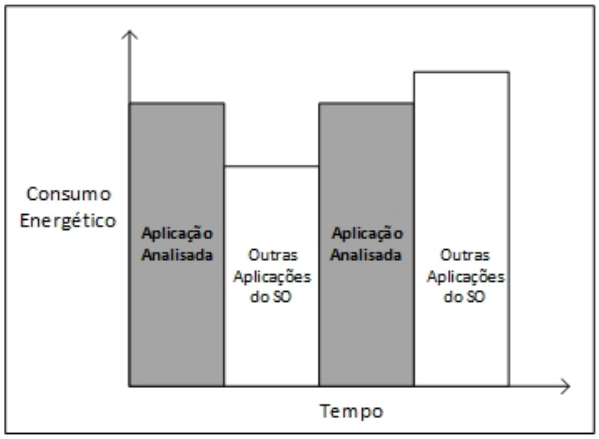


Figura 2: Ilustração do compartilhamento no tempo realizado pelo escalonador do sistema operacional e os diferentes consumos energéticos de cada tarefa.

Fonte: Autoria própria.

Além de discriminar a tarefa analisada das demais tarefas em execução, outra informação importante é o consumo dedicado a cada etapa do processamento. Com isto, dedica-se esforços com a otimização dos trechos da aplicação que demandam maior consumo.

Considerando a grande quantidade de projetos visando aplicações de baixo consumo e as restrições das ferramentas de análise disponíveis, este trabalho desenvolve uma nova plataforma de medição de consumo que irá suportar projetos que buscam o aumento da eficiência energética através da utilização de hardwares dedicados. Destacam-se como características inovadoras desta plataforma: a capacidade de discriminar uma tarefa única em execução no sistema, a medição de trechos específicos do algoritmo analisado e o suporte a coprocessadores criados através de FPGA (*Field Programmable Gate Array*).

O desenvolvimento da plataforma exigiu extenso estudo dos métodos existentes de medição, destacando-se a avaliação da LEAP (*Low-power Energy Aware Platform*) realizada com o auxílio de um dos seus desenvolvedores, Dr. Digvijay Singh, membro do laboratório ASCENT (*Actuated Sensing and Coordinated Embedded Networked Technologies*) da Universidade da Califórnia em Los Angeles (UCLA - *University of California, Los Angeles*). A LEAP é uma das mais avançadas plataformas de medição de consumo de softwares, o presente trabalho é um avanço na abordagem pioneiramente adotada nessa plataforma, adicionando principalmente a capacidade de separar a energia consumida pela aplicação medida dos demais processos ativos no sistema operacional e o suporte à comparação de consumo com

hardwares dedicados.

A plataforma de medição criada é importante parte do projeto que visa prover informações quanto à eficiência energética de hardwares para a detecção de intrusão de redes, quando comparados com soluções em software, desenvolvido com o esforço conjunto de equipes de pesquisa das universidades UTFPR, PUC-PR e UFPR. O objetivo em longo prazo é o desenvolvimento de uma infraestrutura para a análise do consumo de diversos algoritmos. A figura 3 mostra a arquitetura planejada para o sistema completo, que incluirá cinco algoritmos de classificação. O desenvolvimento da plataforma foi indispensável para a medição do consumo neste cenário.

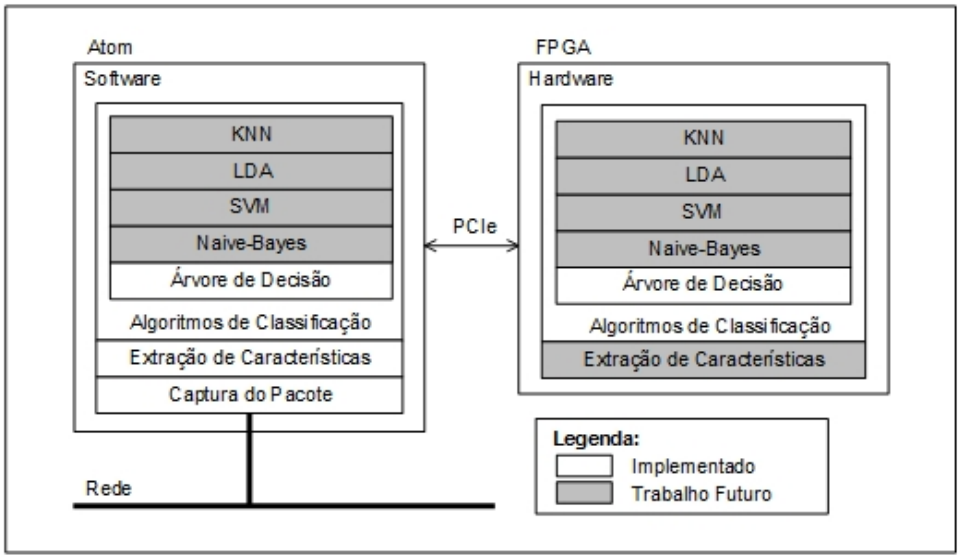


Figura 3: Arquitetura do sistema em desenvolvimento com o auxílio da plataforma de medição criada.

Fonte: Adaptada de (FRANCA et al., 2014a).

Neste trabalho também é analisada a aplicação do método de medição desenvolvido na atual versão do sistema, na qual os módulos para a captura do pacote de dados e a extração de características são implementados em software, enquanto que o classificador baseado em Árvore de Decisão é implementado em software e em hardware, através de uma FPGA operando como coprocessador. O consumo para o processamento apenas em software é comparado com o sistema que realiza a classificação em hardware. Em cada uma das implementações é discriminado o consumo dedicado a cada etapa do processamento, permitindo identificar os trechos em que é demandado maior consumo energético.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

O objetivo deste trabalho é o desenvolvimento de uma plataforma para medição do consumo energético de aplicações implementadas em software e/ou em hardware que forneça informações para o desenvolvimento de soluções de baixo consumo e permita a comparação entre o desempenho de tarefas equivalentes realizadas por software ou hardware. Para isto, a plataforma disponibiliza à aplicação em desenvolvimento um processador x86-64 de uso genérico e uma FPGA como coprocessador.

O desenvolvimento desta plataforma é necessário para a análise da eficiência energética de sistemas híbridos, em que parte do processamento é realizada em software e parte através de um hardware dedicado. A medição do consumo de aplicações nestes sistemas exige requisitos específicos, destacando-se: a medição do consumo demandado por uma determinada aplicação em execução no sistema, sem sofrer influência das demais; uma elevada taxa de amostragem; a medição precisa do tempo de execução da aplicação em análise; e o suporte à utilização de coprocessadores em hardware. A plataforma resultante deste trabalho é a única a atender todos estes requisitos.

Perante a carência de soluções na literatura e no mercado, objetiva-se também viabilizar o desenvolvimento e as medições de consumo do projeto fomentado pelo programa ISRA da Intel, no qual se analisa a eficiência energética de algoritmos para detecção de intrusão de rede implementados em software e hardware. Como exemplo de aplicação da plataforma, neste trabalho é avaliado o consumo de uma aplicação para detecção de intrusão por ataques do tipo *probing* com classificação por Árvore de Decisão realizada em software e em hardware.

1.2.2 OBJETIVOS ESPECÍFICOS

- Desenvolvimento de uma plataforma de medição de consumo energético de aplicações executadas em um processador Atom N2800 e/ou em hardware através de uma FPGA Cyclone IV Gx.
- Desenvolvimento de uma interface PCIe Gen 1 x1 entre o processador Atom e a FPGA, de forma a permitir que a FPGA seja utilizada como coprocessador.
- Avaliar o desempenho da plataforma desenvolvida e situar a mesma perante o estado da arte.

- Apresentar a utilização da plataforma para a análise de uma aplicação para detecção de intrusão por ataque do tipo *probing* com classificação em software ou hardware através de um algoritmo de *Árvore de decisão*.

1.3 ESTRUTURA DA DISSERTAÇÃO

Este trabalho está organizado em seis capítulos. O capítulo 2 analisa os conceitos envolvidos na medição do consumo energético de aplicações em software e aplicações em hardware e discutem-se as plataformas de medição disponíveis na literatura. O capítulo 3 descreve o desenvolvimento da plataforma, o seu método de utilização e validação. Os conhecimentos básicos pertinentes à aplicação para detecção de intrusão analisada, bem como o seu desenvolvimento, são apresentados no capítulo 4. No capítulo 5 são mostrados e discutidos os resultados obtidos, consistindo nas características da medida proporcionada pela plataforma e na análise do consumo energético da aplicação para detecção de intrusão. Por fim, o capítulo 6 contempla as conclusões finais, publicações e propostas de trabalhos futuros.

2 REVISÃO DE LITERATURA

Este capítulo aborda os conceitos envolvidos na medição do consumo energético de aplicações em software e de hardwares em FPGA. Primeiramente, são discutidos os métodos utilizados nos trabalhos presentes na literatura que se propuseram a comparar o consumo entre software e hardware. Na sequência, são analisadas separadamente as medições de softwares e de hardwares, pois grande parte das informações disponíveis aborda individualmente estes cenários.

2.1 MÉTODOS DE COMPARAÇÃO DE CONSUMO: SOFTWARE VS. HARDWARE

A comparação entre o comportamento de implementações equivalentes em software e em hardware é uma prática largamente adotada quando se busca a melhoria do desempenho de algoritmos complexos. Quando a pesquisa procura atender sistemas com restrições energéticas, como dispositivos móveis ou sensoriamento distribuído, o consumo torna-se o principal motivador da migração do processamento para o hardware. Nestes casos, duas técnicas são utilizadas para analisar o consumo energético.

A primeira técnica se utiliza de um componente de lógica reprogramável (FPGA) capaz de embarcar um microprocessador onde é executada a implementação em software. Posteriormente, o mesmo componente configurado com a implementação em hardware é analisado. A vantagem desta abordagem reside no fato de se ter um ambiente de hardware único, no qual se utiliza o mesmo método de medição para ambas as implementações. Desta forma, uma comparação direta entre os resultados é possível (MEINTANIS; PAPAEFSTATHIOU, 2009; BRAGA et al., 2010). Contudo, o consumo em software não reproduz a realidade quando a aplicação em análise originalmente não utiliza uma FPGA para o processamento. É o caso dos dispositivos móveis, que em sua maioria utilizam processadores com arquitetura x86 ou ARM.

A segunda técnica utiliza dois ambientes distintos, um para a implementação em software e outro para a implementação em hardware. O software é executado no processador normalmente utilizado na aplicação em análise e o hardware é implementado em um segundo

componente, sendo comum o uso de FPGAs. O uso de dois ambientes exige que os métodos de medição utilizados para o software e para o hardware sejam diferentes, tornando mais complexa a instrumentação. Este fato é agravado quando o software está em execução sobre um sistema operacional. Em contrapartida, a vantagem desta abordagem é reproduzir com fidelidade o consumo em condições de uso para ambas as implementações, podendo assim analisar o ganho real das melhorias estudadas. Além disto, este método possibilita a análise de soluções com processamento híbrido, em que parte do processamento se mantém em software sendo auxiliado por um coprocessador externo para o processamento em hardware.

Apesar do uso de ambientes distintos para cada uma das implementações ser a técnica mais utilizada pelas pesquisas que se propuseram a comparar software e hardware, em todos os trabalhos encontrados na literatura, a medição do consumo do software é realizada de forma simplista, não considerando fatores primordiais como a política de escalonamento de tarefas realizada pelo sistema operacional e o compartilhamento dos recursos de hardware por outros programas que estão em execução (KESTUR et al., 2010; ZOU et al., 2012; SCHMADECKE; BLUME, 2013). O método de medição adotado nestes trabalhos atribui ao software em análise o consumo total do sistema durante todo o tempo de execução, o que pode incorrer em erros que afetam diretamente o resultado, pois processos concorrentemente ativos e que demandam maior, ou menor, consumo energético que a aplicação medida são erroneamente contabilizados.

Procurando evoluir o método de obtenção do perfil energético de sistemas com processamento híbrido composto por processador x86, Unidade de Processamento Gráfico (GPU - *Graphics Processing Unit*) e FPGA, (TSOI; LUK, 2011) desenvolveu um procedimento de medição para obter o consumo de aplicações que se utilizem destes componentes. Neste método, todos os processadores estão integrados em um mesmo equipamento, conforme arquitetura mostrada na figura 4. Para medir o consumo, um alicate amperímetro (FLUKE i30) é adicionado à alimentação CA do sistema e, com o auxílio de um multímetro (Maplin N56FU), a corrente é amostrada com taxa de 10 Hz. Através de uma interface USB, as medidas são enviadas para o sistema em análise que, por sua vez, armazena cada amostra juntamente com a estampa de tempo do momento de seu recebimento. Na tentativa de segregar apenas a alteração de consumo provocada pela aplicação analisada, o valor medido durante a execução da aplicação é subtraído do consumo médio do sistema quando executando apenas o sistema operacional (inativo). As estampas de tempo do início e do fim da execução da aplicação são armazenadas e posteriormente comparadas com as estampas de tempo das amostras de consumo. A energia consumida é determinada pela integração das amostras de consumo compreendidas no tempo de execução da aplicação em análise.

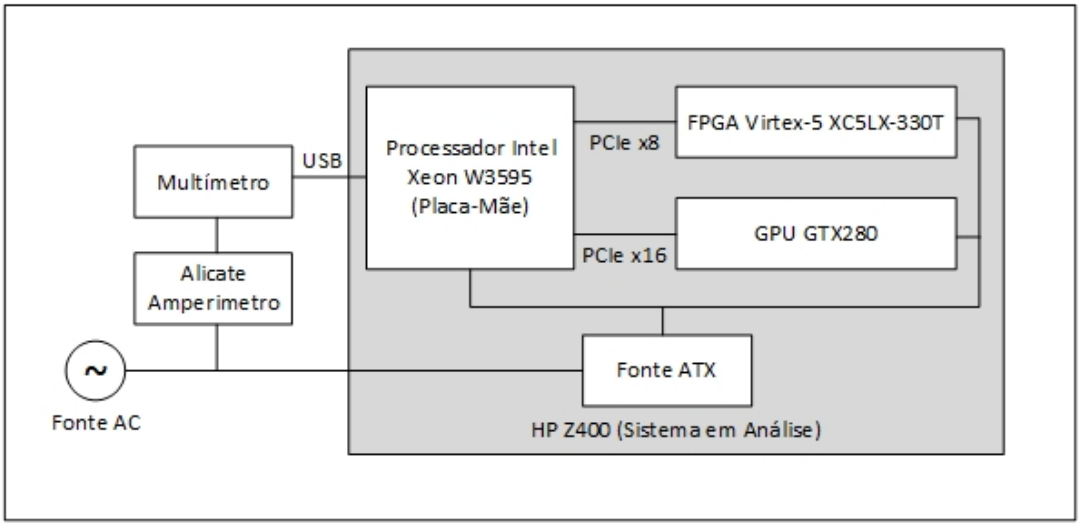


Figura 4: Arquitetura de hardware da plataforma de medição de consumo desenvolvida em (TSOI; LUK, 2011) para análise de sistemas híbrido compostos por processador x86, FPGA e GPU.

Fonte: Adaptada de (TSOI; LUK, 2011).

As melhorias propostas por este procedimento de medição não atendem às seguintes questões essenciais para a comparação do consumo entre software e hardware: a medição do consumo de aplicações em software não considera a atividade do escalonador do sistema operacional, mantendo a mesma restrição dos demais trabalhos; o consumo da linha de alimentação AC não reproduz o consumo instantâneo do sistema, uma vez que os capacitores da fonte defasam a corrente consumida em relação à demanda do sistema; devido ao elevado consumo do sistema quando inativo, o aumento causado pela aplicação em análise representa uma pequena proporção do consumo total, reduzindo a resolução da medida; e o método é incapaz de segregar a parcela de consumo referente a cada processador no caso de implementações com processamento híbrido.

Apesar das restrições encontradas no único procedimento disponível dedicado à análise de consumo de sistemas com processamento híbrido, não há estudos posteriores que procuraram aprimorar a técnica para a comparação de implementações em software e hardware. Perante isto, as informações em relação à medição de consumo que se aplicam ao presente trabalho encontram-se em dois ramos distintos: medição do consumo energético de aplicações em software e medição de consumo energético de hardware em FPGAs.

2.2 MEDIÇÃO DE CONSUMO EM SOFTWARE

2.2.1 CARACTERÍSTICAS DO PROCESSAMENTO EM SOFTWARE

Os sistemas computacionais largamente utilizados são baseados em duas arquiteturas de processadores: x86 e ARM. Estas arquiteturas estão presentes em quase todos os computadores pessoais, *smartphones*, *tablets*, além de muitos outros dispositivos móveis. O funcionamento destes equipamentos é baseado em um sistema operacional, do qual o programa mais importante é denominado *kernel*. O *kernel* é carregado na memória RAM quando o sistema é iniciado e contém procedimentos essenciais para o funcionamento do equipamento, além de determinar as atividades do hardware e controlar a execução dos demais programas do sistema (BOVET; CESATI, 2006).

Um sistema operacional deve atender dois principais objetivos:

- interagir com os componentes do hardware, provendo todos os programas de "baixo nível" para o funcionamento da plataforma;
- prover um ambiente de execução para a aplicação que será utilizada no sistema.

O termo "aplicação" será utilizado para denotar o programa ou conjunto de programas executados pelo usuário para realizar a tarefa pretendida. Desta forma, os sistemas operacionais modernos abstraem todos os detalhes em relação à organização física do computador para a aplicação. Quando uma aplicação pretende usar um recurso do hardware, esta deve requisitar ao sistema operacional, a partir do que o *kernel* avalia a requisição e interage com o hardware conforme pretendido pela aplicação.

Um conceito fundamental utilizado pelos sistemas operacionais é a definição de processo. Um processo é uma instância de um programa em execução. Para manter as funcionalidades do sistema, múltiplos processos concorrentes são mantidos ativos (BOVET; CESATI, 2006).

Apenas um processo tem o controle de um núcleo do processador (CPU - *Central Processing Unit*) em um dado momento; desta forma, somente um fluxo de execução é processado por vez. Uma vez que o número de CPUs é sempre restrito, apenas poucos processos podem ser executados simultaneamente. Para atender todos os processos ativos, o sistema operacional possui um componente denominado escalonador. Este componente é responsável por determinar qual processo tomará controle da CPU em um dado momento. A ação do escalonador de retirar um processo da CPU e atribuir a ela um segundo processo é denominada

troca de contexto.

Alguns sistemas operacionais de propósito específico permitem processos não preemptivos. Neste caso, o escalonador é chamado apenas quando o processo voluntariamente libera a CPU. Contudo, os sistemas operacionais utilizados na grande maioria dos dispositivos são preemptivos, nos quais o escalonador monitora os processos ativos e periodicamente interrompe os processos em execução nas CPUs para que estes sejam substituídos. No escalonamento preemptivo, o processo não tem controle sobre o momento em que será interrompido (BOVET; CESATI, 2006). O Windows e o Linux são exemplos de sistemas operacionais com escalonamento preemptivo.

Com o advento de novas tecnologias de paralelização do processamento, como processos que utilizam mais de uma CPU, criou-se o conceito de *thread*. Um processo pode ser visto como uma *thread* única, porém um processo pode conter múltiplas *threads* que compartilham recursos entre si (código e/ou dados). Processos e *threads* são tratados da mesma forma pelo escalonador (JONES, 2009).

Em sistemas operacionais com escalonamento preemptivo, para que um único processo seja analisado, é necessário considerar que vários outros processos estão concorrentemente ativos e compartilhando os recursos do hardware. Com isto, o monitoramento da atividade do escalonador é um requisito necessário para que uma plataforma de medição seja capaz de avaliar o consumo de uma aplicação. Uma vez que o acesso a este componente está disponível apenas no *kernel*, o desenvolvimento de uma plataforma de medição exige a adoção de um sistema operacional que possua código fonte disponível, permitindo assim a análise e instrumentação do escalonador. Dentre os sistemas que se enquadram neste requisito, o Linux é a melhor opção, pois é largamente utilizado, inclusive por dispositivos móveis.

2.2.1.1 ESCALONAMENTO DE TAREFAS NO LINUX

O problema básico do escalonamento em sistemas operacionais é satisfazer os seguintes requisitos conflitantes: proporcionar um baixo tempo de resposta, conciliar processos de alta e baixa prioridade, prover um bom *throughput* para os processos e evitar postergação indefinida. O conjunto de regras que estabelece quando e por quanto tempo um processo será executado na CPU é denominado política de escalonamento (OLIVEIRA et al., 2008).

O escalonador do Linux divide o tempo do processador em fatias denominadas *quantum*, nas quais são alocados os processos. Esta técnica é chamada *time-sharing*. Se durante a execução de um processo o *quantum* é esgotado, uma troca de contexto é realizada

(OLIVEIRA et al., 2008).

A versão 2.6.23 do *kernel* do Linux, disponibilizada em outubro de 2007, introduziu um novo conceito de escalonamento denominado *Completely Fair Scheduler* (CFS), utilizado desde então. O principal objetivo da política de escalonamento adotada pelo CFS é manter equilibrada a execução dos processos através de uma distribuição "justa" do tempo do processador. Para determinar o equilíbrio, o CFS armazena o tempo total de CPU já atribuído a cada processo, denominado *virtual runtime*. Processos com menores *virtual runtime* indicam que uma menor quantidade de tempo do processador foi atribuída a eles, então maior é a necessidade de processamento. Para manter o equilíbrio entre as tarefas de forma eficiente, o CFS utiliza uma árvore *Red-Black* ordenada pelo *virtual runtime* (JONES, 2009; NOVELL INC., 2011).

Estruturas de dados armazenadas em árvores *Red-Black* possuem como principal característica o fato de serem auto-balanceadas. A figura 5 mostra um exemplo da estrutura de dados utilizada pelo escalonador CFS. Processos com menores *virtual runtime* são armazenados à esquerda, enquanto os processos com maior *virtual runtime* são armazenados à direita. Com isto, o processo que mais precisa do processador é o nó mais à esquerda da árvore; consequentemente, este será o próximo a ser alocado. Ao retirar um processo da CPU, o escalonador atualiza o respectivo *virtual runtime* e o insere novamente na árvore; desta forma, os processos do lado esquerdo da árvore ganham tempo de processamento e o conteúdo da árvore migra da direita para a esquerda de forma a manter o equilíbrio (JONES, 2009).

Para auxiliar a identificação dos elementos manipulados pelo escalonador, bem como pelos demais componentes do *kernel*, cada processo e *thread* possui um identificador numérico único, denominado *Process ID* (PID). Um segundo identificador numérico, denominado *Thread Group Id* (TGID), compartilhado entre todas as *threads* de uma mesma aplicação, também é adicionado a cada elemento.

2.2.2 TÉCNICAS DE ANÁLISE DO CONSUMO DE SOFTWARES

As técnicas para análise do consumo de aplicações em software podem ser classificadas em três grupos: simulação, simulação baseada em medição e medição direta. As simulações baseiam-se em modelos capazes de estimar o consumo através da análise das instruções que compõem a aplicação. Contudo, os simuladores exigem grande esforço para sua construção e são específicos para determinado hardware. Este método se tornaria impeditivamente complexo para aplicações com processamento híbrido, pois nestes casos o consumo depende da interação entre componentes distintos.

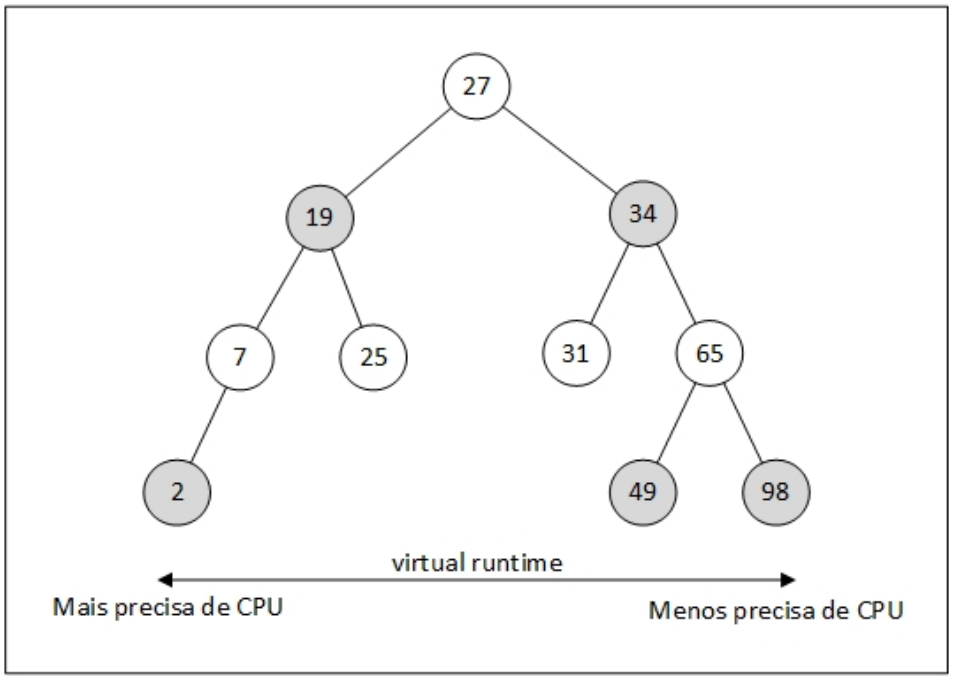


Figura 5: Exemplo de árvore *Red-Black* utilizada pelo CFS. Cada nó representa um processo, cujo índice indica o *virtual runtime*.

Fonte: Adaptada de (JONES, 2009).

De forma similar, as simulações baseadas em medição associam um consumo energético a cada instrução do processador. Contudo, os valores de consumo são obtidos através da medição da energia demandada para cada instrução realizada pelo processador, para a interinstrução e para os demais consumos ligados ao processamento, como a memória *cache*. Este método demanda complicada instrumentação da placa, além de sofrer das mesmas desvantagens do método anterior, puramente por simulações (XIAN et al., 2007).

A medição direta, por sua vez, exige a instrumentação do hardware para a amostragem do consumo e um método capaz de analisar separadamente a aplicação a ser medida, discriminando-a dos demais processos. Apesar disto, uma vez estabelecida a infraestrutura necessária, a medição pode ser facilmente reproduzida para diferentes cenários. Também é importante notar que diversas aplicações requerem uma análise realizada em tempo de execução, pois características críticas para determinar o consumo são conhecidas apenas durante o processamento, como, por exemplo, a natureza dinâmica dos eventos externos. Devido a estes fatores, a técnica a ser analisada em detalhe é a medição direta do consumo de aplicações em software.

2.2.3 ESTADO DA ARTE EM MEDIÇÃO DO CONSUMO DE SOFTWARES

A necessidade de coletar informações do consumo energético de aplicações em software foi suprida parcialmente em trabalhos anteriores através de duas abordagens: medições baseadas em instrumentações inseridas nos processadores pelos fabricantes, denominadas *On-Chip*, ou plataformas de medição de consumo desenvolvidas pela comunidade científica. Contudo, quatro maiores fraquezas permanecem no estado da arte da medição de consumo de aplicações em software: ausência de sincronismo entre a amostra do consumo e a aplicação medida, incapacidade de isolar a aplicação medida de outros processos em execução, incapacidade de medir com precisão o tempo de processamento e incapacidade de segregar um trecho específico de código da aplicação em análise.

2.2.3.1 MEDIÇÕES *ON-CHIP*

As medições de consumo baseadas em informações providas pelo próprio processador têm atingido exatidões aceitáveis, além de serem suportadas pelos atuais processadores x86 (GOEL et al., 2010). Destaca-se como vantagem deste método a obtenção do consumo sem a necessidade de medidores externos e alterações no hardware do sistema.

A arquitetura dos atuais processadores de alto desempenho da Intel, denominada *Sandy Bridge*, é capaz de estimar o consumo energético do componente e adaptar seu comportamento de forma a otimizar seu desempenho. O acesso ao consumo é disponibilizado ao sistema operacional por uma interface denominada RAPL (*Running Average Power Limiting*). A interface RAPL permite obter a energia consumida por módulos do processador em uma janela de tempo definida. Estas informações são disponibilizadas através de registradores específicos para cada módulo. Os módulos disponíveis variam conforme o modelo do processador; processadores de propósito geral disponibilizam acesso ao consumo total do encapsulamento, dos núcleos e da GPU, enquanto processadores para servidores possuem acesso ao encapsulamento, núcleos e DRAM (HACKENBERG et al., 2013). Os registradores que armazenam os contadores de consumo são atualizado a cada 976 μs. A granularidade da informação de consumo é de 15,3 μJ; considerando a taxa de atualização, tem-se uma resolução em potência de 14,9 mW (INTEL CORP., 2011).

Quando informações providas pela RAPL são utilizadas, é necessário considerar que estas não são resultados de uma medição física, mas sim aproximações baseadas em modelos que utilizam contadores de eventos do hardware combinados com pesos predefinidos conforme a natureza de cada evento. Outro fato a ser considerado é que a RAPL informa o total de energia

consumida desde a última leitura; não são informados valores de potência, nem atribuídas estampas de tempo. Desta forma, é necessário aguardar longos períodos para se adquirir a potência média, pois, no caso de uma amostragem de 10 ms, por exemplo, o resultado poderia acarretar em uma imprecisão de até 10%, uma vez que neste período pode haver de 9 a 11 atualizações (INTEL CORP., 2011).

A AMD, a partir da microarquitetura 15h, também introduziu a capacidade de adaptar o desempenho de processadores baseando-se na estimativa da potência dissipada, denominada APM (*Aplication Power Management*). Neste método, o consumo é obtido aplicando os valores dos registradores de desempenho presentes na *Northbridge* em modelos predefinidos. A taxa de atualização típica da informação de consumo é realizada a cada 10 ms e refere-se à potência média durante o último intervalo de tempo (ADVANCED MICRO DEVICES INC., 2013). A resolução do valor da potência no caso do processador AMD Opteron 6274, por exemplo, é de 3,8 mW (HACKENBERG et al., 2013).

A abordagem da AMD de utilizar o valor da potência média no último intervalo de tempo, ao invés da energia acumulada desde a última leitura, tem a desvantagem de possuir a informação de consumo apenas dos últimos 10 ms, necessitando a sua leitura constante. Apesar disto, para pequenos intervalos de tempo, possui uma exatidão consideravelmente maior que a solução disponibilizada pela Intel.

A utilização das medições *On-Chip* para análises de consumo de trechos de código, caso em que se exige taxa de amostragem do consumo igual ou superior a 100 sps, foi analisada em (HACKENBERG et al., 2013); sendo determinado que a utilização do RAPL é inviável, pois taxas de leitura superiores a 20 sps resultam em erros maiores que 5%. Nesse mesmo trabalho, também foi encontrada grande imprecisão na potência medida através do APM, particularmente quando o processador está em estados de baixo consumo, sendo inviável para aplicações em que maior exatidão seja requerida.

2.2.3.2 POWERSCOPE

Uma das primeiras plataformas a medir o consumo energético de aplicações em software foi o PowerScope. Esta ferramenta é capaz de traçar um perfil do consumo energético das aplicações que estão em execução em um laptop com sistema operacional NetBSD. A figura 6 mostra o diagrama em blocos desta plataforma. O ambiente de medição é constituído por três principais componentes: o computador em análise (Laptop IBM 701C), um computador para coleta dos dados e um multímetro de bancada (Hewlett Packard 3458a) (FLINN; SATYANARAYANAN, 1999).

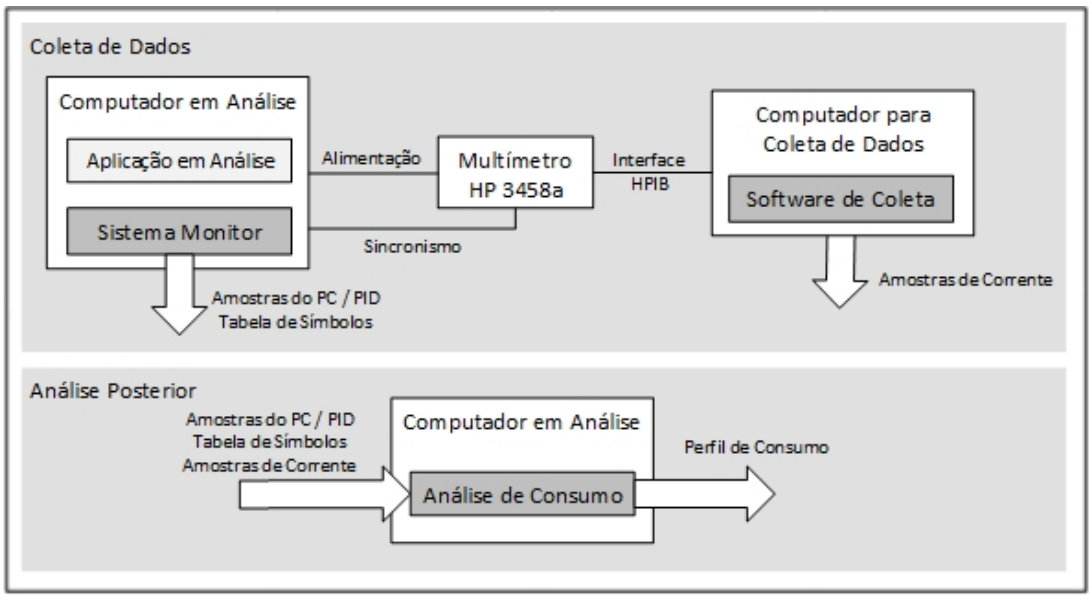


Figura 6: Diagrama em blocos da plataforma de medição de consumo PowerScope. Fonte: Adaptada de (FLINN; SATYANARAYANAN, 1999).

O processo de medição no PowerScope possui duas etapas: coleta de dados e análise posterior. Durante a coleta de dados, o multímetro é utilizado para obter a corrente consumida pelo computador em análise. A bateria do notebook é retirada para evitar drenar corrente desnecessária ao processamento. Dois sinais externos providos pelo multímetro são ligados à porta paralela do computador em análise com o objetivo de sincronizar as medidas realizadas. Um sinal é utilizado para disparar uma nova medida do multímetro, enquanto o outro sinal é utilizado para indicar o momento em que a medida ocorreu. As medidas de consumo realizadas pelo multímetro são lidas pelo segundo computador.

Dois componentes de software controlam a etapa de coleta de dados: Sistema Monitor e Software de Coleta. O Sistema Monitor é responsável por amostrar o *Program Counter* (PC) e o PID do processo em execução no momento em que o sinal de sincronismo proveniente do multímetro é recebido. Na sequência, uma nova medida é disparada. O Software de Coleta é responsável por armazenar as medidas realizadas pelo multímetro. Uma interface de programação (API - *Application Programming Interface*) é disponibilizada para que a aplicação em análise inicie e termine o processo de aquisição de dados, permitindo assim medir trechos específicos de código.

Na etapa seguinte, necessariamente executada no computador em análise para que se tenha acesso à tabela de símbolos do sistema operacional, os dados coletados são computados a fim de se obter os resultados da medição. Durante esta etapa, cada amostra de consumo é

atribuída à aplicação referenciada pelo PID armazenado no momento de sua obtenção. Através do PC, também é possível indicar qual parte da aplicação estava em execução no momento da medição. Desta forma, o consumo total de uma aplicação é determinado aplicando as amostras atribuídas a ela na fórmula 1.

$$E \approx V_{nominal} \sum_{t=0}^{n} I_t . \Delta t \tag{1}$$

em que $V_{nominal}$ é a tensão nominal da alimentação do computador em análise, $I(t)$ é a corrente amostrada, Δt é o período de amostragem (1.6 ms) e n é o número de amostras que ocorreram enquanto o processo medido estava em execução.

O PowerScope é capaz de informar o perfil de consumo das aplicações em execução no computador em análise, bem como discriminar o consumo de uma única aplicação. Entretanto, possui as seguintes limitações: a resolução da medição do tempo de processamento de uma tarefa depende da taxa de amostragem do multímetro, sendo assim incapaz de medir trechos curtos de código; insere relevante *overhead*, uma vez que cada amostra de consumo gera uma interrupção no computador em análise; baseia-se em um sistema operacional pouco utilizado atualmente (NetBSD); e depende de dois computadores para realizar as medições.

2.2.3.3 POWERPACK

PowerPack é uma plataforma de medição que objetiva a análise do consumo de computadores de alta performance (HPC - *High-Performance Computing*), também denominados *clusters* ou supercomputadores. Estes sistemas são constituídos por um conjunto de computadores ligados em rede de forma a resultar em um maior poder de processamento. O PowerPack tem como objetivo fornecer o consumo dos diferentes componentes de um HPC e sincronizar as medições com trechos de código de uma aplicação.

A plataforma suporta um conjunto de medidores de consumo externos, como dispositivos para aquisição de dados da National Instrument e fontes de alimentação ATX capazes de fornecer o consumo instantâneo do sistema (GE et al., 2010). A figura 7 mostra o diagrama em blocos do PowerPack. Um computador do *cluster* é designado para realizar a coleta de dados, sendo todos os medidores ligados a ele. Neste computador, uma aplicação em software é responsável por controlar a medição, receber as informações de cada medidor, realizar operações matemáticas para converter o valor medido em potência consumida e registrar os valores de consumo para posterior análise. Este software também disponibiliza uma API para que a aplicação em análise indique o momento em que se deve iniciar e parar

a coleta de medidas. Durante este período, todas as medições de consumo são atribuídas à aplicação.

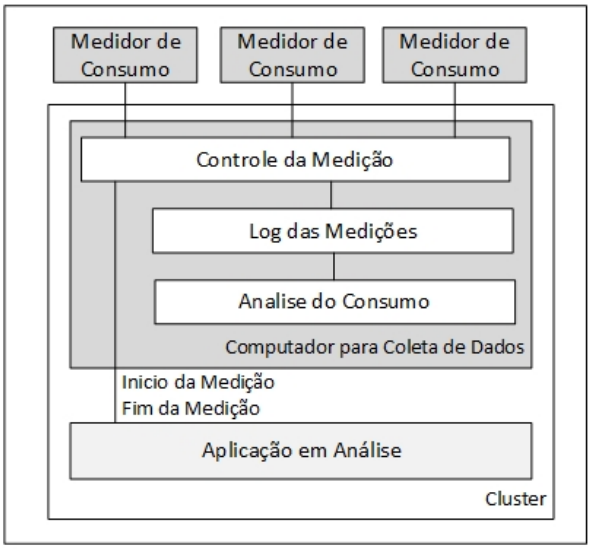


Figura 7: Diagrama em blocos da plataforma de medição de consumo PowerPack.
Fonte: Adaptada de (GE et al., 2010).

Após a execução da aplicação em análise e a realização da coleta de dados, o software da plataforma compila as medições obtidas e gera o resultado da medição. Apesar do PowerPack ser capaz de medir trechos de código, não leva em conta que outras aplicações compartilham o processamento, pois considera que o consumo durante todo o tempo de execução foi causado pela aplicação em análise.

2.2.3.4 MEDIÇÃO DO CONSUMO ATRAVÉS DE PERFORMANCE API

Performance API (PAPI) é um *framework* multiplataforma largamente utilizado para fornecer acesso de "baixo nível" aos registradores de performance disponíveis nos computadores modernos (BROWNE et al., 2000). O principal objetivo do PAPI é disponibilizar, em tempo real, informações da utilização dos recursos de hardware para uma aplicação em execução. Desta forma, a aplicação poderia tomar decisões para manter seu bom desempenho, mesmo perante a escassez de certo recurso. Esta API foi expandida para suportar medidores de energia externo e leitura da informação de consumo disponibilizada por certas arquiteturas de computadores (WEAVER et al., 2012).

O medidor comercial Watt's Up Pro e a ferramenta de medição PowerMon 2 foram integrados ao PAPI, assim como as informações de consumo providas pelo RAPL e APM. As medições fornecidas por estas ferramentas possuem período de amostragem entre alguns

milissegundos a um segundo, porém a requisição de informações provinda das aplicações pode alcançar taxas bem superiores, o que exige a leitura periódica dos medidores e armazenamento do último valor medido para que seja enviado quando solicitado.

O PAPI possui como vantagem a capacidade de fornecer informações de consumo à aplicação em análise, permitindo assim que ações para redução do consumo energético sejam executadas em tempo real. Outro diferencial do PAPI é a padronização do acesso à informação para diferentes plataformas de hardwares, permitindo que uma aplicação instrumentada seja analisada em diferentes cenários sem a necessidade de customização para cada caso. Uma desvantagem é que a arquitetura do PAPI exige que a aquisição de dados seja realizada na mesma plataforma em que a aplicação é executada, inserindo na medição um indesejável *overhead*.

2.2.3.5 POWERMON

PowerMon e PowerMon 2 são ferramentas desenvolvidas para serem integradas a computadores convencionais visando analisar as características do consumo de aplicações específicas. Inserida entre a fonte de alimentação ATX e a placa-mãe, o PoweMon monitora a tensão e a corrente fornecidas em cada uma das seis linhas CC. Os dados coletados são disponibilizados através de uma interface USB. Para facilitar a fixação mecânica, foi utilizado o formato de disco rígido (HD) de 3,5"para acomodar o hardware desenvolvido. A figura 8 mostra as imagens destas ferramentas (BEDARD et al., 2010).

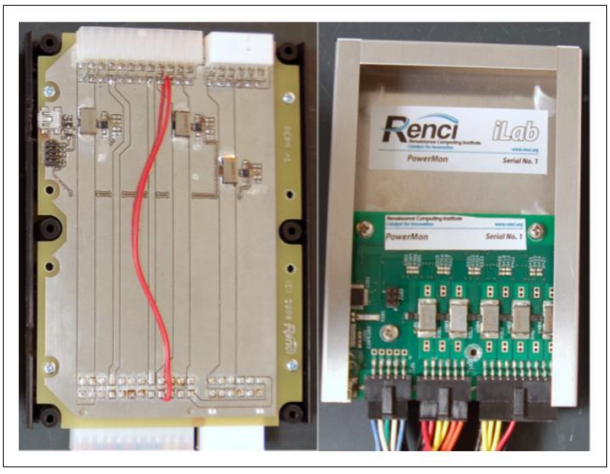


Figura 8: PowerMon, à esquerda, e PowerMon2, à direita.
Fonte: (BEDARD et al., 2010).

O hardware desenvolvido para o PowerMon alcança taxa de amostragem de

aproximadamente 50 sps, enquanto que o PowerMon 2 é capaz de atingir até 3 ksps compartilhados entre os canais utilizados, sendo limitado em 1 kHz a taxa máxima amostrada por canal. O sensoriamento da corrente é realizado através de um resistor de 5 mΩ em série com cada linha de alimentação, sendo capaz de medir correntes de até 10 A. Além da amostragem dos seis canais da fonte ATX, o PowerMon 2 possui duas entradas adicionais para que sejam medidos periféricos específicos, podendo assim monitorar o consumo de HDs e GPUs. Uma estampa de tempo é anexada a cada amostra de consumo com o objetivo de permitir o posterior sincronismo com o período de execução da aplicação em análise.

O software disponibilizado com esta ferramenta permite a leitura e armazenamento das medidas realizadas, além de fornecer em tempo real a última medida obtida, permitindo assim que a aplicação em análise receba a informação de seu próprio consumo. Apesar de fornecer informação de tempo relativo a cada medida, não é possível determinar quais medidas efetivamente ocorreram durante o processamento da aplicação em análise em um ambiente multitarefa, pois não é considerada a atividade do escalonador.

2.2.3.6 POWERINSIGHT

O PowerInsight é uma plataforma de medição de consumo desenvolvida para atender a demanda de medição de consumo no desenvolvimento de software para HPCs. Assim como o PowerMon, o medidor é inserido entre a fonte de alimentação ATX e a placa-mãe, entretanto é capaz de medir correntes de até 30 A por canal. Outro avanço no sensoriamento da corrente realizado pelo PowerInsight é a utilização de sensores de efeito Hall ao invés de resistores em série com a alimentação, reduzindo assim o impacto da inserção do sensor no consumo do sistema (LAROS et al., 2013).

A arquitetura concebida para o PowerInsight permite a criação de uma rede de sensores, sendo que uma unidade do PowerInsight é inserida em cada computador do HPC. Através de uma interface Ethernet é possível compilar as informações dos sensores, obtendo assim o consumo total do *cluster* para a aplicação em análise. A figura 9 mostra o diagrama em blocos da rede de sensores criada através do PowerInsight.

2.2.3.7 MEDIÇÃO CONSIDERANDO ACESSOS A I/O

A plataforma de medição que procurou retratar com maior exatidão o consumo energético de aplicações em software foi desenvolvida em (XIAN et al., 2007). Nela é considerado não apenas o tempo de processamento na CPU como parâmetro de sincronismo

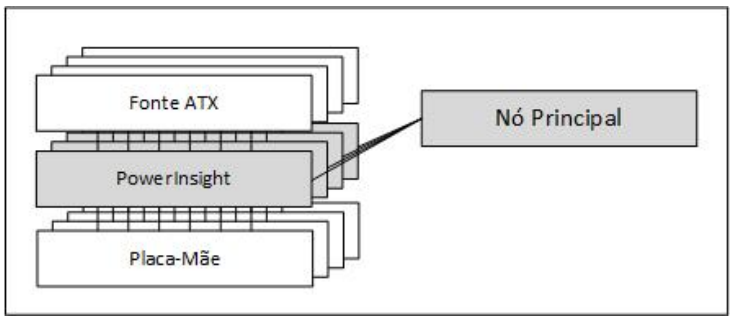


Figura 9: Diagramas em blocos de uma rede de sensores criada com a plataforma PowerInsight. Fonte: (LAROS et al., 2013).

para a medição, mas também os tempos de acesso ao HD e à placa de rede.

A arquitetura de hardware da plataforma é mostrada na figura 10. O computador onde se executada a aplicação em análise é instrumentado através de uma ferramenta para aquisição de dados analógicos (DAQ - *Data Aquisition Tool*) da National Instrument, de forma a obter o consumo individual do processador, do HD e da placa de rede. A taxa de amostragem utilizada é de 50 ksps com 16 bits de resolução. Para reduzir o *overhead* causado pela medição, um segundo computador é utilizado para a leitura das amostras obtidas pelo DAQ e processamento posterior dos dados. Uma comunicação serial, ou de rede, entre os dois computadores é utilizada para a transferência dos dados e negociação do sincronismo.

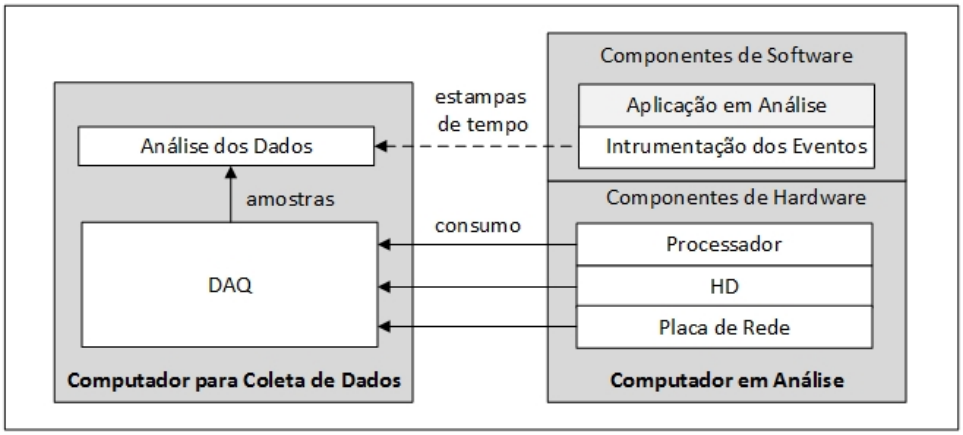


Figura 10: Diagrama em blocos da plataforma de medição criada por Xien et al. Fonte: Adaptada de (XIAN et al., 2007).

As estampas de tempo dos eventos que indicam o uso da CPU, do HD e da placa de rede pela aplicação em análise são obtidas durante a etapa de coleta de dados, sendo utilizada para posterior processamento e sincronismo com as amostras de consumo realizadas pelo DAQ. Para permitir acesso a estas informações, o código do *kernel* do sistema operacional do computador

medido (Linux 2.4.18) é instrumentado.

A determinação do uso da CPU é realizada através do monitoramento da atividade do escalonador. Cada vez que um processo é alocado ou desalocado do processador, a estampa de tempo é armazenada juntamente com o PID do processo.

Quando o HD é solicitado por uma aplicação, o *kernel* envia um comando ao mesmo e recebe uma interrupção quando o comando é finalizado. Armazenando as estampa de tempo destes eventos é possível determinar o período em que o HD foi acessado. Contudo, o *kernel* utiliza um *buffer* para acesso as memórias de massa externas, sendo que a transferência dos dados do *buffer* para o HD é sempre realizada pelo módulo do *kernel* denominado *kupdate*. Para permitir o rastreio do momento que a parte do *buffer* pertencente à aplicação em análise foi gravada no HD, é incluído o PID do processo à estrutura de dados manipulada pelo *kupdate*. É importante notar que este método não considera o *buffer* interno do HD, o que causa uma perda de sincronismo entre os registros de acesso ao HD e o efetivo consumo causado, pois a etapa de maior gasto energético é a escrita ou leitura do disco rígido e não a comunicação.

A placa de rede é tratada de forma similar pelo *kernel*, porém o início da operação de recebimento dos dados é abstraído pela camada de hardware. A interrupção que indica a chegada de um pacote é enviada para o *kernel* apenas após o recebimento completo do pacote; consequentemente, ao receber a interrupção, o consumo já ocorreu. Para tratar esta questão, o tempo para o recebimento de um pacote é medido. Através de um segundo computador, uma sequência de pacotes UDP é enviada para o computador analisado; também é garantido que os pacotes possuem o mesmo tamanho, são enviados em sequência e ocupem toda a banda disponível. Desta forma, o intervalo entre duas interrupções recebidas pelo *kernel* indica o tempo necessário para a chagada dos pacotes.

A estampa de tempo do início do recebimento dos dados da rede é então determinada de forma retroativa em relação ao recebimento da interrupção, levando em conta o tamanho do pacote. Com isto, é possível encontrar as amostras de consumo ocorridas durante o recebimento dos pacotes. Para atribuir o consumo a determinada aplicação, é considerado o PID do processo que abriu o *socket* utilizado na comunicação.

Outro fato considerado na medição da placa de rede é a capacidade de enviar e receber dados simultaneamente (*full-duplex*), sendo que o consumo medido não necessariamente corresponde a só uma das suas funções. São então medidos os consumos da placa quando utilizada apenas para o envio de dados e quando utilizada apenas para o recebimento. Para isto, é realizado um procedimento semelhante ao utilizado para determinar o tempo de recebimento de um pacote. A partir das informações obtidas, os fatores α_t e α_r foram calculados através das

fórmulas 2 e 3. Estes fatores são utilizados para ponderar o valor do consumo da placa de rede quando apenas uma das suas funções é utilizada pela aplicação em análise.

$$\alpha_t = \frac{p_t}{p_t + p_r} \tag{2}$$

$$\alpha_r = \frac{p_r}{p_t + p_r} \tag{3}$$

em que α_t e α_r são os fatores de correção para o consumo relativo apenas à transmissão e ao recebimento de dados, respectivamente, p_t é o consumo medido para a placa de rede apenas transmitindo dados e p_r é o consumo medido para a placa de rede apenas recebendo dados.

As estampas de tempo obtidas no computador analisado são baseadas na função *do_gettimeofday* disponibilizada pelo *kernel* do linux e que fornece uma base de tempo com resolução de micro segundos. Para determinar se uma amostra corresponde a um dado processo, é necessário sincronizar o momento das capturas das amostras de consumo com a base de tempo das estampas obtidas do *kernel*. Foi então criado um método de sincronismo denominado M-sync, mostrado na figura 11 e descrito nos seguintes passos:

1. Através da interface de comunicação, os dois computadores acordam qual componente será utilizado para o sincronismo, normalmente o HD. O computador analisado coloca este componente em estado de baixo consumo (*sleep*) e envia a mensagem *sync_start* para o computador de monitoramento.

2. O computador de monitoramento inicia a amostragem do consumo e envia a mensagem *sync_ready* para o computador analisado. A amostragem do consumo é mantida, sendo seus valores armazenados durante a sincronização.

3. O computador analisado ativa (*wakeup*) o componente, consequentemente seu consumo aumenta drasticamente. A estampa de tempo do momento da ativação (t_c) é enviada ao computador de monitoramento.

4. O computador analisado envia a mensagem *sync_done* para o computador de monitoramento. O computador de monitoramento procura dentre as amostras armazenadas qual identificou o aumento de consumo s_c e atribui a esta medida a estampa de tempo da ativação do componente (t_c).

5. O método de sincronismo é repetido a cada duas horas para se evitar o desvio entre as bases de tempo.

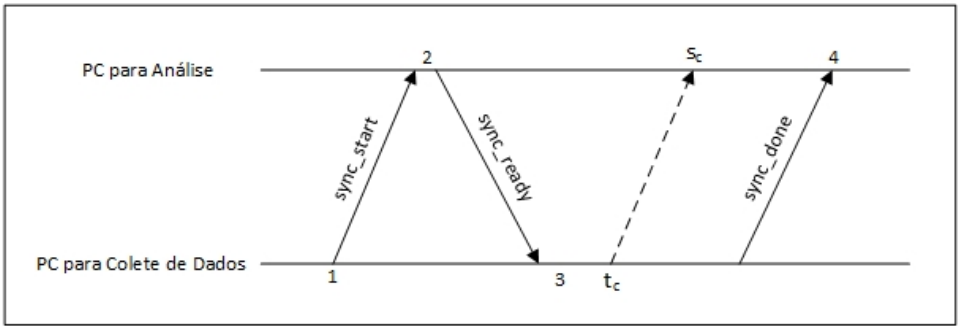


Figura 11: Método de sincronismo M-Sync para mapear as amostras de consumo na mesma base de tempo utilizada pelas estampas obtidas para a aplicação em análise.

Fonte: Adaptada de (XIAN et al., 2007).

Sendo mapeadas as amostras de consumo com a mesma base de tempo das estampas que definem a utilização da CPU, do HD e da placa de rede pela aplicação em análise, é possível determinar o gasto energético da aplicação para cada um destes módulos. Ao se utilizar informações baseadas nos acessos aos recursos de I/O, não apenas no tempo de CPU, esta plataforma apresenta maior exatidão que os demais métodos. Apesar disto, é incapaz de medir um trecho de código específico, pois não disponibiliza à aplicação em análise uma forma de indicar o início e o fim do processamento de interesse.

2.2.3.8 LEAP

Uma aprofundada pesquisa em medição de consumo de aplicações em software e o desenvolvimento de plataformas de medição foram realizados com o apoio da Intel pelo laboratório ASCENT, liderado pelo Prof. Dr. William Keiser, da Universidade da Califórnia em Los Angeles (UCLA). O objetivo deste projeto foi disponibilizar uma plataforma que auxiliasse o desenvolvimento de algoritmos computacionais energeticamente eficientes, o que resultou na criação do LEAP. A principal vantagem desta plataforma, quando comparada às demais plataformas disponíveis, deve-se à utilização de uma arquitetura amplamente difundida (processador Atom) e que possui um baixo custo para replicação (YAN et al., 2013).

O LEAP é constituído por dois componentes principais, ambos comerciais: uma placa-mãe D945GCL2 com processador Intel Atom N330 e um DAQ USB-6215 da National Instrument, mostrados na figura 12. A plataforma opera com sistema operacional Linux openSUSE 11.2.

A arquitetura do hardware do LEAP é mostrada na figura 13. Através dela é possível obter informações de consumo do processador Atom, memória RAM (SDRAM), HD e da ponte

Figura 12: Imagem dos principais componentes do plataforma LEAP. O DAQ da National Instrument, à esquerda e a placa-mãe Atom, à direita.

Fonte: (SINGH; KAISER, 2010).

norte (*northbridge*), denominada *Graphics and Memory Controller Hub* em sistemas da Intel. A corrente consumida por estes módulos é obtida através da amostragem analógica provida pelo DAQ, sendo esta enviada à placa-mãe através de uma interface USB. Conforme informações providas pelos *datasheets* dos componentes internos à placa-mãe envolvidos na alimentação do processador Atom e da ponte norte, o consumo desses componentes é obtido através da instrumentação dos indutores das fontes chaveadas dedicadas a eles. O acesso aos consumos do HD e da memória RAM é obtido através de resistores de 50 mΩ inseridos em série com a linha de alimentação destes módulos. A taxa de amostragem utilizada é de 10 kHz, sendo todos os canais amostrados simultaneamente (DIGVIJAY et al., 2010).

O princípio de medição utilizado no LEAP é baseado em marcações de tempo inseridas no código a ser analisado, permitindo assim determinar o período em que o processo foi executado. Além disso, as bases de tempo do sistema de amostragem do consumo e do processador Atom são sincronizadas, permitindo o rastreamento das medidas relativas a um determinado período. Este sincronismo toma como base de tempo o contador de *clocks* do processador Atom, denominado *Time Stamp Counter* (TSC), permitindo assim, que trechos de código executados tanto em nível de usuário quanto em módulos do *kernel* sejam analisados, pois o TSC é uma base de tempo única e acessível em qualquer ambiente do software.

O sincronismo entre o TSC e o sistema de amostragem é criado a partir de um sinal digital com período de 200 ms produzido através do pino RTS da porta serial disponível na

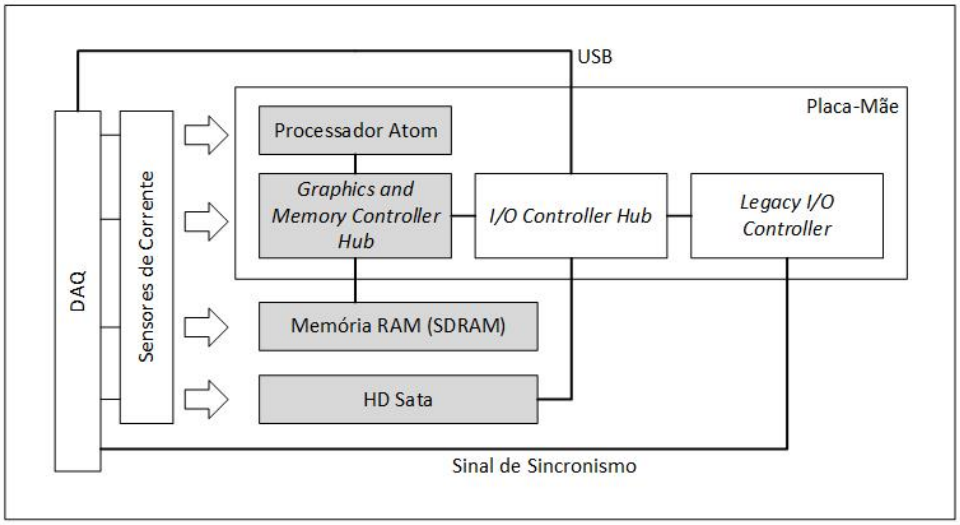


Figura 13: Arquitetura de hardware da plataforma LEAP. Em cinza estão os módulos cujo consumo é medido.

Fonte: Adaptada de (DIGVIJAY et al., 2010).

placa-mãe, sendo este pino amostrado pelo DAQ em conjunto com as demais medidas de consumo. A figura 14 mostra este método, em que cada vez que o sinal de sincronismo é comutado o TSC é armazenado, sendo esta mudança de estado medida pelo DAQ. Um processamento posterior analisa as medições do sinal de sincronismo realizadas pelo DAQ e atribui, a cada amostra em que houve transição do sinal, o relativo valor do TSC armazenado. Considerando que uma amostra é adquirida a cada 100 μs, mas o sinal de sincronismo só é alterado a cada 100 ms, uma distribuição linear atribui os valores de TSC ao restante das amostras.

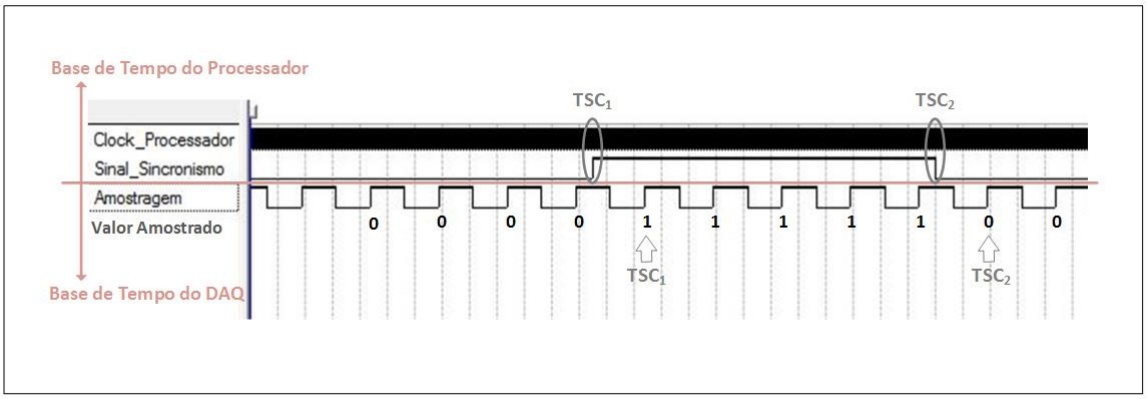


Figura 14: Método de sincronismo da plataforma LEAP.

Fonte: Autoria própria.

A medição de consumo através da plataforma LEAP é realizada em duas etapas: coleta

de dados e análise posterior, conforme arquitetura de software mostrada na figura 15. Para a primeira etapa, o código da aplicação em análise é instrumentado de forma a registrar o valor do TSC antes e depois da execução do algoritmo de interesse. Estes valores, denominados *calipers*, são armazenados na memória RAM a fim de se evitar *overhead* com escritas no HD. Durante a fase de coleta de dados, a aplicação em análise é executada juntamente com o software responsável pela leitura das amostras realizadas pelo DAQ (*start_samplig*). As medidas de consumo obtidas também são armazenadas em memória RAM. Um módulo do *kernel* foi desenvolvido (*probe*) para que o sinal de sincronismo seja comutado. A cada alteração de nível do sinal o valor do TSC é registrado nas mensagens do *kernel*.

Na última etapa, os dados coletados são analisados através de *scripts* e os resultados são compilados. Primeiramente, as amostras obtidas pelo DAQ são sincronizadas com as informações armazenadas pelo módulo do *kernel*. Desta forma, um valor do TSC é atribuído para cada amostra de consumo realizada. Um segundo *script* é utilizado para verificar quais medições ocorreram durante o período em que a aplicação em análise estava em execução. A figura 16 é um exemplo de resultado obtido através do LEAP.

A plataforma LEAP é uma das mais avançadas técnicas de medição do consumo energético de algoritmos computacionais; por este motivo foi detalhadamente analisada em conjunto com seus desenvolvedores, para fins de realização desta dissertação. O presente trabalho é um avanço na abordagem pioneiramente adotada nesta plataforma, adicionando principalmente a capacidade de separar a energia consumida pela aplicação em análise das demais tarefas em execução no processador e o suporte à comparação de consumo com hardwares dedicados.

2.3 MEDIÇÃO DE CONSUMO EM HARDWARE

2.3.1 CARACTERÍSTICAS DOS DISPOSITIVOS LÓGICO PROGRAMÁVEIS

Dispositivo Lógico Programável (PLDs - *Programmable Logic Devices*) é o termo aplicado aos componentes com a capacidade de configurar seu hardware para atender tarefas específicas. Diferente dos processadores, que executam programas através de um hardware fixo, a flexibilidade oferecida por estes componentes permite alterar consideravelmente o seu comportamento sem a substituição de componentes. Devido ao menor custo de desenvolvimento comparado aos CIs dedicados (ASICs - *Application Specific Integrated Circuits*), os PLDs se tornaram uma solução atraente para diversas aplicações embarcadas.

Os PLDs podem ser divididos em três grupos: *Simple Programmable Logic Devices*

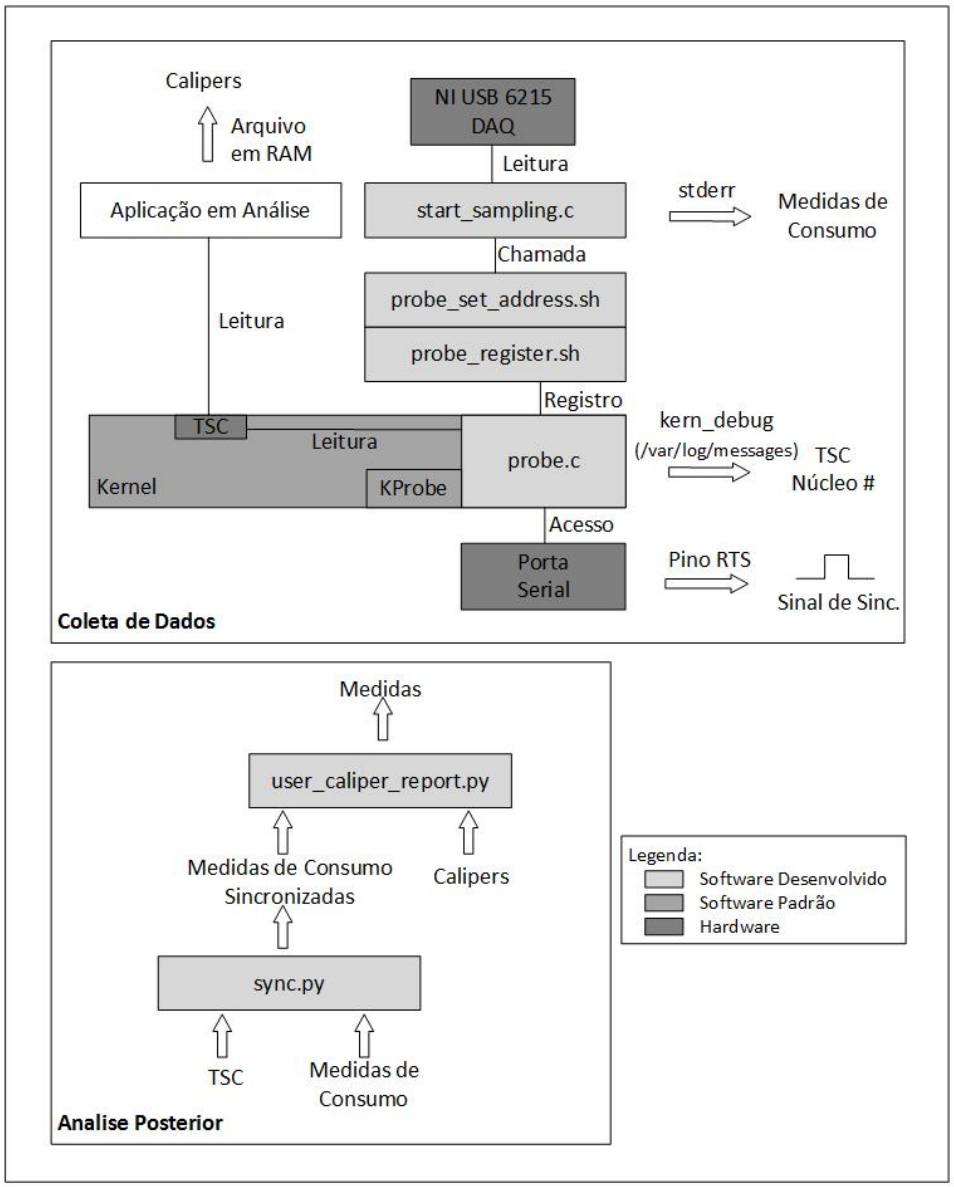


Figura 15: Arquitetura de software da plataforma LEAP, mostrando as etapas de coleta de dados (parte superior) e análise dos dados (parte inferior).

Fonte: Autoria própria.

(SPLDs), *Complex Programmable Logic Devices* (CPLDs) e *Field Programmable Gate Arrays* (FPGAs). Os SPLDs marcaram o início do desenvolvimento dos dispositivos programáveis e compreendem componentes com tecnologia *Programmable Array Logic* (PAL), *Programmable Logic Array* (PLA) e *Generic Array Logic* (GAL). Um dispositivo PAL é composto por um arranjo programável de portas AND seguido por um arranjo fixo de portas OR. Originalmente, estes componentes eram capazes apenas de executar funções combinacionais. Posteriormente, com a adição de flip-flops após cada porta OR, tornaram-se capazes de realizar operações sequenciais, mesmo que ainda simples. Os dispositivos PLA possuem arranjo similar aos

```
CPU_Energy (J) = 20.477718
HDD_Energy (J) = 2.572131
Bridge_Energy (J) = 25.904280
RAM_Energy (J) = 3.230877
CPU_Average_Power (W) = 10.078113
HDD_Average_Power (W) = 1.265875
Bridge_Average_Power (W) = 12.748797
RAM_Average_Power (W) = 1.590077
```

Figura 16: Exemplo de resultado obtido através da plataforma de medição LEAP. Fonte: Adaptada de (DIGVIJAY et al., 2010).

PAL, porém as portas OR também são programáveis, possibilitando uma maior flexibilidade. Contudo, devido a um maior tempo de propagação causado pela adição de interconexões programáveis, houve uma redução da velocidade máxima de trabalho (PEDRONI, 2010).

Como evolução dos dispositivos anteriores, a arquitetura GAL adicionou uma célula de saída mais sofisticada, incluindo flip-flop, porta XOR e multiplicadores, além de um sinal de retorno ao arranjo programável, conferindo assim mais versatilidade ao circuito. Outra evolução marcante foi o emprego de EEPROM para armazenar as interconexões, permitindo que o circuito fosse reprogramado, diferente de seus antecessores que permitiam uma única gravação. Os dispositivos SPLDs são obsoletos e não mais utilizados, com exceção dos GAL que ainda estão disponíveis no mercado, mesmo com interesse prático já bem reduzido. Os CPLDs sucederam os dispositivos anteriores baseando sua arquitetura na associação de vários SPLDs, normalmente GAL, em um mesmo chip, além da adição de *drivers* de I/O mais modernos e suporte a novas tecnologias, como JTAG (PEDRONI, 2010).

As FPGAs são dispositivos mais complexos e com maior densidade de circuitos que os demais componentes. Difere das CPLDs, que associam os blocos PLDs em forma de *pilha*, ao arranjar seus blocos lógicos em forma *matriz*, sendo cada bloco menor e mais sofisticado. As FPGAs modernas embarcam tecnologias avançadas como transceptores de alta velocidade, elementos de memória de rápido acesso, entre outros (PEDRONI, 2010). Desta forma, a escolha pela FPGA como componente para a plataforma de medição deve-se ao fato de ser o elemento disponível mais avançado e capaz de suportar hardwares complexos para diferentes propósitos.

Diversos fabricantes disponibilizam FPGAs, como Altera, Xilinx e Microsemi. A Altera se destaca por sua grande gama de componentes disponíveis, pelo bom suporte técnico e pelas avançadas ferramentas de desenvolvimento e análise, justificando assim a sua adoção neste trabalho.

A Altera disponibiliza o ambiente de desenvolvimento chamado Quartus, no qual é possível criar hardwares a partir de linguagens de descrição de hardware. O Quartus também integra ferramentas capazes de gerar descrições de hardware automaticamente a partir de blocos fornecidos. Desta forma, torna-se menos complexa a adição de processadores, IPs (*Intelectual Property Functions*), memórias, entre outros módulos, ao hardware a ser sintetizado para uma FPGA. Duas ferramentas destacam-se para este propósito. A primeira, chamada Qsys, reduz significativamente o tempo e esforço no desenvolvimento de aplicações em FPGA ao gerar automaticamente as conexões lógicas entre IPs e subsistemas. A segunda, denominada Megawizard, permite a criação de blocos de hardware automaticamente para funções pré-definidas. As implementações para FPGA realizadas neste trabalho se utilizam da linguagem VHDL com o auxílio das duas ferramentas mencionadas acima (ALTERA CORP., 2014d).

2.3.2 TÉCNICAS DE ANÁLISE DO CONSUMO DE HARDWARES

Sendo a FPGA constituída basicamente por portas lógicas e ligações entre elas, o seu consumo energético pode ser analisado através de duas componentes: potência estática e potência dinâmica, conforme a fórmula 4.

$$P_{total} = P_{estática} + P_{dinâmica} \tag{4}$$

A potência estática refere-se ao consumo enquanto o circuito permanece em um mesmo estado. Atribui-se a esta parte do consumo a manutenção da configuração da FPGA e a corrente drenada em um determinado estado das portas lógicas. A potência estática depende da impedância dos componentes, das interconexões internas do *chip*, do circuito configurado e o estado das entradas e saídas.

A potência dinâmica é o consumo demandado para que ocorram as transições de estado das portas lógicas. Com isto, sua análise pode ser realizada através da fórmula 5 (PEDRONI, 2010; JEVTIC; CARRERAS, 2011).

$$P_{dinâmica} = \alpha . C_l . V_{dd}^2 . f \tag{5}$$

em que α é a média do número de transições $0 \rightarrow 1$, denominada atividade das transições, C_l é a capacitância da carga, V_{dd} é a tensão da fonte de alimentação e f é a frequência do *clock*.

O valor da tensão de alimentação é normalmente conhecido e constante, sendo o mesmo é válido para a frequência de operação. A atividade das transições pode ser determinada

através de simulações no modelo do hardware. Com isto, apenas a capacitância da carga se mantém desconhecida para se determinar o consumo. Considerando as características deste parâmetro para os diferentes elementos que compõem uma FPGA, pode-se dizer que a potência dinâmica é dividida em três componentes: a potência do circuito de *clock* (dependente dos recursos dedicados para este fim), a potência consumida pela lógica das unidades funcionais e memórias (em que a capacitância corresponde às cargas manipuladas por cada porta lógica) e a potência dedicada à interconexão dos elementos (em que a capacitância da carga depende do tipo e comprimento dos fios) (JEVTIC; CARRERAS, 2011).

As FPGAs são disponibilizadas como um bloco fechado ao usuário, sendo suas estruturas elétricas não acessíveis externamente. Deste modo, há apenas duas formas de se obter o seu consumo energético: através de simulações baseadas nas características nominais dos elementos internos, levando em consideração o circuito sintetizado e vetores de teste para as entradas, ou através de medições diretas (JEVTIC; CARRERAS, 2011).

Diferentemente dos processadores, nos quais as simulações são a nível de instrução, as simulações de consumo para FPGAs devem ser a nível de transistor, pois não há uma arquitetura (processador, memória e linhas de ligação) fixa que mantém seu comportamento para diferentes aplicações. Para auxiliar a análise dos circuitos desenvolvidos, os fabricantes de *chip* fornecem ferramentas de análise de "baixo nível" e estimativa de consumo (JEVTIC; CARRERAS, 2011).

As ferramentas mais largamente utilizadas são PowerPlay da Altera e XPower da Xilinx. Estas ferramentas fornecem uma detalhada análise de consumo, baseada nos recursos utilizados pelo circuito, as capacitâncias envolvidas e a atividade das transições. Contudo, grandes diferenças entre os valores estimados pelas ferramentas e os valores obtidos com medições físicas foram encontradas pelos trabalhos que se propuseram a analisar a qualidade dos valores simulados. Estudos chegaram a apontar desvios superiores a 200% para os resultados obtidos com o XPower e superiores a 30% para o PowerPlay (MEINTANIS; PAPAEFSTATHIOU, 2008; MEINTANIS et al., 2011; JEVTIC; CARRERAS, 2011). Mesmo que estes dados precisem ser vistos com cautela, pois as medições referem-se a circuitos específicos, além da evolução das ferramentas poder ter reduzido estas diferenças, as simulações de consumo se aplicam para análises preliminares de consumo, não atendendo a projetos que exijam exatidão nas medidas. Destaca-se então a medição direta como o método adequado para análise detalhada do consumo de implementações em FPGA.

2.3.3 MEDIÇÃO DO CONSUMO DE HARDWARES

O trabalho mais avançado encontrado em medição do consumo energético de hardware em FPGA foi desenvolvido em (JEVTIC; CARRERAS, 2011), que propôs um método capaz de discriminar a potência estática, a potência do circuito de *clock*, a potência das interconexões e a potência consumida na lógica. Na abordagem proposta, uma vez que apenas a potência total pode ser medida, cada componente da potência é separada através de medições desenvolvidas especificamente para a análise e processamento posterior dos dados.

Os passos utilizados para separar os diferentes grupos de consumo são listados abaixo e detalhados na sequência. Todas as medições de potência referem-se ao consumo da linha de alimentação dedicada ao *core* da FPGA.

1. A potência estática é medida quando não são aplicados sinal de *clock* e entradas no circuito.
2. A potência do circuito de *clock* é medida quando o sinal de *clock* está ativo e as entradas do circuito são mentidas em “0”.
3. A potência total é medida quando o sinal de *clock* está ativo e um conjunto de vetores é aplicado às entradas do circuito.
4. O consumo das interconexões é obtido através da capacitância efetiva dos fios, computada a partir de valores de consumo. Estes valores são obtidos através da medição de circuitos especialmente desenvolvidos para este propósito.
5. A potência consumida pela lógica é obtida subtraindo os outros três componentes do consumo total.

Primeiramente, o consumo estático é medido quando nenhuma entrada ou *clock* são inseridos no *chip*. É importante notar que foi considerado que o consumo estático é constante, independente dos estados dos pinos de I/O ou do circuito interno. Esta suposição é válida apenas para circuitos pequenos, em que o acréscimo no consumo estático pode ser considerado desprezível.

Em seguida, o consumo do circuito de *clock* é medido em conjunto com o consumo estático. Para isso, o sinal de *clock* foi ativado enquanto todas as entradas foram configuradas para “0”. Para que esta medida efetivamente represente o consumo do *clock*, é necessário que não haja transição das portas internas do circuito. Esta conclusão pode ser obtida analisando o comportamento do modelo do circuito, quando submetido a uma entrada constante.

O consumo total do circuito é então medido através da excitação do circuito com o *clock* e um conjunto de vetores de entrada que reproduza o funcionamento da aplicação. No caso do trabalho apresentado em (JEVTIC; CARRERAS, 2011), um total de 10.000 sinais de entrada com distribuição Gaussiana e diferentes coeficientes de autocorrelação foram utilizados.

O passo seguinte é separar o consumo das interconexões do circuito. Para isto, o consumo da lógica do circuito é eliminado e a capacitância efetiva de todas as conexões é calculada. Primeiro, as medições anteriores são repetidas com o circuito localizado em duas posições distintas da FPGA, uma muito próxima dos pinos de I/O e outra o mais afastado possível, conforme figura 17. Subtraindo os dois valores obtidos para a potência dinâmica, tem-se o consumo causado pela diferença de interconexão entre as duas posições. Destaca-se que para se assumir isto, as entradas e saídas são registradas, de forma a garantir que *glitches* não sejam inseridos devido aos diferentes tempos de propagação dos sinais.

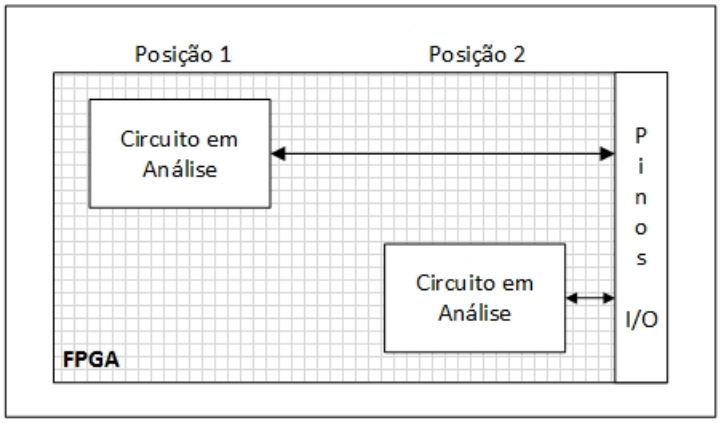


Figura 17: Ilustração dos dois posicionamentos do circuito na FPGA, utilizados para obtenção do consumo das interconexões. O afastamento entre o circuito e os pinos de I/O é alterado com objetivo de observar a diferença de consumo causado.

Fonte: Adaptada de (JEVTIC; CARRERAS, 2011).

Na FPGA utilizada para a elaboração deste método (Xilinx Virtex-2) existem quatro tipos de linhas de conexões, denominadas linhas *longas*, *hex*, *duplas* e *simples*. A capacitância para cada uma destas linhas deve ser calculada. Para isto, o hardware sintetizado em cada uma das duas posições é analisado com o auxílio de um algoritmo computacional para que seja extraída a quantidade e o tipo de linhas utilizadas em cada uma das i conexões entre os pinos de I/O e o circuito. Estes valores são chamados de n_{li}, n_{hi}, n_{di} e n_{si}. Como os vetores de entrada e saída das medições são conhecidos, é possível determinar a atividade das transições em cada pino de I/O, sw_i. É importante ressaltar que este método deve ser adaptado caso seja utilizada outra arquitetura de FPGA.

Considerando que, ao subtrair a potência dinâmica do mesmo circuito posicionado em dois locais distintos da FPGA, a potência da lógica é eliminada, e tendo como base a fórmula 5, a diferença entre os dois consumos medidos pode ser determinada pela fórmula 6.

$$\begin{aligned} P_1 - P_2 = V_{dd}^2.f.(&C_l.\sum_{i=1}^{I+O}[(n_{1li} - n_{2li}).sw_i] \\ &+C_h.\sum_{i=1}^{I+O}[(n_{1hi} - n_{2hi}).sw_i] \\ &+C_d.\sum_{i=1}^{I+O}[(n_{1di} - n_{2di}).sw_i] \\ &+C_s.\sum_{i=1}^{I+O}[(n_{1si} - n_{2si}).sw_i]) \end{aligned} \tag{6}$$

em que P_1 e P_2 são as potências dinâmica medidas para o circuito localizado longe e próximo dos pinos de I/O, respectivamente, I é a largura da palavra de entrada, O é a largura de palavra de saída, V_{dd} é a tensão de alimentação do *core* da FPGA, previamente conhecido, f é a frequência de operação, também previamente conhecida, sw_i é a atividade das transições, obtido conforme explicado anteriormente, e C_l, C_h, C_d e C_s são as capacitâncias efetivas de cada um dos tipos de linhas de conexão.

Restando ainda como desconhecidas apenas as variáveis C_l, C_h, C_d e C_s, estas podem ser obtidas através de uma regressão múltipla aplicada a uma série de medições com diferentes larguras de entrada. Em posse destes valores, finalmente a potência das interconexões pode ser calculada extraindo o número total de cada tipo de linha de conexão utilizada no circuito sintetizado e aplicando a fórmula 7.

$$P = V_{dd}^2.f.sw.(n_l.C_l + n_h.C_h + n_d.C_d + n_s.C_s) \tag{7}$$

O *buffer* de entrada também é alimentado através do *core*, precisando este também ser extraído do restante do valor da potência total ainda não conhecido. O consumo dos *buffers* de entrada pode ser descoberto para determinado modelo de FPGA através da medição de dois circuitos simples: um multiplicador implementado através de apenas um LUT (*Look-Up Table*) e um multiplicador implementado através de três LUTs. Através do método descrito até então, obtêm-se a potência dinâmica de cada circuito, extrai-se a potência de interconexão e obtêm-se a potência do *buffer* de entrada através do sistema da fórmula 8.

$$\begin{cases} P_{m1} = 3.P_{mult} + P_{buffer} \\ P_{m2} = P_{mult} + P_{buffer} \end{cases} \tag{8}$$

em que P_{m1} e P_{m2} são as potências dinâmicas medidas para o circuito multiplicador com um e três LUTs, P_{mult} é a potência da lógica do multiplicador e P_{buffer} é a potência consumida pelo *buffer* de entrada.

Ao aplicar o valor da potência consumida pelo *buffer* e demais valores conhecidos na fórmula 5, é possível obter a capacitância efetiva de um *buffer* de entrada da FPGA. Finalmente, é possível obter-se o consumo do circuito lógico subtraindo o consumo do circuito de *clock*, o consumo da interconexão e o consumo dos *buffers* de entrada do consumo dinâmico do circuito.

A separação entre as diferentes fontes de consumo da FPGA é uma informação importante para o desenvolvimento de hardwares energeticamente eficientes, pois fornece indicações de quais partes do circuito podem estar demandando maior consumo. Contudo, a título de comparação com o consumo de implementações em software, apenas o consumo total da FPGA é de interesse, pois o funcionamento do hardware depende de todos os seus elementos internos.

3 DESENVOLVIMENTO

Este capítulo contempla o desenvolvimento da plataforma de medição de consumo. Primeiramente, é descrito o ambiente disponível para a aplicação que será medida, incluindo o suporte à interface PCIe, que permite a utilização de coprocessadores pelo software. Na sequência, é apresentada a arquitetura desenvolvida para a medição do consumo e é explicado o procedimento para realizar medições através da plataforma. Por fim, é realizada uma comparação entre o método de medição desenvolvido e as demais plataformas disponíveis na literatura.

3.1 AMBIENTE DISPONÍVEL PARA APLICAÇÃO EM ANÁLISE

A necessidade de um método de medição único e capaz de comparar o consumo de algoritmos equivalentes em software e em hardware motivou a definição de uma plataforma de hardware que permitisse a execução de aplicações híbridas, assim como aplicações apenas em software ou apenas em hardware. A arquitetura x86 foi escolhida para a execução das aplicações em software, pois é largamente difundida e a maioria dos algoritmos disponíveis foram desenvolvidos para esta arquitetura. Além disto, processadores x86 são utilizados tanto em aplicações de alto desempenho, quanto em dispositivos móveis. Em relação ao processamento em hardware, uma necessidade imprescindível para projetos de análise de consumo é o estudo de diferentes implementações, exigindo frequentes mudanças. Sendo assim, uma FPGA é a solução adequada, pois permite a avaliação dinâmica de diferentes hardwares.

3.1.1 PRINCIPAIS COMPONENTES DA PLATAFORMA

O ambiente de execução disponível para a aplicação em análise tem como elemento central uma placa-mãe mini-ITX de propósito geral. As placas mini-ITX destacam-se pelo seu baixo consumo e têm como principal característica o formato mecânico padrão de 17 x 17 cm. Uma placa-mãe DN2800NT com um processador Atom N2800 é utilizada, pois, no momento em que este trabalho foi arquitetado, esta placa e seu processador possuíam a mais

atualizada tecnologia Atom disponibilizada pela Intel. Quando comparada às demais famílias de processadores x86, a família Atom tem como principal vantagem a eficiência energética, o que justificou a sua escolha. Em conjunto com a placa-mãe são utilizados 4GB de memória RAM DDR3 e um HD de 500 GB.

O sistema operacional adotado para o Atom é a distribuição Linux openSUSE 13.1 (*kernel* 3.11.10-25). A escolha por esta versão de sistema operacional foi pautada em dois motivos: a disponibilidade de um *kernel* atualizado e compatível com a versão mais atual do compilador para as linguagens C/C++ e o suporte ao *driver* do DAQ utilizado para a amostragem de consumo.

Para as implementações em hardware, é disponibilizada uma placa Cyclone IV Development Kit da Altera. O quadro 1 mostra as características da FPGA EP4CGX150N presente nesta placa.

Característica	**Quantidade**
Elementos Lógicos (LEs)	149.760
Blocos de Memória M9K	720
Memória Embarcada	6.480
Multiplicadores (18 bit x 18 bit)	360
PCI Express (PCIe) hard IP	1
Phase-locked loops (PLLs)	8
Transceptores de I/Os	8
Número Máximo de I/Os	475
Máximo de Canais Diferenciais	216

Quadro 1: Características da FPGA EP4CGX150N presente na plataforma de Medição. Fonte: Adaptado de (ALTERA CORP., 2014a).

A figura 18 mostra uma foto do ambiente de execução disponível para a aplicação em análise.

3.1.2 SUPORTE À INTERFACE DE COMUNICAÇÃO PCIE

A placa-mãe Atom e a placa contendo a FPGA são conectadas através de uma interface PCIe Express com o objetivo de permitir a comunicação entre um software e um coprocessador em hardware. A comunicação PCIe é um barramento de expansão serial de alta velocidade, que foi desenvolvido para substituir os barramentos PCI, PCI-X e AGP, anteriormente utilizados. A PCIe utiliza uma conexão bidirecional capaz de enviar e receber dados simultaneamente, sendo o modelo utilizado conhecido como *dual-simplex*. Esta denominação é atribuída devido

Figura 18: Foto da plataforma de hardware disponível para aplicação em análise composta por uma placa-mãe com processador Atom e uma placa com FPGA da família Cyclone IV.

Fonte: Autoria própria.

à utilização de um transmissor e um receptor diferencial por via de comunicação, conforme mostrado na figura 19 (JACKSON; BUDRUK, 2012).

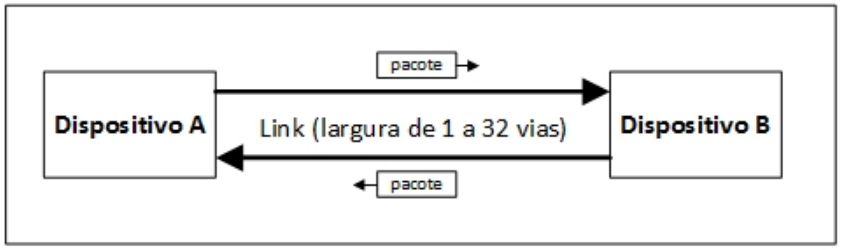


Figura 19: Diagrama em blocos do link da comunicação *dual-simplex* utilizado pela interface PCIe.

Fonte: Adaptada de (JACKSON; BUDRUK, 2012).

O conjunto de sinais de comunicação entre os dispositivos PCIe é denominado *link*. Cada link é constituído de um ou mais pares de transmissores e receptores, sendo cada um deles chamado de *via*. Um link PCIe pode possuir de 1 a 32 vias.

A maioria dos dispositivos PCIe disponíveis no mercado pertencem a duas gerações deste barramento, que diferem pela velocidade máxima de transmissão, sendo elas: 1.x (Gen1) e 2.x (Gen2). Os barramentos Gen1 operam até 2,5 Gbps, enquanto que os Gen2 atingem 5 Gbps. Um *overhead* de 25% está presente na comunicação devido à codificação 8B / 10B utilizada, resultando em uma taxa máxima de transmissão de 2 Gbps por via para a Gen1 e 4 Gbps por

via para a Gen2 (ALTERA CORP., 2014b).

A topologia da interface PCIe, como mostrado na figua 20, possui três tipos de componentes: *root complex*, *switches* e *endpoint*. Denomina-se *root complex* o dispositivo que conecta o processador e a memória ao restante da comunicação PCIe. Este dispositivo realiza requisições aos barramentos em nome do processador e implementa funções essenciais ao funcionamento da comunicação, como o controle de componentes recém conectados ao barramento (*hot plug*), o gerenciamento de energia, o controle das interrupções e a detecção de erro (JACKSON; BUDRUK, 2012).

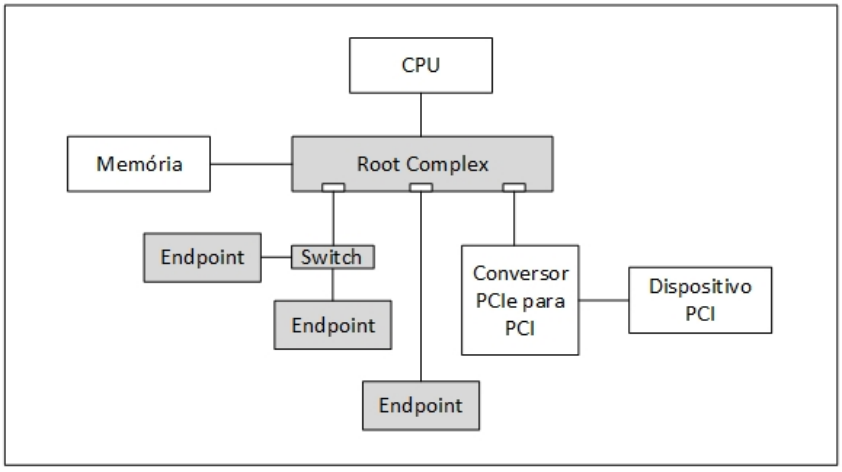


Figura 20: Topologia da interface PCIe indicando os principais componentes: *root complex*, *switch* e *endpoint*.

Fonte: Adaptada de (JACKSON; BUDRUK, 2012).

O *root complex* pode possuir uma ou mais portas; no exemplo da figura 20, o dispositivo possui 3 portas, cada porta originando um barramento PCIe no qual os demais dispositivos são conectados. Um barramento é capaz de suportar até 32 dispositivos; contudo, a natureza ponto-a-ponto do PCIe permite que apenas um dispositivo seja diretamente ligado a ele. Para que mais de um componente seja conectado, os *switches* são utilizados para virtualizar um único barramento para vários dispositivos (JACKSON; BUDRUK, 2012).

Os demais dispositivos conectados ao barramento, que não são *root complex* e *switches*, são denominados *endpoint*. Os *endpoints* são periféricos que complementam o funcionamento do sistema, como Ethernet, USB, dispositivos gráficos, ou, no caso deste trabalho, um coprocessador em hardware.

Os componentes de um barramento PCIe são classificados como *requester* ou *completer*, conforme o comportamento que desempenham na comunicação. *Requesters* são os dispositivos que originam uma transação no barramento PCIe e *completers* são os dispositivos

endereçados. Sendo assim, um *requester* lê dados de um *completer* ou escreve dados em um *completer*. Um dispositivo *enpoint* pode ser *completer* ou *requester*, conforme sua finalidade, bem como pode realizar ambas as funções. Os espaços de memória disponibilizados por dispositivos do barramento PCIe e acessíveis ao sistema operacional em execução no processador são denominados BAR (JACKSON; BUDRUK, 2012).

3.1.2.1 HARDWARE DESENVOLVIDO PARA FPGA

O conector PCIe disponível na placa-mãe possui apenas 1 via de comunicação da primeira geração (Gen1 x1), enquanto que a interface disponível na placa com a FPGA é de quatro vias (Gen1 x4). Para permitir a conexão física entre as duas placas, é utilizado um adaptador, no qual, apesar de comportar conectores de maior porte, apenas a ligação de uma via é realizada.

As FPGAs da família Cyclone IV GX, como a utilizada neste trabalho, possuem transceptores bidirecionais (*full-duplex*) de alta velocidade. Estes blocos podem ser configurados de forma a operar com diversos padrões industriais de comunicação. As FPGAs desta família também incorporam um *Hard IP* para PCIe capaz de prover uma solução completa para o acesso à interface, implementando a camada física, a camada de enlace de dados e a camada de transação (ALTERA CORP., 2014a). Desta forma, os sinais da PCIe podem ser ligados diretamente aos pinos da FPGA, eliminando a necessidade de componentes externos e aumentando a eficiência energética da arquitetura.

A implementação da interface PCIe na FPGA exigiu o desenvolvimento de uma arquitetura de hardware para viabilizar a operação do *Hard IP* e para prover à aplicação em análise uma fácil integração. O hardware desenvolvido possui as funções de *requester* e *completer*, ou seja, além de responder às requisições do barramento, é capaz de iniciar novas transações, suportando assim, o envio de interrupções para o software em execução no processador Atom.

O diagrama em blocos da Figura 21 mostra a arquitetura criada para a FPGA. O componente central deste hardware é o IP PCIe obtido a partir da ferramenta Qsys. O IP foi configurado como um *endpoint* da comunicação PCIe com 1 via de comunicação da primeira geração (Gen1 x1). Este componente tem a função de prover acesso aos blocos de memória compartilhados com o software (BAR) através de barramentos Avalon-MM, que, por sua vez, são disponibilizados para o hardware da aplicação em análise. O Avalon-MM permite uma fácil integração por ser um barramento paralelo de 32 bits de largura com endereçamento byte a byte. Desta forma, a complexidade da comunicação PCIe é abstraída para o hardware da

aplicação, sendo convertida em operações de leitura e escrita equivalentes ao acesso dos demais componentes internos de uma FPGA, como, por exemplo, uma memória RAM.

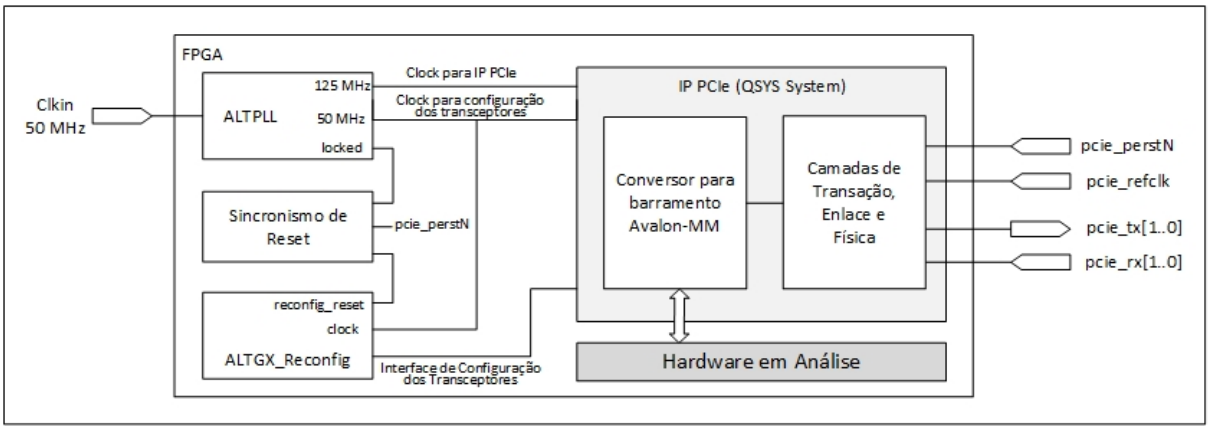


Figura 21: Diagrama em blocos do hardware da FPGA para implementação da interface PCIe. Fonte: Adaptada de (HAWKINS, 2012).

A operação do IP PCIe exige a implementação de um PLL para converter o *clock* de 50 MHz, disponível na placa da FPGA, para 125 MHz, necessário para sincronismo com o barramento. Também é exigido um bloco para configuração dos transceptores logo após a ocorrência de um reset. Estes hardwares foram criados a partir da ferramenta MegaWizard. Qsys e MagaWizard, conforme descrito na seção 2.3.1, são ferramentas disponibilizadas em conjunto com o ambiente de desenvolvimento Quartus II 13.0 sp1, utilizado neste projeto.

A fim de permitir o correto sincronismo e a filtragem de *glithes* durante o reset da PCIe, foram utilizados componentes baseados no material disponibilizado por (HAWKINS, 2012). Em resumo, os seguintes artefatos compõem o projeto do hardware que implementa a comunicação PCIe na FPGA:

pcie_gen1x1_top.vhd: Entidade topo do projeto desenvolvida em VHDL, sendo responsável por instanciar todos os componentes e definir os sinais de controle entre eles.

qsys_system.qsys: Projeto criado na ferramenta Qsys para instanciar, rotear e configurar o IP PCIe.

altgx_reconfig.qip: Componente criado na ferramenta MegaWizard e responsável por configurar os transceptores PCIe no reset da interface.

altpll.qip: PLL criado na ferramenta MagaWizard para gerar o *clock* dos componentes qsys_system e altgx_reconfig.

sync.sv e glitch_filter.sv (sincronismo de reset): Componentes utilizados para sincronismo dos sinais de reset e filtragem de *glitches* ocorridos no início da operação do sistema.

pcie_gen1x1_top.tcl: Script para automatizar a configurações do projeto no Quartus e definir os pinos de I/O utilizados na FPGA.

set_to_flash.sh: Script para conversão do arquivo .sof, gerado a partir da síntese do hardware realizada pelo Quartus, em .flash para gravação na memória flash da placa com a FPGA. Permite que a FPGA seja configurada com o hardware criado logo após a energização da placa.

3.1.2.2 DRIVER DESENVOLVIDO PARA O SISTEMA OPERACIONAL

Para que uma aplicação em software utilize facilmente a comunicação PCIe implementada na FPGA, é necessário que uma camada de software abstraia a complexidade da comunicação e as particularidades da interface criada. O ambiente Linux proporciona duas principais maneiras de se estabelecer uma comunicação PCIe: utilizando a estrutura Sysfs (automaticamente criada pelo *kernel* do Linux) ou desenvolvendo um driver para o dispositivo.

A abordagem inicialmente adotada neste trabalho utilizava a estrutura Sysfs do Linux. Nesta estrutura, durante a inicialização do sistema operacional, uma árvore de diretórios é disponibilizada em */sys/bus/pci/devices/*. Ao se manipular os arquivos presentes neste local, tem-se acesso a todos os dispositivos PCI/PCIe encontrados no equipamento. Desta forma, é possível virtualizar as memórias disponibilizadas pelos dispositivos do barramento (BARs) em uma aplicação a nível de usuário, permitindo assim realizar escritas e leituras como se fosse qualquer outra posição de memória disponibilizada pelo Linux. A vantagem de se adotar este método é não precisar desenvolver uma camada de software para se ter acesso às funcionalidades básicas disponibilizadas por um dispositivo do barramento PCI/PCIe; porém, através deste método, não é possível receber interrupções provenientes do barramento. Isto exige a realização de constantes requisições ao barramento (*polling*) sempre que se precisa determinar o momento em que o coprocessador terminou uma tarefa. Considerando que este método reduz a eficiência energética da solução, a utilização do Sysfs como forma de acesso ao coprocessador em hardware foi descartada.

Um driver dedicado para a plataforma foi então desenvolvido a fim de tomar controle da comunicação com a FPGA e criar um dispositivo no ambiente Linux (*/dev/alt_up_pci0*). Através deste driver, uma aplicação do usuário, além de realizar operações de leitura e escrita no espaço de memória da FPGA, recebe sinalizações do driver sempre que ocorre uma interrupção

no barramento PCIe. Com isto, o driver criado propicia, para aplicação em análise, o acesso a todas as funcionalidades do barramento.

Aplicações que exigem elevadas taxas de transmissão para grande volume de dados necessitam de métodos mais eficientes de acesso ao barramento que instruções de leitura e escrita, pois estas ocupam o processador durante todo o processo de transmissão do dado. Para atender altas taxas de transmissão, o driver criado é capaz de controlar até quatro DMAs simultaneamente, se disponíveis no hardware da FPGA. O DMA é um módulo de hardware capaz de realizar transferências de blocos de memória sem ocupar o processador principal durante o período de transmissão. DMAs são largamente utilizados em hardwares para FPGA que possuem interfaces de comunicação com altas taxas de transmissão, como é o caso da PCIe.

O desenvolvimento do driver foi baseado no exemplo fornecido pela Altera para o kit de desenvolvimento DE4 (ALTERA CORP., 2011), porém, foram necessárias adaptações para suportar o hardware criado na FPGA, além de desenvolver a captura e sinalização das interrupções provenientes do barramento, originalmente não suportadas.

Uma biblioteca para simplificar o acesso ao driver foi criada em linguagem C com o objetivo de facilitar a utilização do coprocessador pelo software da aplicação em análise. Desta forma, o acesso aos dados disponibilizados por um hardware na FPGA é realizado através das seguintes funções:

alt_up_pci_open: Inicia a comunicação com a FPGA através do *driver*.

alt_up_pci_close: Finaliza a comunicação com a FPGA através do *driver*.

alt_up_pci_read: Realiza a leitura de um bloco de dados da memória disponibilizada pela FPGA.

alt_up_pci_write: Realiza a escrita de um bloco de dados na memória disponibilizada pela FPGA.

alt_up_pci_dma_add: Adiciona a transferência de um bloco de memória através de DMA na fila de transições do *driver*.

alt_up_pci_dma_go: Inicia uma transferência por DMA que está na fila do *driver*.

3.2 PLATAFORMA PARA MEDIÇÃO DE CONSUMO

Para obter o consumo em software (placa-mãe Atom) e em hardware (placa com FPGA), foi desenvolvida uma arquitetura de medição que permite taxas de amostragens acima

de 1 ksps, sendo assim capaz de medir o consumo de pequenos trechos de código. A arquitetura também provê um sincronismo que permite identificar a execução de uma aplicação específica no processador Atom. Outro requisito imprescindível atendido pela plataforma de medição criada é permitir uma fácil integração com a aplicação em análise, de forma a causar o menor impacto em desempenho possível.

3.2.1 FERRAMENTA DE AMOSTRAGEM UTILIZADA

Diversas ferramentas para amostrar o consumo foram exploradas pelas plataformas de medição presentes na literatura, sendo que as limitações normalmente encontradas são a baixa taxa de amostragem e o elevado custo. Um requisito necessário e dificilmente atendido é a possibilidade das amostras serem lidas, tanto pela própria plataforma de medição, possibilitando uma solução unificada e compacta, quanto por um segundo computador, reduzindo assim o *overhead* causado pela amostragem. A possibilidade de replicar a plataforma de medição através da utilização de itens comerciais de fácil aquisição também foi um fator considerado na escolha da ferramenta de amostragem.

As ferramentas de aquisição de dados (DAQ) da National Instruments atendem a estas necessidades, além de fornecer biblioteca em C para o desenvolvimento de aplicações para a leitura das medidas. O DAQ escolhido é o NI USB-6008, pois atende os requisitos e possui um baixo custo de aquisição. A figura 22 mostra uma imagem do DAQ utilizado. Com a adoção desta ferramenta, o aumento da taxa de amostragem e do número de canais de medição pode ser facilmente obtido através da troca do DAQ por outro modelo de maior capacidade e diretamente compatível com a biblioteca utilizada.

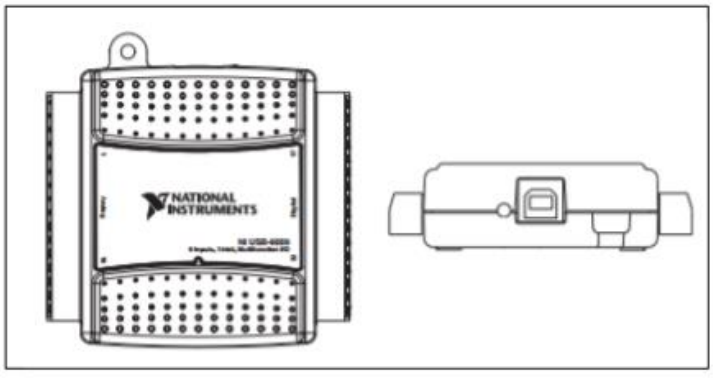


Figura 22: Imagem do sistema de aquisição de dados (DAQ) NI USB-6008 utilizado na plataforma de medição.

Fonte: Adaptada de (NATIONAL INSTRUMENTS CORP., 2012).

O DAQ NI USB-6008 possui interface de comunicação USB, 12 entradas/saídas digitais, 2 saídas analógicas, 8 entradas analógicas, que podem ser configuradas como 4 entradas diferencias, 12 bits de resolução digital para as entradas/saídas analógicas e frequência de amostragem de até 10 ksps (NATIONAL INSTRUMENTS CORP., 2012). O DAQ utiliza o CI Burr-Brown ADS7870 como elemento central do seu hardware. Através do *datasheet* deste componente é possível obter as especificações elétricas do sistema de amostragem, considerando que os recursos utilizados são as entradas analógicas configuradas como diferenciais e operando com as seguintes características: excursão da medição de ± 1 V; ganho para o sinal de entrada de 20 V/V; e temperatura ambiente de 20°C. As características de interesse deste trabalho são mostradas no quadro 2.

Parâmetro	**Valor**
Capacitância de Entrada	de 4 à 9,7 pF
Impedância de Entrada	7 MΩ
Rejeição à Interferência Entre Canais	100 dB
Corrente de Fuga	10 pA
Resolução	12 bits
Erro de *Offset* (*)	-0,5 LSB
Erro de Linearidade Integral	± 2 LSB
Erro de Linearidade Diferencial	$\pm 0{,}5$ LSB
Erro de Ganho (*)	0 LSB
Taxa Máxima de Amostragem	10 ksps
Tempo de *Set-up* do Dado	10 ns
Tempo de Amostragem do Dado	10 ns

Quadro 2: Especificações elétricas da ferramenta de amostragem (DAQ NI USB-6008), conforme configuração utilizada na plataforma de medição.

Fonte: Adaptado de (TEXAS INSTRUMENTS INC., 2005).

(*) Considerando ganho de 20 V/V e temperatura ambiente de 20°C.

3.2.2 ARQUITETURA DO HARDWARE DE MEDIÇÃO

A figura 23 mostra o diagrama do hardware da plataforma de medição de consumo. As placas são alimentadas através de duas fontes 15 VCC independentes, fornecidas através de uma fonte programável Agilent E3646A. Utilizou-se uma fonte programável para garantir uma regulação constante da tensão, independente da variação do consumo das cargas ou flutuação da rede CA. Dois resistores utilizados como sensores de corrente, um de 270 mΩ e um de 1 Ω, foram respectivamente inseridos nas linhas de alimentação da placa-mãe e da placa da FPGA. O resistor de 270 mΩ foi obtido através da associação de quatro resistores de 1 Ω em paralelo. Para garantir a exatidão das medidas, os valores dos resistores correspondem ao valor medido através

de um multímetro e não aos seus valores nominais. Através das entradas analógicas do NI DAQ USB-6008, configuradas para medição diferencial, é amostrada a tensão sobre os resistores. Conhecendo a tensão de alimentação de cada placa (15 V) e a corrente consumida, obtida a partir da tensão sobre os resistores, obtêm-se o consumo instantâneo da placa no momento que o DAQ realiza uma amostra.

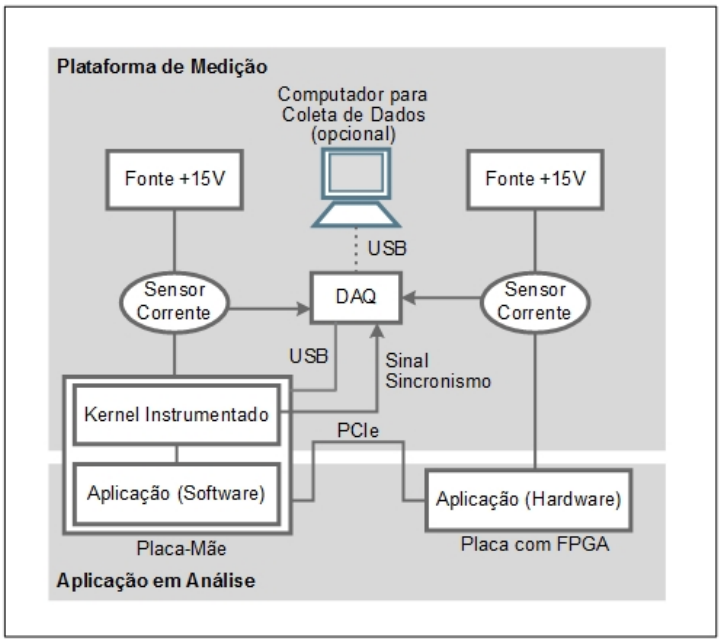


Figura 23: Diagrama em blocos do hardware da plataforma de medição (acima) e do hardware disponível para a aplicação em análise (abaixo).

Fonte: Autoria própria.

Para prevenir *overheads* desnecessários, a figura 23 mostra um segundo computador utilizado para a coleta dos dados adquiridos pelo DAQ. Este segundo computador é opcional, podendo a coleta dos dados ser realizada pela própria placa-mãe Atom. Neste caso, considerando que a plataforma é capaz de medir individualmente a aplicação em análise, o processamento demandado para a leitura dos dados do DAQ não afeta o resultado da medição. O único *overhead* causado pela conexão do DAQ na própria plataforma deve-se à elevação do consumo proveniente da alimentação deste componente pela porta USB. O consumo típico do DAQ é de apenas 400 mW, representando uma elevação de aproximadamente 2,7% no consumo médio da plataforma, quando executando o sistema operacional (inativa).

Um dos maiores avanços da plataforma implementada é a capacidade de medir o consumo individualizado de um aplicação no sistema operacional. Para permitir isto, um sinal de sincronismo é gerado sempre que a aplicação em análise está em execução no processador Atom. Este sinal é gerado através dos pinos da interface paralela disponível na placa-mãe, que,

por sua vez, é amostrado pelo DAQ em conjunto com as demais medidas de consumo.

Mesmo que a latência do sinal de sincronismo gerado deva ser validada, uma baixa latência é esperada como resultado da arquitetura da placa-mãe. A porta paralela está integrada ao *Legacy I/O Controler*, que, por sua vez, está diretamente ligado ao *I/O Controler Hub* da placa.

Considerando que uma entrada do DAQ é dedicada ao sinal de sincronismo, até três linhas de medição (entradas analógicas diferencias) podem ser utilizadas simultaneamente, sendo que duas delas estão dedicadas à medição do consumo total da placa-mãe Atom e da FPGA. Resta ainda uma linha de medição disponível, que pode ser utilizada para medições de submódulos das placas, como, por exemplo, memória RAM ou HD. Um arquivo de configuração é utilizado para armazenar um nome de identificação, a tensão e a impedância do sensor de corrente de cada linha de medição; desta forma, torna-se simples a alteração ou adição de linhas de medição, não sendo necessário alterar a solução de software implementada.

Devido a limitações do DAQ, a taxa de amostragem é reduzida conforme um maior número de entradas analógicas são utilizadas. A progressão da taxa de amostragem com o número de linhas de medição ativas é mostrado no quadro 3.

Quantidade de Linhas de Medição	Taxa de Amostragem
1	5 ksps
2	3 ksps
3	1 ksps

Quadro 3: Progressão da taxa de amostragem do DAQ conforme o número de linhas de medição utilizadas.

Fonte: Autoria própria.

Como resultado da arquitetura desenvolvida, o quadro 4 mostra a lista de materiais necessários para a replicação da plataforma e a figura 24 mostra o ambiente de hardware pronto para a medição.

3.2.3 ARQUITETURA DO SOFTWARE DE MEDIÇÃO

A medição de um único processo em execução em um sistema operacional multitarefa exigiu a criação de uma arquitetura para sincronismo entre a aplicação em análise e as amostras de consumo. Para isso, a arquitetura do software de medição executado na placa-mãe Atom permeia tanto a camada de aplicação do usuário quanto o *kernel* do sistema operacional. A figura 25 mostra em detalhe as partes que compõem o software da plataforma de medição.

Descrição	Quantidade
Placa-mãe Intel DN2800MT	1
Cyclone IV GX FPGA Development Kit	1
2GB DDR3 Corsair CM3X2GSD1066	2
HD 500GB 7200rpm 16M Sansung	1
Adaptador para conector PCIe de 1x para 16x	1
National Instruments DAQ USB-6008	1
Resistor Axial 1 Ω 500 mW	5

Quadro 4: Lista de materiais que compõem a plataforma de medição.

Fonte: Autoria própria.

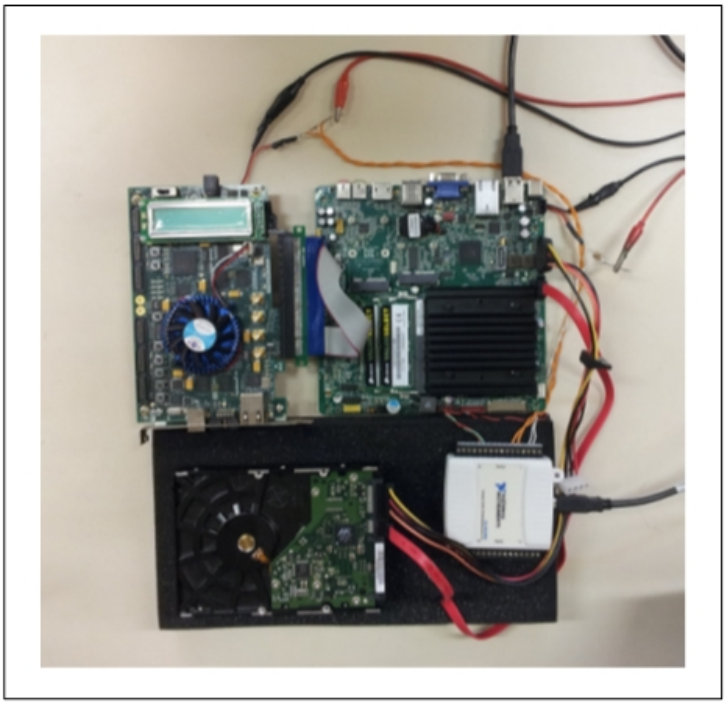


Figura 24: Foto da plataforma de medição desenvolvida.

Fonte: Autoria própria.

Um módulo inserido no *kernel* monitora o funcionamento do escalonador de forma a indicar, através do sinal de sincronismo, sempre que a aplicação em análise inicia ou para sua execução no processador. Isto permite identificar os períodos em que a aplicação não estava em execução para que o processador tratasse as demais tarefas do sistema operacional, mesmo que o trecho de código de interesse já tenha iniciado.

A instrumentação do *kernel* do Linux pode ser realizada de duas formas. Uma delas é editar o código do sistema operacional e compilar para a plataforma a ser utilizada. Este método insere o menor *overhead* possível, pois a informação pode ser tratada no próprio

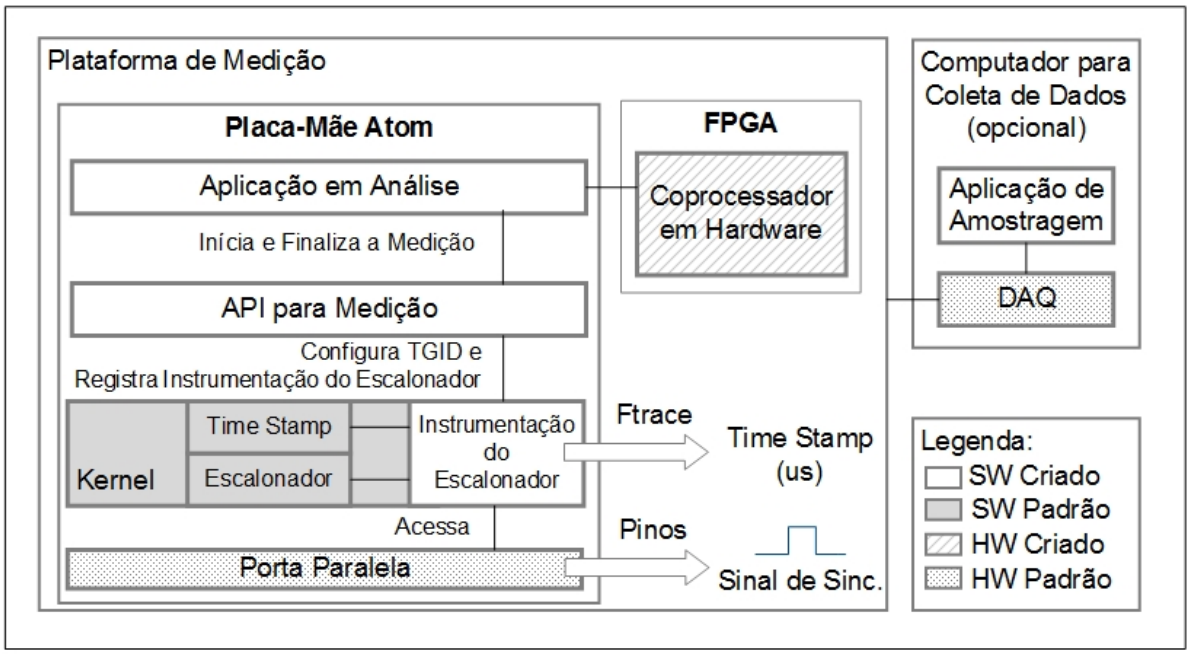


Figura 25: Diagrama em blocos do software da plataforma de medição indicando os módulos de software desenvolvidos e o ambiente de software em que são executados.

Fonte: Autoria própria.

contexto em que está disponível. Contudo, a utilização do código para outros ambientes de hardware e a atualização do sistema operacional se tornam atividades complexas. A segunda forma de instrumentar o *kernel* é utilizar a ferramenta *Kernel Probe* (Kprobe). Kprobe é uma API disponibilizada pelo Linux que permite instrumentar rotinas no nível do *kernel*. Através desta ferramenta é possível adicionar sensores em qualquer função do *kernel* desde que seu símbolo esteja disponível no mapa de memória (*/proc/kallsyms*). A principal característica desta API é permitir que rotinas customizadas sejam executadas sempre que determinada função do *kernel* for chamada. Através do Kprobe, apesar da instrumentação do escalonador estar intrinsecamente ligada ao funcionamento do *kernel*, o código desenvolvido se torna compatível com diferentes versões do *kernel* e a atualização do sistema operacional é realizada sem afetar o funcionamento da instrumentação (NOVELL INC., 2011).

Utilizando-se do Kprobe, a instrumentação do escalonador analisa cada troca de contexto e verifica o TGID (explicado na seção 2.2.1.1) do processo que está sendo retirado do processador e do processo que está iniciando a execução. Quando o processo que está prestes a iniciar (ou parar) pertence à aplicação em análise, é alterado o estado do pino da porta paralela e é armazenada a estampa de tempo. Com isto, sempre que o sinal de sincronismo está acionado significa que o trecho de código a ser medido está em execução. Através da análise do código fonte do *kernel*, especificamente do escalonador CFS, determinou-se que é possível obter o

momento em que determinado processo é alocado no processador através da instrumentação da função *switch_to()* do escalonador, chamada no momento da troca de contexto.

Além do sincronismo e da exatidão da amostragem do consumo instantâneo, a medição da energia consumida pela aplicação em análise exige um método eficaz de medir o tempo de processamento. Durante o desenvolvimento da plataforma, os três métodos seguintes de medição do tempo de processamento foram avaliados:

Comando *time*: Através do comando *time* fornecido com as distribuições do Linux, as informações armazenadas pelo escalonador no descritor do processo são utilizadas para determinar o tempo de processamento do mesmo. O resultado é obtido ao final da execução da aplicação.

Prós:

Descrimina o tempo de processamento em tempo de execução em nível de usuário, tempo de execução em nível do *kernel* e tempo real da execução.

Contras:

Permite apenas a medição do tempo total de execução de uma aplicação, não sendo possível medir trechos de código;

Uma vez que tem o jiffy como base de tempo, a resolução é de 1 ms, não permitindo medir aplicações que demandem menor processamento.

TSC: Com a instrumentação do escalonador é possível determinar o momento em que o processo medido é alocado no processador e quando o mesmo é desalocado, podendo assim armazenar o contador do *clock* (TSC) nestas ocasiões.

Prós:

Método de medição baseado na base de tempo mais elementar do sistema (*clock* do processador).

Contras:

Para se obter valores legíveis de tempo é necessário considerar a frequência do *clock* do processador, necessitando obrigatoriamente que o valor do *clock* seja constante e conhecido durante a medição;

É necessário conhecer precisamente qual a frequência do processador, sendo que esta informação normalmente é representada por valores nominais (1 GHz, por exemplo).

Estampa de Tempo do *Kernel*: Baseado no mesmo princípio do método anterior, porém utiliza a estampa de tempo do *kernel* para registro do momento da troca de contexto do processador. O *kernel* obtém este valor utilizando o TSC para determinar o tempo decorrido desde a última interrupção do *timer* do sistema (jiffy), sendo capaz de prover

uma resolução de 1 μs. A função *do_gettimeofday()*, disponibilizada pela API do *kernel*, é capaz de retornar a estampa de tempo com resolução de microssegundos. Outra maneira de acessar este valor é através dos registros do ftrace, que anexa a cada mensagem a estampa de tempo.

Prós:

Não depende da constância e conhecimento do valor do *clock* do processador para a medição;

Permite portar a plataforma de medição para diferentes hardwares sem a necessidade de ajustes específicos para cada situação.

Contras:

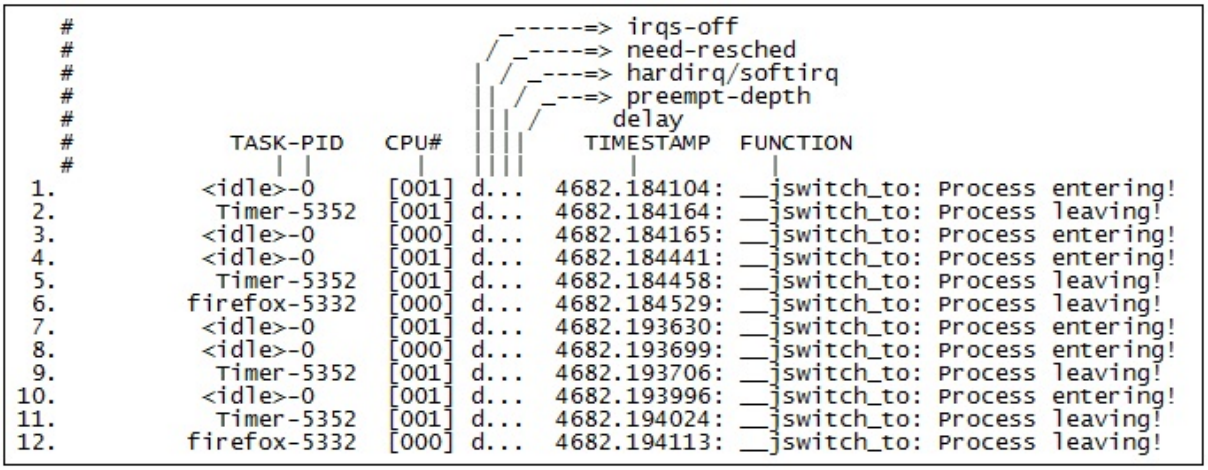
Possui menor resolução que o método baseado no TSC.

A medição através da estampa de tempo do *kernel* foi escolhida, pois apresenta uma exatidão adequada, além de possuir as características que melhor atendem a proposta da plataforma, como a portabilidade para outros hardwares. A exatidão do método de medição do tempo de processamento adotado é discutida na seção 5.1.2.

Inicialmente, a informação de tempo para cada intervenção do módulo que instrumenta o escalonador era armazenada através do kprintf (mensagens do *kernel* armazenadas em */var/log/messages*), porém quando havia grande atividade do escalonador era inserido considerável *overhead* no tempo de execução da aplicação em análise. O registro da estampa de tempo foi então migrado para o ftrace, que é uma ferramenta do *kernel* para análise de funções reentrantes, sendo otimizada para causar o menor *overhead* possível. O ftrace utiliza um buffer rotativo em memória RAM e, diferentemente do kprintf, não grava as mensagens no HD, nem as envia para o console.

A figura 26 mostra um exemplo das informações armazenadas pela instrumentação do escalonador, no qual o software Firefox, executado por dois núcleos do processador Atom, foi utilizado como aplicação em análise. As informações são armazenadas em colunas, sendo possível obter qual *thread* da aplicação está em execução (colunas TASK e PID), qual o núcleo está processando a tarefa (coluna CPU), qual a estampa de tempo no momento em que houve o chaveamento de contexto (coluna TIMESTAMP) e, por fim, a informação de que o registro se refere à retomada ou interrupção do processamento na CPU (coluna FUNCTION). Com estas informações, é possível analisar detalhadamente qual o fluxo do processamento das tarefas nos núcleos do processador.

Utilizando como exemplo as informações da figura 26, é possível determinar que duas *threads* da aplicação em análise (Firefox) foram executadas durante o período registrado (firefox



```
   #                                _-----=> irqs-off
   #                               / _----=> need-resched
   #                              | / _---=> hardirq/softirq
   #                              || / _--=> preempt-depth
   #                              ||| /     delay
   #           TASK-PID   CPU#    ||||    TIMESTAMP  FUNCTION
   #              | |       |     ||||       |         |
 1.          <idle>-0     [001] d...  4682.184104: __jswitch_to: Process entering!
 2.           Timer-5352  [001] d...  4682.184164: __jswitch_to: Process leaving!
 3.          <idle>-0     [000] d...  4682.184165: __jswitch_to: Process entering!
 4.          <idle>-0     [001] d...  4682.184441: __jswitch_to: Process entering!
 5.           Timer-5352  [001] d...  4682.184458: __jswitch_to: Process leaving!
 6.         firefox-5332  [000] d...  4682.184529: __jswitch_to: Process leaving!
 7.          <idle>-0     [001] d...  4682.193630: __jswitch_to: Process entering!
 8.          <idle>-0     [000] d...  4682.193699: __jswitch_to: Process entering!
 9.           Timer-5352  [001] d...  4682.193706: __jswitch_to: Process leaving!
10.          <idle>-0     [001] d...  4682.193996: __jswitch_to: Process entering!
11.           Timer-5352  [001] d...  4682.194024: __jswitch_to: Process leaving!
12.         firefox-5332  [000] d...  4682.194113: __jswitch_to: Process leaving!
```

Figura 26: Exemplo de informações armazenadas pela instrumentação do escalonador através do ftrace. Foi utilizado o software Firefox como aplicação em análise.

Fonte: Autoria própria.

- PID 5332 e timer - PID 5352), pois sempre que ocorre uma retirada da tarefa pelo escalonador, é armazenada sua identificação. Também é possível obter o tempo de processamento em cada alocação das *threads*; por exemplo, na linha 8 da figura, registrou-se a alocação da *thread* firefox no núcleo 0 do processador, sendo esta desalocada no registro da linha 12, o que resulta em um tempo de processamento de 414 μs.

Para simplificar o acesso às funcionalidades da instrumentação do *kernel*, foi desenvolvida uma API capaz de configurar o TGID a ser medido e indicar o início e o fim do trecho de código a ser analisado. As funções disponibilizadas para a aplicação em análise são as seguintes:

set_pid: Configura o TGID da aplicação que será medida. O TGID deve ser configurado antes do início da medição.

set_mypid: Configura o TGID para medir a aplicação que chamou esta função.

start_measurement: Inicia a medição. Usado para indicar o início do trecho de código a ser medido.

start_selfmeasurement: Inicia a medição da aplicação que chamou esta função. Usado para automaticamente configurar o TGID da aplicação que a chamou e iniciar a medição.

stop_measurement: Para a medição em curso. Usado para indicar o final do trecho de código a ser medido.

Por fim, foi desenvolvida uma aplicação para realizar a leitura das amostras obtidas pelo DAQ e compilar as informações de tempo armazenadas pela instrumentação do escalonador.

3.2.4 PROCEDIMENTO PARA MEDIÇÃO

Conforme fluxograma mostrado na figura 27, o processo de medição possui duas etapas: coleta de dados e análise posterior. Na primeira etapa, a aplicação a ser medida é instrumentada para chamar as rotinas da API disponibilizada imediatamente antes e após o trecho de código a ser analisado. O apêndice A mostra um exemplo de código instrumentado. A coleta de dados é iniciada com a execução da aplicação de leitura das amostras do DAQ (comando *sudo daq*, executado no console do Linux). Esta aplicação contabiliza as medidas de consumo realizadas pelo DAQ durante toda a fase de coleta de dados. Na sequência, o código instrumentado é executado.

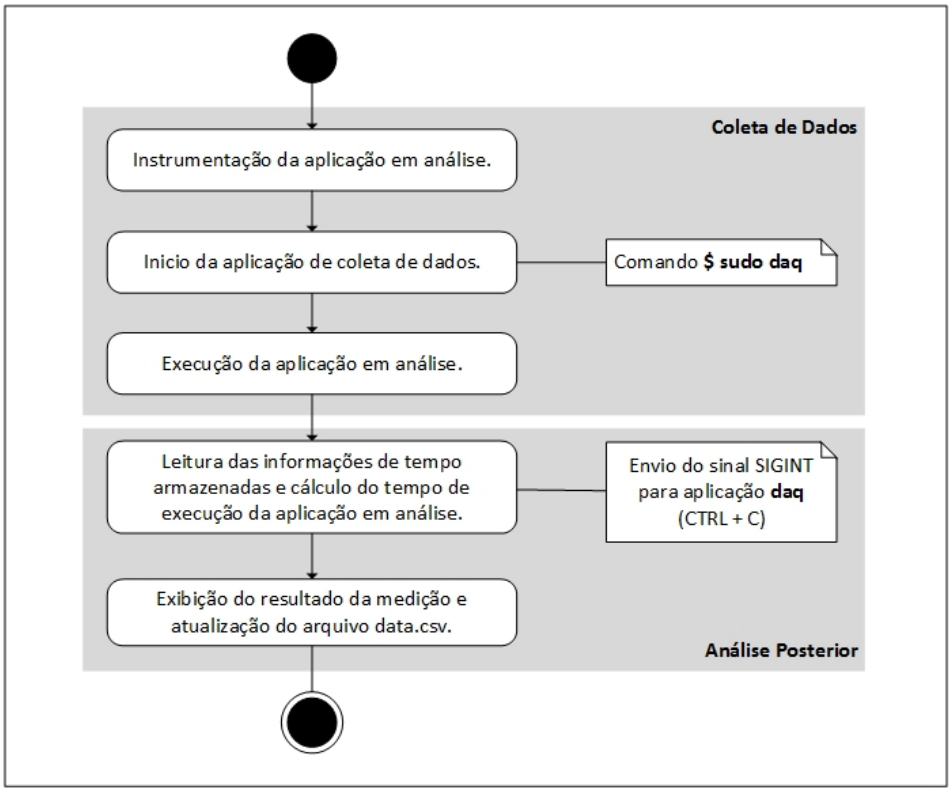


Figura 27: Fluxograma do procedimento de medição de consumo através da plataforma proposta. São indicadas as duas etapas da medição, coleta de dados e análise posterior.

Fonte: Autoria própria.

Finalizada a execução da aplicação em análise, inicia-se a etapa de análise dos dados ao enviar a um sinal *SIGINT* (CTRL + C) à aplicação de coleta de dados (*daq*). Nesta etapa, as

informações de tempo armazenadas através do ftrace serão analisadas e será calculado o tempo de execução da aplicação medida. Todos os resultados da medição são exibidos no console em que a aplicação de coleta estava em execução, além de serem salvos automaticamente como uma nova linha de um arquivo em forma de tabela (*~/data.csv*).

3.2.5 MEDIDAS OBTIDAS ATRAVÉS DA PLATAFORMA

Três principais medidas são obtidas através da plataforma desenvolvida: potência média, tempo de processamento e energia consumida. Durante a fase de coleta de dados, cada nova amostra de consumo obtida é validada através do sinal de sincronismo, sendo uma amostra considerada para a medição sempre que o sinal de sincronismo está ativo. Este processo é realizado para todas as linhas de medição ativas. Para cada amostra válida, o valor medido é convertido para potência instantânea através da fórmula 9.

$$P = \frac{V_{DAQ}}{R}.V_{linha} \tag{9}$$

em que P é a potência instantânea, V_{DAQ} é a tensão sobre o sensor de corrente (resistor) medida pelo DAQ, R é a resistência do sensor de corrente e V_{linha} é a tensão da linha de alimentação instrumentada.

A potência instantânea é então utilizada para atualizar os valores de consumo máximo, mínimo e médio da aplicação em análise. A potência média é obtida através da uma média simples entre todas as amostras válidas durante uma medição.

Ao término da execução da aplicação em análise, os dados armazenados pela instrumentação do escalonador são computados em busca de estampas de tempo associadas ao início ou ao fim das alocações da aplicação no processador. Isto permite sumarizar o tempo total de processamento da aplicação, o tempo de processamento por núcleo do processador, o tempo linear de processamento e o tempo real da execução. O tempo linear de processamento indica o tempo em que ao menos uma *thread* da aplicação estava em execução no processador, enquanto o tempo total de processamento da aplicação indica a soma dos tempos de processamento de todas as *threads* da aplicação. O tempo real é o tempo decorrido entre o início do processamento da aplicação e o seu fim, independentemente do tempo em que a aplicação não estava alocada no processador. A figura 28 mostra os tempos de processamento medidos pela plataforma, tendo como exemplo uma aplicação com duas *threads* executada por um processador, quando configurado para utilizar apenas um núcleo (acima) e dois núcleos (abaixo).

O processador Atom N2800 presente na plataforma possui dois núcleos com uma

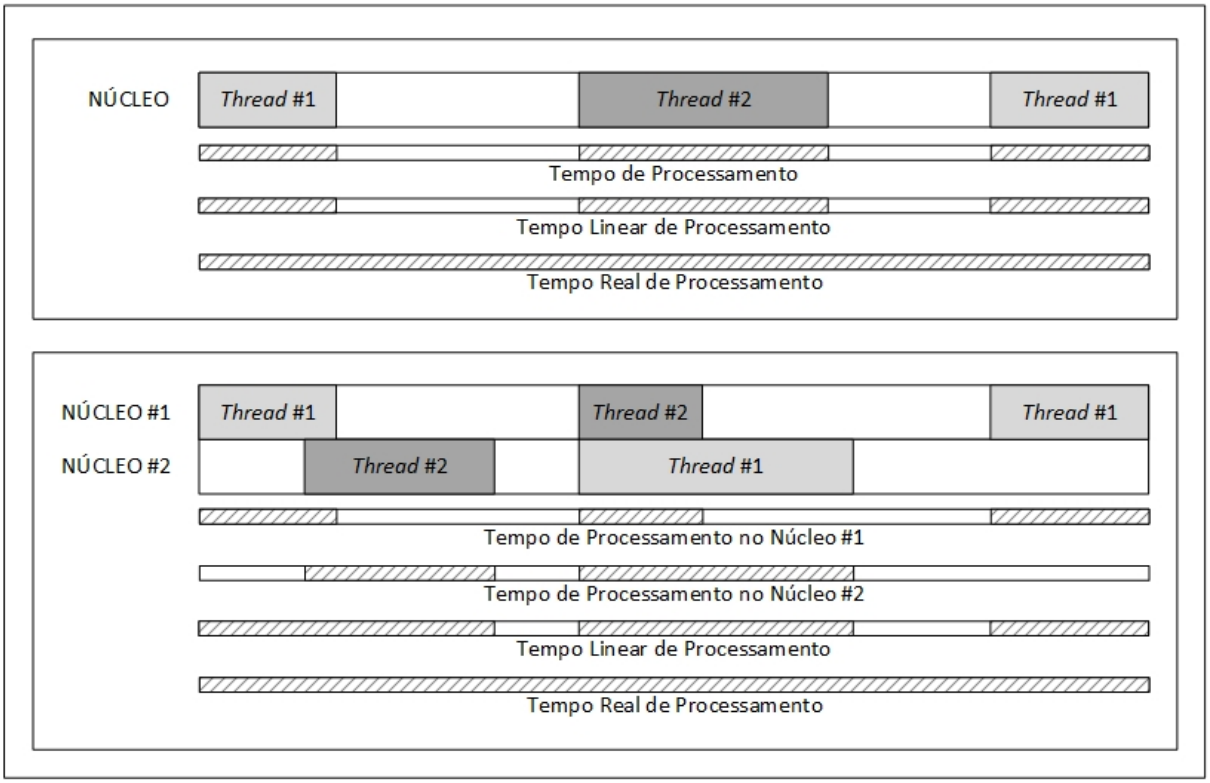


Figura 28: Tempos de processamento medidos pela plataforma desenvolvida para uma aplicação com duas *threads* executadas por um único núcleo do processador (acima) e por dois núcleos (abaixo).

Fonte: Autoria própria.

hyper-thread cada. A tecnologia de *Hyper-threading* da Intel utiliza de forma mais eficiente os recursos do processador, permitindo que múltiplas *threads* sejam executadas em um mesmo núcleo, não necessitando a leitura do contexto de cada uma delas a partir de uma memória externa. Isso resulta em considerável aumento da capacidade de processamento (TIAN et al., 2003). Uma *hyper-thread* do processador é tratada pelo sistema operacional da mesma forma que um núcleo físico. Com isso, quando essa tecnologia está habilitada, não é possível determinar quando efetivamente as instruções das aplicações em análise estão sendo executadas. Por esse motivo, através da BIOS da placa-mãe Atom, foi configurado para que o processador não utilize *hyper-threads* em sua operação. Através da BIOS também é possível optar pela utilização de um ou dois núcleos do processador durante as medições.

Considerando que a frequência de amostragem do consumo é constante, a energia total consumida pela aplicação é calculada através do produto do tempo linear de processamento pela potência média. Ao final de uma medição, os seguintes valores são computados e exibidos no console da plataforma: número total de vezes que as *threads* da aplicação em análise foram alocadas no processador (*total process entrances*); tempo de processamento por núcleo do

processador (*process time on CORE[N]*, em que N é o índice do núcleo); tempo total de processamento (*total process time*); tempo linear de processamento (*total linear process time*); tempo real de processamento (*real time*); o número total de amostras obtidas pelo DAQ durante a coleta de dados (*total samples*); o número de amostras válidas, obtidas quando o sinal de sincronismo estava acionado (*total valid samples*); a potência média por placa (*average power*); a potência máxima por placa medida (*max power*); a potência mínima por placa medida (*min power*); e, por fim, a energia consumida pela aplicação em análise (*energy consumption*). As principais medidas também são automaticamente salvas em um arquivo em formato de tabela. Além destas medidas, é computado o tempo total desde a última vez que o TGID foi configurado na instrumentação do escalonador; desta forma, é possível medir trechos de códigos iterativos, no quais as funções para início e fim da medição são chamadas em cada ciclo da iteração. A figura 29 mostra um exemplo de resultado obtido através da plataforma de medição desenvolvida.

3.2.6 VALIDAÇÃO DO MÉTODO DE SINCRONISMO

O principal conceito utilizado pelo sincronismo empregado na plataforma de medição é a instrumentação do escalonador do sistema operacional capaz de identificar o momento em que a aplicação em análise foi alocada para processamento. Considerando que o *kernel* utilizado possui seu código fonte disponível, através da inspeção do algoritmo do escalonador é possível garantir que a instrumentação realizada é processada no momento correto. Apesar disto, um experimento foi elaborado para claramente visualizar se o resultado obtido pela implementação corresponde ao comportamento de uma aplicação de funcionamento conhecido. A aplicação de teste criada possui apenas a chamada da função *sleep()*, em que sistema operacional se encarrega de não alocar a aplicação para processamento até que se expire o período programado. A figura 30 mostra as informações resultantes para uma período de 10 segundos.

Analisando o resultado obtido, é possível verificar que a aplicação foi alocada apenas duas vezes no processador, uma no início da medição até a chamada da função *sleep()* (linhas 1 e 2 da figura 30) e a segunda, após decorrido o período estipulado (linhas 3 e 4). Também é possível notar que o intervalo entre as duas execuções corresponde aos 10 segundos programados.

Outra validação necessária para o sincronismo desenvolvido refere-se à latência do sinal gerado através da porta paralela da placa-mãe Atom. Um experimento foi então desenvolvido para verificar se o tempo entre o acionamento do sinal realizado pela instrumentação do escalonador e a sua efetiva medição pelo DAQ é suficientemente curto, de

```
Power measures stopped!
Total process entrances: 8827
Process time on CORE[0] (us): 4762194
Process time on CORE[1] (us): 2700294
Process time on CORE[2] (us): 3591758
Process time on CORE[3] (us): 1975408
Total process time (us): 13029654
Total linear process time (us): 11177597
Real time (us): 14223507

Total linear process time since last TGID set: 11177597

Atom Motherboard
Total samples: 10101
 Total valid samples: 4630
 Average Power (W): 20.3894
 Max Power (W): 30.5703
 Min Power (W): 17.7862
Energy Consumption (J): 227.904

FPGA Board
Total samples: 10101
 Total valid samples: 4630
 Average Power (W): 3.14349
 Max Power (W): 5.86597
 Min Power (W): 2.50913
Energy Consumption (J): 35.1367
```

Channel Name [Channel 1]	Energy (J) [Channel 1]	Avarage Power (W) [Channel 1]	Maximum Power (W) [Channel 1]	Minimum Power (W) [Channel 1]	Linear Time (us) [Channel 1]
Atom Motherboard	227,904	20,3894	30,5703	17,7862	11177597

Channel Name [Channel 2]	Energy (J) [Channel 2]	Avarage Power (W) [Channel 2]	Maximum Power (W) [Channel 2]	Minimum Power (W) [Channel 2]	Linear Time (us) [Channel 2]
FPGA Board	35,1367	3,14349	5,86597	2,50913	11177597

Figura 29: Exemplo de resultado obtido através da plataforma de medição desenvolvida. Acima, resultado exibido no console após a finalização da aplicação de coleta de dados; abaixo, o resultado automaticamente salvo em um arquivo em forma de tabela (data.csv).

Fonte: Autoria própria.

```
  #                              _-----=> irqs-off
  #                             / _----=> need-resched
  #                            | / _---=> hardirq/softirq
  #                            || / _--=> preempt-depth
  #                            ||| /     delay
  #           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
  #              | |       |   ||||       |         |
1.         example-4700  [000] ....  3079.343658: reset_stat: Process running!
2.         example-4700  [000] d...  3079.343679: __jswitch_to: Process leaving!
3.          <idle>-0     [000] d...  3089.343848: __jswitch_to: Process entering!
4.         example-4700  [000] ....  3089.343906: reset_stat: Process stopping!
```

Figura 30: Registros da instrumentação do escalonador para função *Sleep* validando que o monitoramento corresponde ao comportamento esperado.

Fonte: Autoria própria.

forma a permitir a correta atribuição das variações de consumo causadas pela aplicação em análise. O experimento consiste em medir uma aplicação de teste capaz de gerar um nível de

consumo que se diferencie do consumo da plataforma executando as demais tarefas do sistema. Considerando que não há controle sobre os demais processos ativos no sistema operacional e o consumo demandado por eles, a maneira mais fácil de obter uma variação no nível de consumo é criar uma carga para o sistema que cause o menor consumo possível.

A aplicação de teste desenvolvida possui um único laço sobre a operação NOP, não demandando acesso aos componentes externos ao processador, nem execução de códigos externos à memória cache. A medição foi realizada com apenas um núcleo do processador ativado, visando assim garantir que outras aplicações não estejam em execução durante o evento monitorado. Para gerar carga que exija elevado consumo, diferenciando-se da aplicação de teste, foram abertos e mantidos em execução outros programas, como, por exemplo, o ambiente de desenvolvimento NetBeans 7.4, compilando um grande projeto. Um segundo computador conectado ao DAQ foi utilizado para ler e armazenar todas as amostras obtidas do sinal de sincronismo e do consumo da placa-mãe. A taxa de amostragem utilizada foi de 5 ksps.

O funcionamento do sinal de sincronismo pode ser verificado através dos gráficos da figura 31, que mostra os dados lidos pelo DAQ durante a execução da aplicação de teste. O gráfico 1 mostra o resultado obtido durante cerca de 20 ms de execução. Nele é possível notar que sempre que a aplicação em teste é alocada para execução, indicada pelo acionamento do sinal de sincronismo, há uma redução do consumo da placa-mãe. Para validar que a latência do sinal de sincronismo não interfere na medição, o gráfico 2 destaca a transição do sinal (amostras 19 a 26 do gráfico 1), permitindo observar que a mudança do estado é medida exatamente na mesma amostra em que começa a redução do consumo. Desta forma, confirma-se que o sinal de sincronismo utilizado é adequado para a taxa máxima de amostragem de 5 ksps permitida pela plataforma de medição.

3.2.7 COMPARAÇÃO COM AS PLATAFORMA DISPONÍVEIS NA LITERATURA

O quadro 5 mostra uma comparação entre a plataforma de medição desenvolvida e as plataformas apresentadas na seção 2.2.3. A coluna “Taxa de Amostragem” indica a frequência nominal da amostragem documentada ou estimada através do dispositivo utilizado para medição. Uma maior frequência de amostragem permite uma melhor resolução para medição de pequenos trechos de código, desde que exista sincronismo entre a aplicação em teste e o sistema de aquisição (coluna “Medição Sincronizada”). O termo “flexível” indica que o hardware utilizado para a amostragem do consumo pode ser facilmente alterado. Para a plataforma criada, outros DAQs da National Instrument, com melhor resolução ou taxa de amostragem, são diretamente compatíveis. Os modelos atualmente disponíveis permitem até

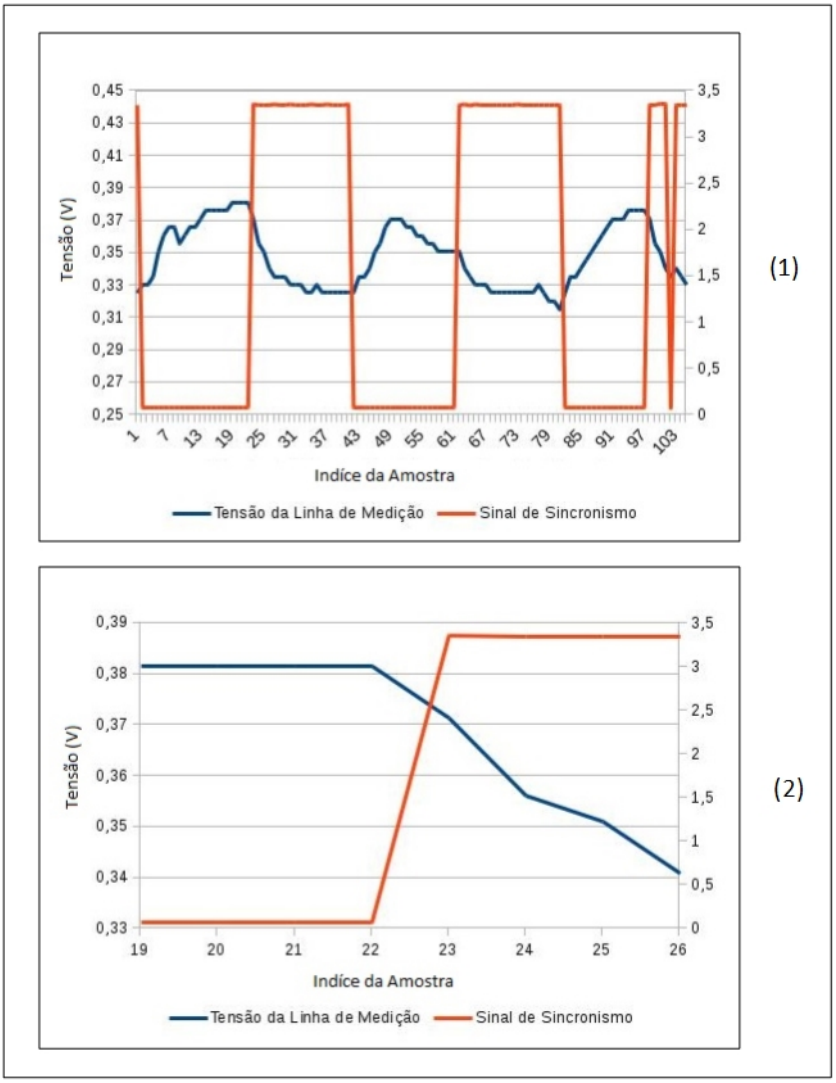


Figura 31: Gráfico com as amostras de consumo obtidas para a aplicação de validação do sinal de sincronismo. Acima, é mostrado gráfico com aproximadamente 20ms de amostras obtidas durante a execução da aplicação. Abaixo, é destacada a região de interesse para a validação do sinal de sincronismo.

Fonte: Autoria própria.

2 Msps com resolução de 16 bits.

A plataforma desenvolvida, o PowerScope e a plataforma que considera os acessos aos dispositivos de I/O são as únicas capazes de associar cada amostra de consumo com a tarefa que está alocada no processador. Esta funcionalidade é indicada na coluna "Medição Individual da Tarefa". A coluna "Medição do Tempo de Processamento" representa a capacidade de medir precisamente o tempo de processamento de uma seção de código, descontando as intervenções do escalonador do sistema operacional, permitindo assim obter a energia consumida pela aplicação. Finalmente, a coluna "SW vs. HW" indica o suporte a aplicações com processamento

Plataforma de Medição	Taxa de Amostragem (sps)	Medição Individual da Tarefa	Medição Sincronizada	Medição do Tempo de Processamento	SW vs. HW
PowerScope	625 (flexível)	✓	✓		
PowerPack	250k (flexível)		✓		
PowerMon2	3k				
PAPI	1		✓	✓	
PowerInsight	1k				
Acesso a I/O	50k (flexível)	✓	✓		
LEAP	10k (flexível)		✓		
Desenvolvida	5k (flexível)	✓	✓	✓	✓

Quadro 5: Comparação entre a plataforma de medição desenvolvida e as demais plataformas de medição do consumo de aplicações em software disponíveis na literatura.

Fonte: Autoria própria.

híbrido. Como indicado no quadro 5, a plataforma criada é a única que incluí todas as características necessárias para medir com precisão o consumo energético de aplicações em software e comparar com aplicações equivalentes em hardware.

3.2.7.1 COMPARAÇÃO COM A MEDIÇÃO OBTIDA PELO LEAP

O LEAP, conforme apresentado na seção 2.2.3.8, é uma das mais avançadas plataformas de medição de consumo. O laboratório ASCENT, onde o LEAP foi desenvolvido, disponibiliza esta ferramenta para que seja acessada remotamente e utilizada para a medição do consumo energético de algoritmos computacionais (ASCENT, 2013). Desta forma, o LEAP pode ser utilizado como parâmetro de comparação para a plataforma desenvolvida, permitindo situar a qualidade do método de medição criado perante uma solução bem aceita pela comunidade científica.

Para a comparação entre a plataforma desenvolvida e o LEAP, foi realizada a medição de uma mesma aplicação de teste nas duas plataformas. A aplicação de teste é uma implementação do *benchmark* Linpack, que analisa a capacidade de processamento dos sistemas através de funções de álgebra linear densa. O Linpack é largamente utilizado para avaliar o desempenho e a eficiência de novas arquiteturas de computadores (HEINECKE et al., 2013).

Em cada uma das plataformas, as medições foram repetidas 100 vezes consecutivas. O resultado de cada iteração foi armazenado, permitindo uma avaliação estatística dos valores medidos. Como as duas plataformas avaliadas possuem hardwares distintos e instrumentam de

forma diferente as placas, não é possível uma comparação direta entre os valores de consumo; contudo, é possível analisar a dispersão das medidas realizadas em cada plataforma. Desta forma, compara-se a qualidade da medição proporcionada pelas duas ferramentas, sendo melhor quanto menor for a incerteza das medições. Os resultados obtidos serão mostrados na seção 5.2.

4 APLICAÇÃO ANALISADA

Este capítulo contempla o desenvolvimento da aplicação na qual a plataforma criada foi utilizada como ferramenta de análise. Primeiramente, são abordados os conceitos que embasam o desenvolvimento de um sistema de detecção de intrusão. Na sequência, é descrita a implementação da aplicação para detecção de intrusão de rede por ataques do tipo *probing* com classificação em software e hardware através de Árvore de Decisão. Por fim, é apresentado o procedimento de medição de consumo utilizado.

4.1 DETECÇÃO DE INTRUSÃO

A internet tem sofrido substanciais mudanças nos últimos anos. Tendências claras indicam um rápido aumento no número de dispositivos conectados e uma crescente preocupação com segurança da informação e privacidade. Estes fatos, aliados à considerável parcela de dispositivos móveis existentes, têm despertado grande interesse pelo desenvolvimento de soluções energeticamente eficientes para a segurança dos dispositivos conectados à grande rede. Pesquisas recentes procuraram aperfeiçoar a eficiência energética de algoritmos computacionais movendo parte do processamento realizado em software para hardwares dedicados. Estudos anteriores ao presente trabalho indicaram que uma implementação em hardware de um algoritmo de detecção de intrusão de rede pode reduzir em 99,97% a energia consumida por uma aplicação equivalente em software, além de alcançar um desempenho 15 vezes maior (FRANCA et al., 2014b).

Devido ao grande número de ameaças, dispositivos conectados à internet devem estar protegidos de forma autônoma e em tempo real. Uma forma comum de se iniciar um ataque é explorar o sistema alvo à procura de vulnerabilidades. Esta técnica, denominada *probing*, realiza uma varredura pela rede, procurando por serviços vulneráveis, através da exploração de todas as portas abertas. Uma forma eficiente de se aumentar a segurança de um sistema é detectar o ataque nesta fase inicial.

Sistemas de Detecção de Intrusão (IDS - *Intrusion Detection System*) são aplicações

de segurança que analisam o tráfego da rede e classificam cada pacote como normal ou ataque. Conforme o funcionamento do algoritmo de classificação, IDSs podem ser classificados em duas categorias: baseado em assinatura ou baseado em anomalia (GOGOI et al., 2012).

Sistemas baseados em assinatura examinam cada pacote recebido da rede em procura de um padrão de ataque conhecido. Este método possui baixa taxa de falsos positivos; contudo, é incapaz de identificar novos tipos de ataques. Além disso, é necessária a constante atualização do banco de assinaturas, aumentando linearmente o esforço computacional.

Sistemas baseados em anomalia, em contrapartida, analisam o tráfego da rede e identificam quando este desvia de um comportamento normal. Para isso, um modelo do comportamento normal da rede é criado a partir de entradas conhecidas. Um alarme é gerado quando as características da rede divergem deste modelo. Esta abordagem pode causar falsos alarmes, porém é capaz de identificar ataques ainda não mapeados. Uma vez que novos ataques aparecem diariamente, esta é uma vantagem relevante.

4.1.1 IDS BASEADO EM ANOMALIA

Sistemas de detecção baseados em anomalia são normalmente implementados em duas etapas subsequentes: extração de características e classificação dos pacotes (TSAI et al., 2009). Na extração de características, cada pacote recebido da rede é analisado e uma série de atributos são extraídos. Estes valores são agrupados em um vetor de características, utilizado como entrada para o classificador.

O vetor de características compreende dois tipos de atributos: atributos não dependentes do estado, extraídos diretamente do cabeçalho do pacote, e atributos dependentes do estado, que consideram os pacotes anteriormente recebidos. Para este segundo grupo, os atributos normalmente são contadores, os quais contam, por exemplo, o número de bytes enviados do cliente para o servidor durante um período pré-definido de tempo (GOGOI et al., 2012).

O estágio de classificação é construído, em sua maioria, usando técnicas de Aprendizagem de Máquina (ML - *Machine Larning*). Estas técnicas são capazes de descobrir modelos ocultos em um conjunto de dados de treinamento. Os algoritmos ML também são conhecidos por sua adaptabilidade, tolerância a falhas e imunidade a ruídos (informações que confundem a classificação) (WU; BANZHAF, 2010).

Aplicações de Aprendizagem de Máquina são construídas baseadas em funções que mapeiam dados de entrada em categorias conhecidas; tais funções são denominadas

classificadores. A criação e o uso dos classificadores são frequentemente realizados em duas etapas distintas. A primeira etapa, ou treinamento, utiliza uma base de dados de treinamento e um algoritmo de análise para criar o modelo de ataque. Em sistemas de detecção de intrusão, o vetor de características é a entrada para o classificador e a saída é a determinação entre duas classes: tráfego normal ou ataque (BRUGGER, 2004).

A segunda etapa, executada em tempo real, é a classificação, em que o vetor de características é processado pelo classificador. Para IDS, os classificadores mais comuns são Máquina de Vetores de Suporte (SVM - *Support Vector Machine*), Árvore de Decisão (DT - *Decision Tree*) e K-Vizinhos mais Próximos (KNN - *K-Nearest Neighbors*) (TSAI et al., 2009). As Árvores de Decisão são adequadas para a detecção de intrusão, pois os dados de entrada podem ser discretos ou contínuos, o algoritmo não é computacionalmente intensivo e pode ser facilmente implementado em sistemas de tempo real.

As Árvore de Decisão são constituídas por um conjunto de regras "se-então", em que a classificação ocorre percorrendo a árvore da raiz até atingir uma folha, a qual possui uma classe associada (normal ou ataque). Cada nó não terminal da árvore especifica o teste para um determinado atributo. O algoritmo C4.5 é uma escolha comum na construção do modelo de DTs (MITCHELL, 1997). A figura 32 mostra um exemplo de modelo criado a partir deste algoritmo.

```
=== Classifier model (full training set) ===

J48 pruned tree
------------------
protocol_type = tcp
|   ip_len <= 104
|   |   tcp_fpush = 0: attack
|   |   tcp_fpush = 1
|   |   |   tcp_ffyn = 0: normal
|   |   |   tcp_ffyn = 1
|   |   |   |   tcp_ack <= 2825023790
|   |   |   |   |   tcp_seq <= 400185759: normal
|   |   |   |   |   tcp_seq > 400185759: attack
|   ip_len > 104
|   |   flag = REJ: normal
|   |   flag = RSTR: anomaly
|   ...
protocol_type = udp
|   ...
```

Figura 32: Trecho de um modelo para Árvore de Decisão gerado através do algoritmo C4.5. Fonte: (FRANCA et al., 2014b).

4.2 DESENVOLVIMENTO DA APLICAÇÃO PARA DETECÇÃO DE INTRUSÃO

A arquitetura da figura 33 foi desenvolvida para analisar o perfil do consumo e guiar a busca pela eficiência energética em aplicações para detecção de intrusão. O processamento se inicia quando um pacote é recebido pelo controlador Ethernet da placa-mãe e repassado para o processador. A software em execução no processador Atom é responsável por capturar os pacotes recebidos da rede através da biblioteca Libpcap, que permite acesso estruturado aos dados do pacote (MCCANNE; JACOBSON, 1993). Estes dados são utilizados pelo módulo extrator de características para criar o vetor de atributos que será submetido ao classificador. Até este ponto, todo o processamento é realizado apenas pelo software. No classificador, uma das implementações é escolhida (software ou hardware). Quando a classificação é realizada em hardware, a aplicações utiliza a FPGA como coprocessador.

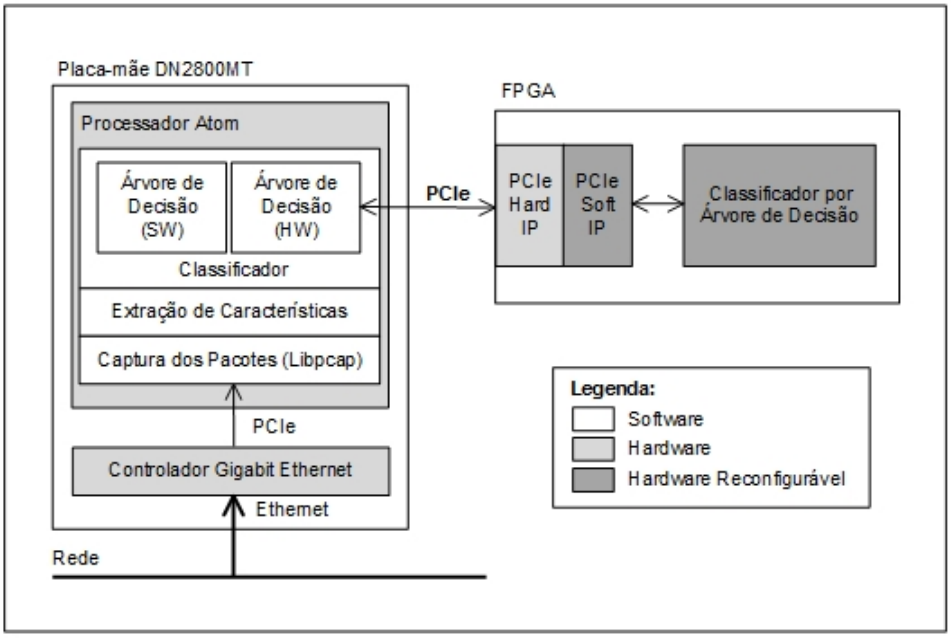


Figura 33: Arquitetura da aplicação para detecção de intrusão analisada através da plataforma de medição desenvolvida.

Fonte: (CEMIN et al., 2014).

4.2.1 ALGORITMO DE CLASSIFICAÇÃO

A implementação de um classificador é realizada em três fases: aquisição da base de dados, criação do modelo e desenvolvimento do código. A base de dados utilizada foi criada utilizando uma ferramenta de auditoria, denominada Nessus, para produzir ataques do tipo *probing* e ferramentas diversas para gerar tráfego normal (SSH, SNMP, IMAP, and HTTP). Cada

vetor da base de dados possui 50 atributos extraídos dos pacotes de rede gerados. A escolha destes atributos baseou-se na literatura e visou características representativas para detectar os ataques (MCCANNE; JACOBSON, 1993).

A base de dados resultante possui 9.431.075 pacotes, dos quais 14.962 são ataques. Para obter uma base de treinamento balanceada, selecionaram-se 3.740 vetores referentes a pacotes normais e 3.740 vetores referentes a ataques. As instâncias restantes foram usadas para criar uma segunda base de dados, também balanceada e com 7.480 elementos, utilizada para testar o modelo criado.

Antes de criar o modelo do classificador, utilizou-se um algoritmo genético para selecionar, do total de 50 atributos, os atributos mais relevantes para a classificação. A melhor exatidão foi obtida para um subconjunto de 11 atributos. Desta forma, apenas os seguintes atributos são utilizados para a criação do modelo e posteriormente utilizados durante a fase de classificação: *checksum* do cabeçalho IP (IP_CHECKSUM); porta de origem do cabeçalho TCP (TCP_SPORT); porta de destino do cabeçalho TCP (TCP_DPORT); *flag* "não fragmente" (IP_DF); *flag* RST (TCP_FRST); *flag* PUSH (TCP_FPUSH); *flag* ACK (TCP_FACK); e quatro atributos baseados nos últimos 2 segundos de comunicação: número de pacotes enviados do servidor para o mesmo serviço do cliente (COUNT_SERV_S2C), número de pacotes enviados do servidor para o cliente com o *flag* ACK setado (NUM_ACK_S2C), número de pacotes enviados do cliente para o servidor com o *flag* ACK setado (NUM_ACK_C2S) e número de pacotes enviados do cliente para o servidor com o *flag* SYN setado (NUM_SYN_C2S).

Finalmente, aplicou-se o algoritmo C4.5 à base de dados, resultando em uma Árvore de Decisão com 28 folhas e 24 nós. O modelo criado apresentou uma exatidão de 99,87% quando validado com a base de teste.

A saída do algoritmo C4.5 é uma representação textual da árvore. Então, criou-se um *script* para converter este modelo para duas linguagem: VHDL e C++. O VHDL é utilizado para sintetizar o circuito do classificador na FPGA e o código C++ usado para o classificador em software.

4.2.2 HARDWARE DO COPROCESSADOR

A figura 34 mostra em detalhe o hardware implementado na FPGA. O coprocessador opera como um periférico na interface PCIe, em que os atributos fluem do Atom para a FPGA e os resultados da classificação no sentido oposto.

O periférico possui duas partes principais: um banco de registradores e o

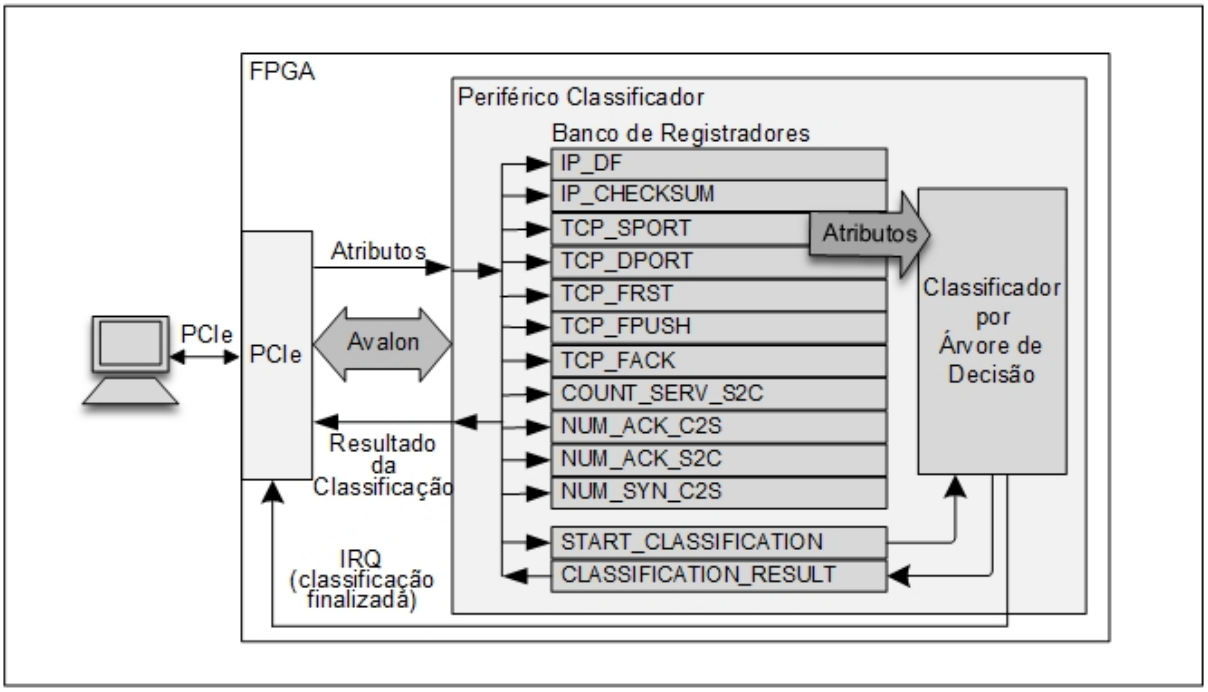


Figura 34: Hardware do coprocessador para classificação por Árvore de Decisão em hardware. Os nomes dos registradores são conforme codificação da aplicação.

Fonte: Adaptada de (FRANCA et al., 2014a).

classificador por Árvore de Decisão. Para controlar o periférico, o software manipula os valores dos 11 atributos no banco de registradores e então escreve no registrador START_CLASSIFICATION. Quando o resultado está disponível, o classificador o escreve no registrador CLASSIFICATION_RESULT e gera uma interrupção, enviada através da PCIe, para o software. Finalmente, o software lê o resultado da classificação presente no registrador.

Além de capturar pacotes recebidos da rede, a biblioteca Libpcap permite utilizar um arquivo em memória como entrada de dados. Utilizou-se então os 9,4 milhões de pacotes da base de dados coletada para garantir que o classificador em software e hardware são totalmente equivalentes e obtêm o mesmo resultado.

4.2.3 MEDIÇÕES REALIZADAS ATRAVÉS DA PLATAFORMA DESENVOLVIDA

Como parte do processamento foi transferido do processador Atom para a FPGA, é necessário medir o consumo energético nos dois cenários: processamento apenas em software e processamento com auxílio do coprocessador em hardware. Com este objetivo, duas configurações foram utilizadas. Na configuração utilizada para a medição do consumo da aplicação apenas em software, somente a placa-mãe Atom é utilizada e a placa com a FPGA foi

desconectada do barramento PCIe. Na segunda configuração, a FPGA é conectada ao Atom.

Nas medições do consumo energético, os dados são lidos de um arquivo previamente armazenado no HD, ao invés da interface de rede. Com isso, o sistema classifica os pacotes o mais rápido possível, tornando mais evidentes as diferenças de comportamento entre as soluções. Em cada medição, um total de 10.000 pacotes são processados e os mesmos pacotes são utilizados para os dois cenários avaliados.

Com o objetivo de identificar o consumo demandado em cada etapa da classificação (leitura do pacote, extração de características e classificação), foram medidos os consumos para o processamento completo dos pacotes, para a leitura dos pacotes e extração de características e, por fim, apenas para a leitura dos pacotes. Desta forma, o consumo de cada etapa é obtido subtraindo os valores de consumo para o processamento parcial dos pacotes do consumo de todo o processo de classificação. As medições dos consumos parciais foram obtidas desabilitando individualmente cada etapa do processamento no software da aplicação. Este mesmo método foi aplicado para a classificação em software e em hardware. Na classificação em hardware, adicionalmente, foram medidos os tempos de processamento para a escrita do vetor da característica e a leitura do resultado, através do barramento PCIe, e o tempo para o recebimento da interrupção indicando o final da classificação pelo hardware. Os resultados serão mostrados na seção 5.3.3

5 RESULTADOS E DISCUSSÕES

Neste capítulo são apresentados os resultados obtidos com o desenvolvimento da plataforma de medição. Primeiramente, são discutidas a resolução e a incerteza da medição de consumo proporcionada pela plataforma. Para melhor situar este trabalho perante as demais soluções disponíveis, são comparadas as medições obtidas através da plataforma criada e através da plataforma LEAP (detalhada na seção 2.2.3.8). Na sequência, são mostrados os recursos da FPGA utilizados pelo coprocessador desenvolvido para a classificação por Árvore de Decisão em hardware. Por fim, é apresentado e analisado o consumo energético da aplicação para detecção de intrusão de rede com classificação em software e hardware.

5.1 CARACTERÍSTICAS DA PLATAFORMA DESENVOLVIDA

A resolução e a incerteza da medição são características essenciais para caracterizar um instrumento de medição. A resolução, no caso da plataforma criada, indica qual o menor consumo que é passível de medição. Considerando que o objetivo da ferramenta é possibilitar a medição de trechos de código, quanto menor a resolução melhor se caracterizar o perfil de consumo das etapas de processamento de uma aplicação. A incerteza, por sua vez, mostra qual o intervalo de confiança que se tem para uma determinada medida, menores incertezas permitem identificar mais claramente a diferença de consumo entre pequenos trechos de código. Nesta seção são discutidos a obtenção e os valores da resolução e da incerteza para a plataforma de medição desenvolvida.

Nesta seção também será mostrado o consumo energético demandado pelo suporte à interface PCIe. Como este valor é medido juntamente com a aplicação em análise, sua grandeza indica o quanto uma solução é influenciada pela comunicação entre software e o coprocessador em hardware.

5.1.1 RESOLUÇÃO DA MEDIDA DE CONSUMO

A resolução da medida de consumo obtida pela plataforma desenvolvida é derivada das duas medidas básicas realizadas: potência média e tempo de processamento. Como discutido na seção 3.2.3, o tempo de processamento é obtido através da diferença entre as estampas de tempo do *kernel* lidas nos momentos em que a aplicação em análise é alocada e retirada do processador pelo escalonador. Considerando que a estampa de tempo do *kernel* é informada com resolução de 1 μs, esta também é a resolução da medição do tempo de processamento da plataforma.

Para determinar a resolução da medida de potência, são consideradas as características do amostrador (DAQ) e os parâmetros da instrumentação feita nas placas da plataforma. A tabela 1 mostra as características de interesse para o cálculo da resolução da potência.

Tabela 1: Parâmetros para obtenção da resolução da medida de consumo. Os dados são derivados do quadro 2 da seção 3.2.1 e das características da arquitetura discutida na seção 3.2.2.

Parâmetro	Valor
Resolução do Tempo de Processamento	1 μs
Tensão de Alimentação das Placas	15 V
Sensor de Corrente de Menor Resistência	270 mV
Amplitude da Faixa Nominal do DAQ	2 V
Resolução Digital do DAQ	12 bits

Fonte: Autoria própria.

Através da amplitude da faixa nominal e da resolução digital do medidor, é possível determinar que a resolução da medida de tensão obtida sobre o sensor de corrente é de 0,49 mV. Considerando a resistência deste sensor, a resolução da medida da corrente consumida é de 1,8 mA. Por fim, multiplicando este valor de corrente pela tensão de alimentação das placas, tem-se uma resolução da medida de potência de 27,1 mW.

Para o cálculo da resolução da medida de energia, é realizado o produto entre as resoluções do tempo de processamento e da potência consumida, resultando em uma resolução de 27,1 nJ.

5.1.2 INCERTEZA DA MEDIDA DE CONSUMO

Assim como a resolução, a incerteza da medida de consumo é derivada das duas medidas básicas realizadas pela plataforma. Porém, não existe uma aplicação padrão, na qual previamente se conhece as suas grandezas (tempo de processamento e consumo energético conhecidos e constantes), que possa ser utilizada para definir estatisticamente a incerteza da

medição. Nestes casos, a incerteza é obtida através de informações técnicas e observações práticas da medição, como, por exemplo, especificações do fabricante dos medidores envolvidos, informações de certificado de calibração ou conhecimento do comportamento e das propriedades das grandezas medidas. As incertezas definida desta forma são denominadas incertezas do tipo B.

A incerteza da medição do tempo de processamento pode ser determinada tendo como base de comparação os tempos obtidos através das estampas do contador de *clock* do processador (TSC), pois esta é a base de tempo mais elementar do sistema. A tabela 2 mostra os tempos de processamento de um conjunto de aplicações de teste com processamento linearmente crescente, medidos través de três métodos: estampa de tempo do *kernel* (utilizada pela plataforma), TSC e comando *time*. As aplicações medidas consistem em um único laço que itera uma variável até atingir o valor indicado pela primeira coluna da tabela.

Tabela 2: Erro da medição do tempo de processamento tendo como base medições realizadas através do TSC.

Aplicação	**Estampa do *Kernel* [t_1] (us)**	**TSC [t_2] (us)**	**Comando *time* (us) (*)**	**Erro% [$(t_1 - t_2)/t_2 * 100$]**
do..while(1E6)	5889	5873	4000	0,27
do..while(1E7)	59267	59109	60000	0,27
do..while(1E8)	589053	587465	588000	0,27
do..while(1E9)	5893057	5877238	5884000	0,27
do..while(1E10)	45357458	45235061	45352000	0,27

Fonte: Autoria própria.

(*) O comando time é incapaz de medir tempos menores que 1ms.

Os resultados mostrados na tabela 2 referem-se às aplicações com tempo de processamento superior a 1 ms, para que assim o erro causado pela resolução da medida (1 μs) não represente mais do que 0,1% (1 μs / 1 ms) da incerteza encontrada. Através das informações da tabela, é possível determinar que o erro do tempo de processamento medido é de +0,27%. Considerando que o erro é constante, independente do tempo medido (erro sistemático), pode ser compensado pelo software de medição da plataforma, não afetando o resultado da medida. Desta forma, com a precisão do padrão utilizado (TSC), não é observada uma incerteza (erro estatístico) no tempo de processamento medido pela plataforma.

O comando *time* é o método de medição do tempo de processamento mais adotado para aplicações em ambiente Linux, motivo pelo qual os seus resultados foram mostrados na tabela 2. Considerando os resultados obtidos, é possível constatar que o método de medição utilizado na plataforma é notavelmente mais exato do que seria utilizando o comando *time*, principalmente para tempos de processamento inferiores a 100 ms.

A incerteza da medição da potência pode ser calculada baseando-se nas características do amostrador utilizado (DAQ). Considerando que a largura do degrau LSB é a diferença entre as medidas resultante de dois valores digitais consecutivos amostrados pelo DAQ, para fins da medição de consumo da plataforma, este valor corresponde à resolução da potência calculada na seção 5.1.1. As informações relevantes para o cálculo da incerteza são mostradas na tabela 3.

Tabela 3: Parâmetros para obtenção da incerteza da medição da potência. Os dados são derivados do quadro 2 da seção 3.2.1 e da resolução da medida da potência calculada na seção 5.1.1.

Parâmetro	Valor
Largura do Degrau LSB (*)	27,1 mW
Erro de *Offset*	-0,5 LSB
Erro de Linearidade Integral	$\pm$2 LSB
Erro de Linearidade Diferencial	$\pm$0,5 LSB
Erro de Ganho	0 %FSR

Fonte: Autoria própria.

(*) A largura do degrau LSB corresponde à resolução da potência medida.

O erro de *offset* de -0,5 LSB, por ser sistemático, é corrigido pelo software de leitura do DAQ, podendo ser descartado do cálculo da incerteza. Considerando que os erros de linearidade integral e diferencial não são correlacionados, o erro total é obtido através da soma dos dois valores, sendo assim a incerteza da medida é de $\pm$2,5 LSB, o que resulta em $\pm$67,8 mW.

Para calcular a incerteza da medida de energia obtida pela plataforma, considera-se a propagação do erro para o produto de duas medidas não correlacionadas (potência e tempo de processamento), podendo ser expressa pela fórmula 10 (REES, 1984).

$$\frac{\delta(x.y)}{x.y} = \sqrt{(\frac{\delta(x)}{x})^2 + (\frac{\delta(y)}{y})^2} \tag{10}$$

em que x e y são duas medidas não correlacionadas, $\delta(x.y)$ é a incerteza resultante do produto destas duas medidas, $\delta(x)$ é a incerteza de x e $\delta(y)$ é a incerteza de y.

Aplicando o valor calculado para a incerteza da medição da potência na fórmula 10 e considerando que erro estatístico da medição do tempo de processamento é desprezível, a incerteza da medição de energia da plataforma desenvolvida pode ser calculado conforme fórmula 11.

$$\begin{gathered} \frac{\delta(P.t_p)}{P.t_p} = \sqrt{(\frac{0{,}0678}{P})^2 + (\frac{0}{t_p})^2} \\ \delta(E) = \pm 0{,}0678.t_p \end{gathered} \tag{11}$$

em que $\delta(E)$ é a incerteza da medição de energia, t_p é o tempo de processamento da aplicação

em análise e P é a potência média durante a execução da aplicação.

5.1.3 AUMENTO DO CONSUMO DEVIDO À INTERFACE PCIE

Para obter o custo energético do suporte à interface PCIe, foram medidos os consumos médios da placa-mãe Atom e da placa da FPGA em dois cenários: com o cabo PCIe desconectado e a FPGA com nenhum hardware programado; e com o cabo PCIe conectado entre as duas placas e a FPGA programada com o hardware desenvolvido para suportar a comunicação. Nessas medições, apenas o sistema operacional está em execução na placa-mãe Atom.

A tabela 4 mostra os resultados obtidos. Sem a PCIe, a placa-mãe Atom consome 14,48 W e a placa da FPGA 2,81 W. Adicionando o suporte à PCIe, o consumo do Atom aumentou 1,14 W, alcançando 15,62 W, e a FPGA aumentou 0,45 W, alcançando 3,26 W. O resultado líquido é que, na plataforma desenvolvida, o suporte à interface PCIe aumentou o consumo médio total em 1,59 W.

Tabela 4: Custo energético do suporte à interface PCIe

Placa	**Consumo s/PCIe** $[P_1]$ **(W)**	**Consumo c/PCIe** $[P_2]$ **(W)**	**Custo Energético da PCIe** $[P_2 - P_1]$ **(W)**
Placa-mãe Atom	14,48	15,62	1,14
Placa com FPGA	2,81	3,26	0,45

Fonte: Autoria própria.

Esse experimento também revelou que o consumo não se altera com o *throughput* da PCIe. Quando programada a FPGA com o projeto de referência para comunicação PCIe fornecido pela Altera (ALTERA CORP., 2014c), foi observado que o consumo se mantém o mesmo quando a PCIe não está em uso ou quando está em operação com 166 MB/s.

5.2 COMPARAÇÃO DA PLATAFORMA DESENVOLVIDA COM O LEAP

A plataforma LEAP, conforme descrito na seção 2.2.3.8, é uma ferramenta de medição de consumo de aplicações em software bem aceita pela comunidade científica. Nesta seção, como maneira de avaliar a qualidade da medida realizada pela plataforma desenvolvida, é feita uma comparação entre as medidas de consumo obtidas pelo método criado e pelo LEAP.

Os dados estatísticos das medições de consumo da aplicação de *benchmark* Linpack são mostrados na tabela 5. O mesmo método de medição, descrito na seção 3.2.7.1, foi

utilizado para a plataforma desenvolvida e para o LEAP. Os consumos medidos não podem ser diretamente comparados, pois se referem a hardwares distintos e a diferentes métodos de instrumentação. Entretanto, os dados estatísticos da tabela permitem comparar a qualidade das medidas.

Tabela 5: Comparação entre a qualidade da medição de consumo obtida pela plataforma desenvolvida e pelo LEAP. Os dados correspondem a 100 repetições da medição da aplicação de *benchmark* Linpack.

Medidas	Plataforma Desenvolvida	LEAP
Consumo Médio (J)	427,44	146,77
Desvio Padrão	0,57 (0,13%)	6,93 (4,72%)
Desvio Absoluto (J)	1,58 (0,37%)	16,81 (11,45%)
Incerteza (Tipo A) (J) (*)	$\pm$1,47 (0,34%)	$\pm$17,88(12,18%)

Fonte: Autoria própria.

(*) Considerando um nível de confiança de 99 %.

O desvio absoluto indica a diferença máxima encontrada entre uma medida e a mediana das 100 medições realizadas. O valor do desvio absoluto obtido para a plataforma desenvolvida é de 1,58 J, ou 0,37 % do consumo médio, enquanto que, para o LEAP, este valor é de 16,81 J, ou 11,45 % do consumo médio.

A incerteza permite estabelecer um intervalo em relação a um valor medido, em que se tem maior probabilidade de encontrar os valores obtidos em repetições da medição da mesma grandeza. Os valores de incerteza apresentados na tabela possuem uma confiança de 99% considerando uma distribuição normal e são mostrados em porcentagem do valor médio, permitindo assim que as incertezas entre as duas plataformas sejam comparadas. A medição realizada pela plataforma desenvolvida possui uma incerteza de 0,34 %, aproximadamente 36 vezes menor que a incerteza do LEAP, que é de 12,18 %.

É importante notar que o DAQ utilizado na plataforma pode ser trocado por outro modelo com melhor desempenho, tendo disponível no mercado amostradores diretamente compatíveis com até 2 Msps e resolução de 16 bits. A utilização de um DAQ com melhores especificações, por reproduzir com maior fidelidade a curva de consumo do sistema, reduziria ainda mais a incerteza da medição. Esta característica não está presente no LEAP, pois parcela considerável da incerteza da medição é devido à incapacidade de separar a aplicação em análise dos demais processos em execução.

Os valores da tabela 5 referentes à plataforma desenvolvida também confirmam a incerteza calculada através da fórmula 11, que, devido ao tempo de processamento de 28,72 s medido para a aplicação, resulta em uma incerteza de $\pm$1,94 J; valor este pouco superior ao

desvio absoluto medido.

5.3 APLICAÇÃO ANALISADA: DETECÇÃO DE INTRUSÃO

Nesta seção, são apresentados os recursos da FPGA utilizados para a implementação do coprocessador em hardware para a classificação através de Árvore de Decisão. Na sequência, são discutidos os tempos dedicados a cada etapa do processamento e o consumo energético da aplicação para detecção de intrusão da rede. O objetivo desta avaliação é demonstrar a utilização da plataforma de medição como ferramenta de análise do perfil de consumo de aplicações híbridas.

5.3.1 RECURSOS DE HARDWARE UTILIZADOS PELO COPROCESSADOR

Os recursos da FPGA utilizados pelo coprocessador desenvolvido para a classificação em hardware da aplicação para detecção de intrusão de rede são mostrados na tabela 6. O circuito usa 6793 elementos lógicos (LEs - *Logical Elements*), ou 4,54% da FPGA Altera Cyclone IV EP4CGX150DF31C7.

Tabela 6: Resultado da síntese do hardware do coprocessador (FPGA Altera Cyclone IV EP4CGX150DF31C7).

Recurso	Quantidade Utilizada	% (FPGA)	% (Circuito)
Elementos Lógicos	**6793**	**4,54%**	**100,0%**
Classificador (Lógica de Controle)	2140	1,43%	31,5%
Classificador (Árvore de Decisão)	283	0,19%	4,2%
PCIe	4370	2,92%	64,3%
Bits Dedicados para Memória	**24432**	**0,37%**	**100,0%**
PCIe	24432	0,37%	100,0%

Fonte: Autoria própria.

A maioria dos recursos lógicos (e todos os bits dedicados para memória) são usados pelo bloco PCIe. A lógica de controle do classificador (atributos e registradores de controle) ocupam 2140 LEs. A lógica da árvore de decisão utiliza 283 LEs, ou apenas 4,2% do circuito do coprocessador. O circuito sintetizado tem uma frequência de operação máxima de 128 MHz. A frequência utilizada na interface da plataforma é de 125 MHz.

5.3.2 TEMPO DE PROCESSAMENTO

A plataforma de medição e o método de medição utilizado possibilitam a medição individual de cada etapa do processamento, permitindo assim encontrar os gargalos do sistema.

A classificação em hardware é conhecidamente da ordem de nanosegundos (FRANCA et al., 2014b), porém o tempo total observado para a classificação através do coprocessador é da ordem de dezenas de microssegundos. Através do perfil do tempo de processamento das etapas da classificação, é possível identificar ineficiências na comunicação PCIe e no recebimento da interrupção pelo software.

A figura 35 mostra o tempo gasto em cada etapa do processamento para a classificação apenas em software e para a versão que utiliza o coprocessador em hardware. Pode-se notar que o tempo total de processamento para 1 milhão de pacotes subiu de 28,23 s para 92,73 s quando utilizada a classificação em hardware.

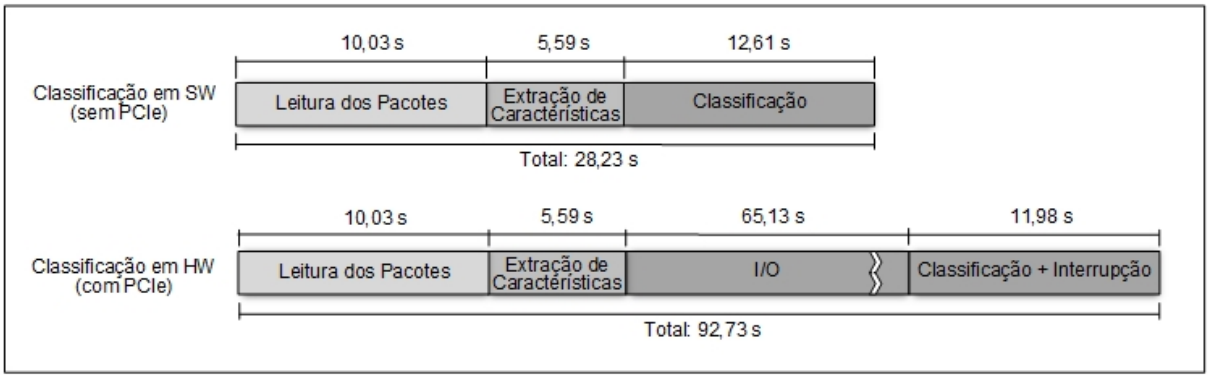


Figura 35: Tempo de processamento para a classificação de 1 milhão de pacotes pela aplicação para detecção de intrusão de rede.

Fonte: Autoria própria.

Através das informações da figura 35, é possível concluir que o tempo dedicado à comunicação PCIe representa aproximadamente 83% do processamento quando utilizado o coprocessador em hardware, sendo o principal responsável pela redução do *throughput* da classificação. Com o levantamento do perfil do processamento proporcionado pela plataforma, é possível determinar que a leitura e escrita dos dados através da PCIe é a parcela da aplicação analisada que representa maior perda de eficiência, sendo que a otimização dessa etapa poderia impactar consideravelmente no tempo total de processamento.

5.3.3 CONSUMO ENERGÉTICO

A tabela 7 mostra os consumos energéticos da aplicação para detecção de intrusão de rede medidos através da plataforma desenvolvida. As medidas de consumo são mostradas no formato “A + F”, em que A é o consumo da placa-mãe Atom e F é o consumo da placa com FPGA. Para a classificação em software, o consumo referente à FPGA é 0, pois apenas a placa-mãe é utilizada.

Tabela 7: Consumo da aplicação para detecção de intrusão de rede. Os valores de consumo são mostrados no formato "A + F", A é o consumo da placa-mãe Atom e F é o consumo da placa com FPGA.

Medida	Classificação em SW	Classificação em HW
Consumo Médio (W)	15,51 + 0	16,77 + 3,26
Tempo de Processamento (1M pc) (s)	28,23	92,73
Energia Consumida (1M pc) (J)	437,94 + 0	1554,84 + 302,59
Energia para Classificação (mJ/pc)	0,43	1,86

Fonte: Autoria própria.

Um resultado contrário às expectativas iniciais foi o consumo energético para a classificação em hardware ser maior do que o consumo para a classificação em software. Como mostrado pelos resultados, isso é causado pelas ineficiências da comunicação PCIe. Devido aos dois fatores descritos anteriormente (o aumento do consumo médio e o aumento do tempo de processamento causado pela PCIe), para a aplicação analisada, o consumo por pacote da classificação é 77 % menor em software do que em hardware.

A análise de consumo apresentada, não visa aferir e comparar a eficiência da implementação do classificador em software e hardware, nem viabilizar a utilização de FPGAs em dispositivos móveis. O objetivo é demonstrar a aplicação da plataforma de medição criada e as possibilidades de análise permitidas através das informações providas pelas medições realizadas.

6 CONSIDERAÇÕES FINAIS

6.1 CONCLUSÕES

Neste trabalho foi desenvolvida uma plataforma de medição de consumo energético para avaliação de aplicações em software e/ou em hardware. A plataforma criada oferece o suporte às aplicações híbridas, em que parte do processamento em software é atribuído a um coprocessador em hardware. Para isso, foi implementada uma interface de comunicação PCIe, permitindo a troca de informação entre o software e o hardware. Esta é a primeira plataforma de medição de consumo energético capaz de comparar implementações de algoritmos equivalentes em software e hardware.

O método de medição proposto pela plataforma permite a medição individual de aplicações ou pequenos trechos de código, separando dos demais processos em execução no sistema operacional. Um método de sincronismo foi criado entre as amostras de consumo e a execução do software da aplicação em análise. Esse sincronismo permite identificar cada vez que o software é alocado no processador pelo escalonador, sendo que, apenas as medições consumo realizadas durante a execução, são contabilizadas para a energia consumida pela aplicação em análise. Esse método de medição também permite a determinação exata do tempo de processamento da aplicação, identificando os tempos de execução ocorridos em cada núcleo do processador.

Os erros da medição realizada pela plataforma foram analisados, determinando que a incerteza da medida do tempo de processamento é desprezível, mesmo quando validada contra a base de tempo mais elementar do processador, o contador de *clock*. A incerteza encontrada para a medida da potência é de $\pm$67,8 mW, resultando em uma incerteza para a medição de energia expressa pela fórmula $\pm 0,0678.t_p$ J, em que t_p é o tempo de processamento da aplicação em análise. Esta incerteza pode ser ainda reduzida utilizando outro modelo de amostrador diretamente compatível com a plataforma e com melhores especificações.

Quando comparadas as características da plataforma desenvolvida com a plataforma LEAP, uma das mais avançadas ferramentas de medição de consumo energético de softwares,

contata-se que o método de medição criado possui as seguintes vantagens: suporte ao processamento híbrido; medição individual da aplicação em análise, retirando da medição o consumo dos demais processos em execução; e a medição independente da frequência do *clock* do processador, permitindo a avaliação de aplicações em execução sob o regime de controle do *clock* realizado pelo sistema operacional. Além disso, a medida realizada pela plataforma criada possui uma incerteza 36 vezes menor do que a medida do LEAP, indicando que este trabalho resultou em uma ferramenta de medição muito mais precisa.

A exatidão da medição, aliada à capacidade de medir individualmente um processo em execução em um sistema multitarefa, permite obter o perfil do consumo de cada etapa do processamento de uma aplicação, fornecendo assim informações exatas e importantes para projetos de baixo consumo. Além disso, auxilia na identificação de gargalos e ineficiências do sistema.

Para ilustrar a importância e utilidade da plataforma de medição desenvolvida, neste trabalho foi apresentada a análise do consumo de uma aplicação para detecção de intrusão de rede por *probing* ataque, com classificação em software e hardware, realizada durante a pesquisa por soluções de segurança de rede energeticamente eficientes fomentada pelo programa *Intel Strategic Research Alliance*.

Para a aplicação analisada, identificou-se que o aumento do consumo médio e o aumento do tempo de processamento causado pela comunicação necessária entre o software e o coprocessador em hardware (PCIe) resultaram em um maior consumo por pacote para a classificação em hardware do quem em software (77% menor).

Por fim, quando comparada com as demais plataformas disponíveis na literatura e no mercado, a plataforma desenvolvida é a única que reúne todos os requisitos necessários para a medição exata do consumo energético de aplicações em software e/ou em hardware, sendo eles: medição individual de uma aplicação em um sistema multitarefa; sincronismo entre a amostra de consumo e a execução da aplicação em análise; medição exata do tempo de processamento; e o suporte para a execução de aplicações híbridas.

6.2 PROPOSTA PARA TRABALHOS FUTUROS

De modo a dividir o consumo total de cada placa, permitindo uma análise por módulo de hardware, sugere-se a inserção de sensores de correntes que possibilitem a medição individual do processador, memória RAM e HD, bem como a medição individual de cada linha de alimentação da FPGA. A fim de aprimorar a exatidão da atribuição do consumo para a

aplicação em análise, podem-se criar novos sinais de sincronismo para considerar, além da execução no processador, os períodos de acesso aos dispositivos de I/O, como o HD ou placa de rede.

Para projetos que exigirem avaliações de sistemas híbridos, sugere-se a otimização da implementação da comunicação PCIe, de modo a aumentar sua eficiência energética e incorporar as recentes tecnologias de *Hard IP* disponibilizadas pela Altera. Também é possível aprimorar o driver PCIe desenvolvido para a placa-mãe Atom, de forma a realizar a programação do hardware da FPGA dinamicamente, permitindo automatizar a medição de um conjunto de hardwares distintos.

Quando finalizado o projeto, no qual este trabalho é utilizado como ferramenta de desenvolvimento e análise, há a possibilidade de disponibilizar o acesso remoto à plataforma de medição, provendo à comunidade científica uma opção precisa de análise para projetos de baixo consumo.

6.3 PUBLICAÇÕES

O desenvolvimento desta dissertação proporcionou, até o momento, a publicação do seguinte artigo:

[1] CEMIN, P. et. al. A Network Intrusion Detection Coprocessor: Software vs. Hardware Power Consumption Analysis. In: 4th Workshop on Circuits and Systems Design (WCAS 2014). [S.l.: s.n.], 2014.

Além disso, os seguintes artigos foram desenvolvidos com o auxílio da plataforma de medição:

[2] FRANCA, A. et al. Moving network protection from software to hardware: an energy efficiency analysis. In: IEEE Computer Society Annual Symposium on VLSI. [S.l.: s.n.], 2014.

[3] FRANCA, A. et al. The Energy Cost of Network Security: A Hardware vs. Software Comparison. In: Accepted to International Symposium on Circuits and Systems (ISCAS 2015). [S.l.: s.n.], 2015.

REFERÊNCIAS

ADVANCED MICRO DEVICES INC. **BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 10h-1Fh Processors**. AMD, 2013. Disponível em: <http://support.amd.com/TechDocs/42300_15h_Mod_10h-1Fh_BKDG.pdf>.

ALTERA CORP. **Using PCI Express on DE4 Boards**. Altera, 2011. Disponível em: <ftp://ftp.altera.com/up/pub/Altera_Material/12.0/Tutorials/Using_PCIe_on_DE4.pdf>.

ALTERA CORP. **Cyclone IV Device Handbook, Volume 1**. Altera, 2014. Disponível em: <http://www.altera.com/literature/hb/cyclone-iv/cyiv-5v1.pdf>.

ALTERA CORP. **IP Compiler for PCI Express - User Guide**. Altera, 2014. Disponível em: <http://www.altera.com/literature/ug/ug_pci_express.pdf>.

ALTERA CORP. **PCI Express High Performance Reference Design**. 2014. Disponível em: <http://www.altera.com/support/refdesigns/ip/interface/ref-pciexpress-hp.html>. Acesso em: 18 out. 2013.

ALTERA CORP. **Quartus II Software: The Proven Productivity Leader**. 2014. Disponível em: <http://www.altera.com/products/software/quartus-ii/about/qts-performance-productivity.html>. Acesso em: 06 jan. 2015.

ASCENT. **DEEP - Decision support for Energy Efficient Processing**. 2013. Disponível em: <http://lasr.cs.ucla.edu/leap/FrontPage>. Acesso em: 08 de fev. de 2015.

BEDARD, D. et al. Powermon: Fine-grained and integrated power monitoring for commodity computer systems. In: **IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the**. [S.l.: s.n.], 2010. p. 479–484.

BOVET, D. P.; CESATI, M. **Understanding the Linux Kernel**. 3. ed. Sebastopol: OŔeilly, 2006.

BRAGA, A. et al. Performance, accuracy, power consumption and resource utilization analysis for hardware / software realized artificial neural networks. In: **Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on**. [S.l.: s.n.], 2010. p. 1629–1636.

BROWNE, S. et al. A portable programming interface for performance evaluation on modern processors. **Int. J. High Perform. Comput. Appl.**, Sage Publications, Inc., Thousand Oaks, CA, USA, v. 14, n. 3, p. 189–204, ago. 2000. ISSN 1094-3420.

BRUGGER, S. T. **Data Mining Methods for Network Intrusion Detection**. [S.l.], 2004.

CEMIN, P. et al. A network intrusion detection coprocessor: Software vs. hardware power consumption analysis. In: **4th Workshop on Circuits and System Design (WCAS 2014)**. [S.l.: s.n.], 2014.

CISCO. **Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013 to 2018**. [S.l.], 2014.

DIGVIJAY, S. et al. **The Atom LEAP Platform For Energy-Efficient Embedded Computing: Architecture, Operation, and System Implementation**. [S.l.], 2010.

FLINN, J.; SATYANARAYANAN, M. Powerscope: a tool for profiling the energy usage of mobile applications. In: **Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on**. [S.l.: s.n.], 1999. p. 2–10.

FRANCA, A. et al. **Method and Energy-Efficient FPGA-Based SOC Implementation for Anomaly Detection System - Third In-depth Report**. [S.l.], 2014.

FRANCA, A. et al. Moving network protection from software to hardware: an energy efficiency analysis. In: **IEEE Computer Society Annual Symposium on VLSI**. [S.l.: s.n.], 2014.

GE, R. et al. Powerpack: Energy profiling and analysis of high-performance systems and applications. **Parallel and Distributed Systems, IEEE Transactions on**, v. 21, n. 5, p. 658–671, May 2010. ISSN 1045-9219.

GOEL, B. et al. Portable, scalable, per-core power estimation for intelligent resource management. In: **Green Computing Conference, 2010 International**. [S.l.: s.n.], 2010. p. 135–146.

GOGOI, P. et al. Packet and flow based network intrusion dataset. In: PARASHAR, M. et al. (Ed.). **Contemporary Computing**. [S.l.]: Springer Berlin Heidelberg, 2012, (Communications in Computer and Information Science, v. 306). p. 322–334. ISBN 978-3-642-32128-3.

HACKENBERG, D. et al. Power measurement techniques on standard compute nodes: A quantitative comparison. In: **Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on**. [S.l.: s.n.], 2013. p. 194–204.

HAWKINS, D. W. **Altera PCIe Analysis**. [S.l.], 2012.

HEINECKE, A. et al. Design and implementation of the linpack benchmark for single and multi-node systems based on intel xeon phi coprocessor. In: **Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on**. [S.l.: s.n.], 2013. p. 126–137. ISSN 1530-2075.

INTEL CORP. **Intel 64 and IA-32 Architectures Software Developerś Manual Volume 3A: System Programming Guide, Part 1**. Intel, 2011. Disponível em: <http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html>.

JACKSON, M.; BUDRUK, R. **PCI Express Technology - Comprehensive Guide to Generations 1.x, 2.x, 3.x**. [S.l.]: MindShare, 2012.

JEVTIC, R.; CARRERAS, C. Power measurement methodology for fpga devices. **Instrumentation and Measurement, IEEE Transactions on**, v. 60, n. 1, p. 237–247, Jan 2011. ISSN 0018-9456.

JONES, M. T. **Inside the Linux 2.6 Completely Fair Scheduler**. [S.l.], 2009.

KESTUR, S.; DAVIS, J.; WILLIAMS, O. Blas comparison on fpga, cpu and gpu. In: **VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on**. [S.l.: s.n.], 2010. p. 288–293.

LAROS, J.; POKORNY, P.; DEBONIS, D. Powerinsight - a commodity power measurement capability. In: **Green Computing Conference (IGCC), 2013 International**. [S.l.: s.n.], 2013. p. 1–6.

MCCANNE, S.; JACOBSON, V. The bsd packet filter: A new architecture for user-level packet capture. In: **Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings**. Berkeley, CA, USA: USENIX Association, 1993. (USENIX'93), p. 2–2.

MEINTANIS, D.; GEORGOPOULOS, K.; PAPAEFSTATHIOU, I. Fpga power consumption measurements and estimations under different implementation parameters. In: **Field-Programmable Technology (FPT), 2011 International Conference on**. [S.l.: s.n.], 2011. p. 1–6.

MEINTANIS, D.; PAPAEFSTATHIOU, I. Power consumption estimations vs measurements for fpga-based security cores. In: **Reconfigurable Computing and FPGAs, 2008. ReConFig '08. International Conference on**. [S.l.: s.n.], 2008. p. 433–437.

MEINTANIS, D.; PAPAEFSTATHIOU, I. On the power consumption of security algorithms employed in wireless networks. In: **Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE**. [S.l.: s.n.], 2009. p. 1–5.

MITCHELL, T. M. **Machine Learning**. New York: McGraw-Hill, 1997. ISBN 0070428077, 9780070428072.

NATIONAL INSTRUMENTS CORP. **User Guide and Specifications for NI USB-6008/6009 Bus-Powered Multifunction DAQ USB Device**. NI, 2012. Disponível em: <http://www.ni.com/pdf/manuals/371303m.pdf>.

NOVELL INC. **openSUSE System Analysis and Tuning Guide**. Novell, 2011. Disponível em: <https://doc.opensuse.org/documentation/html/openSUSE_121/opensuse-tuning/cha.tuning.kprobes.html>.

OLIVEIRA, R. S.; CARISSIMI, A. S.; TOSCANI, S. S. **Sistemas Operacionais**. 3. ed. Porto Alegre: Bookman, 2008.

PEDRONI, V. A. **Eletrônica Digital Moderna e VHDL**. Rio de Janeiro: Elsevier, 2010.

REES, C. Error propagation calculations. **Geochimica et Cosmochimica Acta**, v. 48, n. 11, p. 2309 – 2311, 1984. ISSN 0016-7037.

SCHMADECKE, I.; BLUME, H. Hardware-accelerator design for energy-efficient acoustic feature extraction. In: **Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on**. [S.l.: s.n.], 2013. p. 135–139.

SINGH, D.; KAISER, W. J. **The Atom LEAP Platform For Energy-Efficient Embedded Computing**. [S.l.], 2010.

STATCOUNTER. **Global Stats [Online]**. 2014. Disponível em: <http://gs.statcounter.com>. Acesso em: 15 de jun. de 2014.

TEXAS INSTRUMENTS INC. **ADS7870 12-Bit ADC, MUX, PGA and Internal Reference Data Acquisition System Datasheet**. TI, 2005. Disponível em: <http://www.ti.com/lit/ds/symlink/ads7870.pdf>.

TIAN, X. et al. Exploring the use of hyper-threading technology for multimedia applications with intel openmp compiler. In: **Parallel and Distributed Processing Symposium, 2003. Proceedings. International**. [S.l.: s.n.], 2003. p. 8 pp.–. ISSN 1530-2075.

TSAI, C.-F. et al. Intrusion detection by machine learning: A review. **Expert Systems with Applications**, v. 36, n. 10, p. 11994 – 12000, 2009. ISSN 0957-4174.

TSOI, K. H.; LUK, W. Power profiling and optimization for heterogeneous multi-core systems. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, v. 39, n. 4, p. 8–13, Dec 2011. ISSN 0163-5964.

WEAVER, V. et al. Measuring energy and power with papi. In: **Parallel Processing Workshops (ICPPW), 2012 41st International Conference on**. [S.l.: s.n.], 2012. p. 262–268. ISSN 1530-2016.

WU, S. X.; BANZHAF, W. The use of computational intelligence in intrusion detection systems: A review. **Applied Soft Computing**, v. 10, n. 1, p. 1 – 35, 2010. ISSN 1568-4946.

XIAN, C.; CAI, L.; LU, Y.-H. Power measurement of software programs on computers with multiple i/o components. **Instrumentation and Measurement, IEEE Transactions on**, v. 56, n. 5, p. 2079–2086, Oct 2007. ISSN 0018-9456.

YAN, J. et al. Accurate and low-overhead process-level energy estimation for modern hard disk drives. In: **Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing**. [S.l.: s.n.], 2013. p. 171–178.

ZOU, D.; DOU, Y.; XIA, F. Optimization schemes and performance evaluation of smith–waterman algorithm on cpu, gpu and fpga. **Concurrency and Computation: Practice and Experience**, v. 24, n. 14, p. 1625–1644, 2012. ISSN 1532-0634.

APÊNDICE A – EXEMPLO DE CÓDIGO INSTRUMENTADO PARA MEDIÇÃO

```
#include <power_measurement.h>

int main(int argc, char** argv) {
    unsigned long long i=0;
    set_mypid();                        // Configura TGID.
    start_measurement();                // Inicia medida.
    while(i < 10000000000) {
        i++;
    }
    stop_measurement();                 // Finaliza medida.
    return 0;
}
```