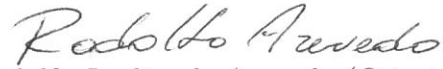
Estimativa de Consumo de Energia em Nível de Instrução para Processadores Modelados em ArchC

Este exemplar corresponde à redação da Dissertação apresentada para a Banca Examinadora antes da defesa da Dissertação.

Campinas, 26 de Outubro de 2007.



Rodolfo Jardim de Azevedo (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Ma, Josué Tzan Hsin
M11e Estimativa de consumo de energia em nível de instrução para processadores modelados em ArchC / Josué Tzan Hsin Ma -- Campinas, [S.P. :s.n.], 2007.

Orientador : Rodolfo Jardim de Azevedo
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Consumo de energia. 2. Arquitetura de computadores. 3. Estimativa de potência. I. Azevedo, Rodolfo Jardim de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Instruction level power consumption estimation for ArchC processors.

Palavras-chave em inglês (Keywords): 1. Energy consumption. 2. Computer architecture. 3. Power estimation.

Área de concentração: Sistemas de computação

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Rodolfo Jardim de Azevedo (IC-UNICAMP)
Prof. Dr. Paulo Cesar Cenducatte (IC-UNICAMP)
Prof. Dr. Manoel Eusébio de Lima (CIN-UFPE)

Data da defesa: 26-10-2007

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 26 de outubro de 2007, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Manoel Eusébio de Lima
CIn. – Universidade Federal do Pernambuco

Prof. Dr. Paulo César Centoducatte
IC – UNICAMP

Prof. Dr. Rodolfo Jardim de Azevedo
IC – UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Estimativa de Consumo de Energia em Nível de Instrução para Processadores Modelados em ArchC

Josué Tzan Hsin Ma[1]

Setembro de 2007

Banca Examinadora:

- Rodolfo Jardim de Azevedo (Orientador)
- Manoel Eusébio de Lima
  CIn - UFPE
- Paulo César Centoducatte
  IC - Unicamp
- Guido Costa Souza de Araújo (Suplente)
  IC- Unicamp

[1]Suporte financeiro de: Bolsa do CNPq (processo 381829/2006-5) 2006 - 2007

Resumo

A constante redução do tamanho e o conseqüente aumento do número de transistores em um mesmo chip faz com que a potência dissipada pelos circuitos digitais aumente exponencialmente. Esse fato, combinado com a crescente demanda por dispositivos portáteis, têm levado à uma crescente preocupação quanto ao consumo de energia. Quanto mais potência é dissipada mais calor é gerado e mais energia é gasta com o seu resfriamento. Como resultado, projetistas estão considerando cada vez mais o impacto de suas decisões nesse quesito. Atualmente, ADLs[1] têm sido utilizadas para projetar novos processadores. Essas linguagens descrevem o comportamento da arquitetura para cada ação ou instrução. ADLs, além de diminuirem o tempo de projeto, são úteis para descobrir problemas arquiteturais em um nível mais elevado. Nesse trabalho, foi desenvolvida uma ferramenta de estimativa de consumo de energia em nível de instrução utilizando-se como base a ADL ArchC e, como estudo de caso, um processador SPARCv8. Como resultado do uso da ferramenta desenvolvida, uma simulação de um programa com estimativa de consumo de energia pode ser realizada 100 vezes mais rápida, na média, em relação ao fluxo tradicional.

[1] ADL: Architecture Description Language.

Abstract

The constant reduction in size and consequential increase in number of transistors inside a chip causes an exponential growth in digital circuit power consumption. Combined with the growing demand for portable electronic devices, this has led to a rising concern about energy consumption. The more power is dissipated, the more heat is generated, and the more energy is spent in the cooling process. As a result, designers have been more and more considering the impact of their decisions on this matter. Currently, ADLs[1] are being used to design new processors. These languages describe the architectural behaviour for each action or instruction. Besides decreasing the time-to-market gap, ADLs are useful in discovering architectural problems at a higher level. This work presents an instruction level power estimation tool that uses ArchC ADL as a base, and a SPARCv8 processor as a case study. By using the developed tool, a simulation of a program with estimated power consumption can be accomplished 100 times faster, in average, than the traditional tools.

[1] ADL: Architecture Description Language

Agradecimentos

Eu gostaria de agradecer a Deus por ter me dado a vida, a força e a sabedoria necessários para concluir mais esse projeto.

Gostaria de agradecer minha familia pelo apoio dado, por ter acreditado em mim e em meu potencial

Gostaria de agradecer ao meu orientador por ter me dado a oportunidade, as idéias e o apoio necessário para concluir.

Gostaria de agradecer aos irmãos da igreja por terem me ajudado em todo o tempo, durante as horas difíceis e alegres.

Gostaria de agradecer às pessoas especiais em minha vida, que a tornam mais legais e alegres, fazendo com que cada momento pareça único e extremamente agradável.

Gostaria de agradecer meus amigos e colegas do LSC e do IC.

Gostaria de agradecer meus amigos da Casa6 e extinta Zebulom.

Gostaria de agradecer também ao suporte financeiro da bolsa CNPq 381829/2006-5.
E aqui fica aquele agradecimento para aquelas pessoas que fizeram algo por mim mas não me lembrei. Afinal, ninguém é perfeito.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A constante redução do tamanho e o conseqüente aumento do número de transistores em um mesmo chip faz com que a potência dissipada pelos circuitos digitais aumente exponencialmente. Esse fato, combinado com a crescente demanda por dispositivos portáteis, têm levado à uma crescente preocupação quanto ao consumo de energia.

Quanto mais potência é dissipada mais calor é gerado e mais energia é gasta com o seu resfriamento. Como resultado, projetistas estão considerando cada vez mais o impacto de suas decisões nesse quesito. É por esse motivo que o consumo de energia é o ponto chave de qualquer projeto de sistemas atualmente e prover ferramentas de fácil uso, rápidas e precisas para essa estimativa é o desafio de pesquisadores e projetistas.

A crescente preocupação com o consumo de energia em sistemas digitais torna necessária a modelagem e estimativa de consumo de energia em todos os seus componentes, principalmente no processador. Nestes, o consumo é conseqüência da seqüência de instruções que são executadas e, também, dos dados manipulados.

Os desenvolvedores devem portanto estar preocupados com o quanto de energia que pode ser economizada enquanto trabalham nas várias fases de implementação do sistema. Para tanto, necessitam de ferramentas que possam medir ou estimar o consumo do sistema durante o percurso do projeto. Quanto mais informações sobre este consumo estiverem à disposição dos projetistas, mais fácil será o desenvolvimento do projeto de um sistema eficiente e econômico.

Entre as várias implicações do aumento do consumo de energia estão:

- **Menor tempo de vida das baterias:** este é um fator de mercado importante, pois pode defnir a escolha do usuário por um produto ou outro, que tenha características semelhantes, mas que forneça uma maior autonomia.
- **Redução do tempo de vida do circuito:** o aumento exponencial na densidade de potência $\frac{W}{cm^2}$ pode estragar o circuito facilmente.
- **Aumento do custo para o resfriamento do chip:** é outro fator importante de mercado que, inevitavelmente, ocasiona o aumento do custo do produto final.

Apesar de existirem inúmeras ferramentas de estimativa de consumo de energia, desempenho e área, são poucas as opções que estão disponíveis para estimá-las em um nível mais alto de abstração. Estimar consumo de energia em *gate-level* de circuitos mais complexos torna-se uma tarefa excessivamente onerosa em questão de tempo. Uma outra dificuldade é que o uso destas ferramentas é feitas nos últimos estágios do ciclo do desenvolvimento do projeto, retarda a detecção de uma escolha de projeto mal-feita.

Por isso, conforme a complexidade do circuito aumenta, é necessário prover ao projetista um meio de subir nos níveis de abstração. Atualmente, linguagens de descrição de arquiteturas (ADLs) têm sido utilizadas para projetar novos processadores. Essas linguagens permitem descrever o comportamento da arquitetura para cada ação ou instrução desses processadores. ADLs, além de diminuirem o tempo de projeto, são úteis para descobrir problemas arquiteturais em um nível mais elevado de abstração.

O objetivo deste projeto é prover uma ferramenta para estimativa de consumo de energia de execução de programas em processadores, a nível de instrução, com suporte ao modelo ora existente no ArchC..

1.1 Contribuições deste Trabalho

Este trabalho está inserido na área de projeto de sistemas digitais, mais especificamente em análise de potência de circuitos digitais em alto nível, e foi totalmente desenvolvido

dentro do LSC[1] [2], do Instituto de Computação da UNICAMP e propõe uma ferramenta de estimativa de consumo de energia em nível de instruções para processadores modelados em ArchC.

As principais contribuições deste trabalho estão resumidas abaixo:

- Análise e combinação de metodologias voltadas para estimativa de consumo de energia anteriormente propostas por vários autores;
- Desenvolvimento de uma ferramenta para estimativa de consumo de energia em nível de instruções para ArchC;
- Esta ferramenta não aumenta o tempo de simulação do ArchC;
- Desenvolvimento de uma ferramenta que oferece um ganho de desempenho em comparação às outras existentes no mercado.

1.2 Organização do Texto

Esta dissertação está organizada da seguinte forma: o capítulo 2 apresenta os conceitos básicos utilizados e analisados neste trabalho além de uma revisão bibliográfica do estado da arte nos modos, ferramentas e níveis de estimativa de consumo de energia.

No capítulo 3 são apresentadas as ferramentas utilizadas neste trabalho além de apresentar a metodologia e o desenvolvimento da ferramenta proposta.

No capítulo 4 são apresentados os resultados, as validações das ferramentas, as tabelas comparativas entre ferramentas convencionais e a ferramenta desenvolvida e a tabela contendo os resultados finais.

O capítulo 5 conclui essa dissertação dando as considerações finais, as contribuições desse trabalho e possíveis trabalhos futuros.

[1] http://www.lsc.ic.unicamp.br
[2] LSC: Laboratório de Sistemas de Computação

Capítulo 2

Conceitos Básicos

Neste capítulo serão mostrados alguns conceitos básicos sobre o consumo de energia em sistemas digitais não sub-micro e sua estimativa. Como este trabalho está focado em um nível mais alto de abstração, não serão discutidos aqui trabalhos ou conceitos relacionados aos níveis mais baixos como *gate-level* e *transistor-level*. Em alto nível estão localizadas as estimativas em nível arquitetural, nível comportamental, nível de instruções e em nível de sistema.

2.1 Consumo de Energia

Primeiramente é necessário descrever como um dispositivo CMOS[1], composto pelo par nMOS-pMOS, consome energia pois, essa é a tecnologia mais utilizada atualmente em projetos de sistemas digitais. A figura 2.1 mostra os principais componentes de um transistor dessa tecnologia.

Os elementos mostrados são o *gate*, a fonte e o dreno; e entre os dois últimos há o canal, que é percorrido por uma corrente quando o transistor está ativo. Ao se aplicar uma certa tensão (dependente da tecnologia) no *gate*, o canal[2] é polarizado, permitindo a passagem da corrente.

O consumo de energia em um dispositivo como este pode ser dividido em duas cate-

[1]CMOS: Complementary Metal Oxide Semiconductor.

[2]Quando se fala em tecnologia de processo, $.18\mu$, por exemplo, está se referindo a largura do canal.

pois todo o modelo é baseado em programas simples, com apenas um tipo de instrução em cada núcleo executado, sendo construído utilizando propriedades simples e visíveis ao nível de instruções e é reutilizável, podendo ser usado como base para testes em outros modelos de processadores.

3.4 ArchC

O algoritmo 3.4.1 mostra um possível modo de adicionar ao ArchC o suporte à estimativa de consumo de energia em nível de instruções. No algoritmo apresentado, *power* é um acumulador utilizado para calcular o consumo de energia, enquanto X é o consumo médio da instrução. Ao final da simulação o consumo médio do processador é dado pela razão de *power* pelo número de instruções executadas, como já demonstrado na seção 2.3.

Algoritmo 3.4.1 Comportamento da função *add* em ArchC com suporte a estimativa de consumo de energia

```
void ac_behavior(add_reg)
{
    writeReg(rd, readReg(rs1) + readReg(rs2));
    update_pc(0,0,0,0,0);
    power+=X;
}
```

Outra possível implementação para o ArchC é descrever em uma função a tecnologia e os valores do consumo de cada instrução como mostrado no algoritmo 3.4.2.

Algoritmo 3.4.2 Alternativa para o comportamento da função *add* em ArchC com suporte a estimativa de consumo de energia

```
void ac_behavior(add_reg)
{
    writeReg(rd, readReg(rs1) + readReg(rs2));
    update_pc(0,0,0,0,0);
    power+=function(X,Y);
}
```

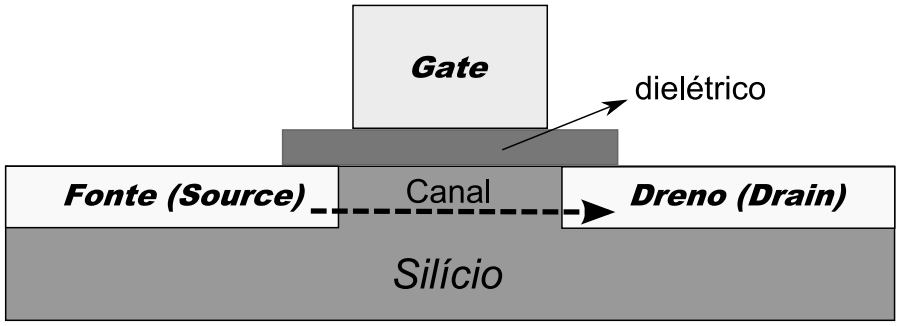


Figura 2.1: Principais componentes de um transistor

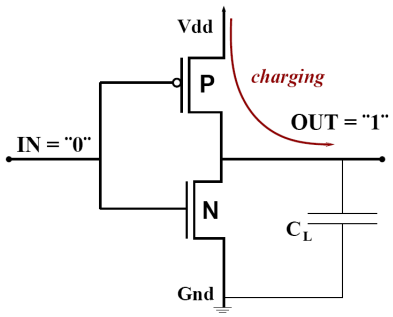
gorias principais: o estático e o dinâmico. O consumo estático é aquele utilizado para manter o dispositivo ligado, sem alteração de seu nível lógico. Esse consumo estático se deve ao fato de haver vazamentos nos transistores, também chamado de *leakage current*. A *leakage current* ocorre da fonte para o dreno, causado pelos reduzidos limiares de voltagem que impedem o transistor de se desligar por completo. A potência dissipada pelos vazamentos também atende pelo nome *leakage power*. Esse tipo de potência começa a ser mais significativa em tecnologias nanométricas.

O consumo dinâmico ocorre durante o chaveamento dos transistores. É aquele utilizado pelo sistema em funcionamento, o consumo útil. Essa é a energia gasta para carregar ou descarregar os capacitores na saída do gate. A potência dissipada pelos chaveamentos domina todo o consumo do sistema (em tecnologias não sub-micros) e é conhecida também pelo nome de *switching power*. Esse processo pode ser visto na figura 2.2. Esta figura mostra a mais simples das portas lógicas CMOS, um inversor. Esta porta lógica é composta por dois transistores: um tipo-P (parte superior da figura 2.2(a)) e um tipo-N (parte inferior da figura 2.2(a)).

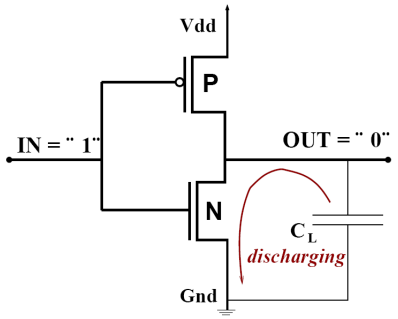
A figura 2.2(a) mostra como o capacitor na saída do gate é carregado. Supondo que a entrada IN deste gate esteja inicialmente com o valor lógico 0, o transistor inferior, do tipo N, está desligado e, conseqüentemente, o superior está ligado. Quando o superior está ligado, está conduzindo e dessa maneira, C_L em OUT é carregado.

Agora, supondo que IN faça a transição $0 \rightarrow 1$. Isso causa o contrário, fazendo o

transistor superior parar de conduzir e o inferior começar a conduzir, isso é mostrado na figura 2.2(b). Com isso o capacitor na saída do gate é descarregado de OUT para Gnd.



(a) IN em 0 carrega C_L



(b) IN em 1 descarrega C_L

Figura 2.2: Carga e descarga dos capacitores

Uma boa aproximação para o cálculo do consumo dinâmico é a fórmula 2.1, onde C_L é a capacitância de carga do *gate*, V_{dd}^2 é o quadrado da tensão de alimentação, f é a freqüência de funcionamento do circuito e T_{Rate} é o *toggle rate*[3].

$$P_{din} = C_L V_{dd}^2 f T_{Rate} \tag{2.1}$$

Há ainda um outro tipo de consumo, denominado de potência de curto-circuito ou *short-circuit power*. Esse consumo ocorre quando os transistores nMOS e pMOS estão conduzindo em tecnologia CMOS de $0 \rightarrow 1$ (ou $1 \rightarrow 0$). Pelo fato de o chaveamento não ocorrer instantaneamente na vida real, por um breve período de tempo há uma ligação

[3] *Toggle Rate*: Taxa de transições lógicas por unidade de tempo.

entre a tensão de alimentação (Vcc) e o terra (Gnd). Para circuitos que tenham tempos baixos de transição, a contribuição de short-circuit power pode ser baixa. No entanto, para circuitos com tempos altos de transição a contribuição pode ser de até 30% de todo o consumo.

O quadro do consumo de energia para CMOS é este:

- **Switching Power:** $70 \approx 90$ %
- **Leakage Power:** $5 \approx 25$ %
- **Short-Circuit Power:** 5 %

2.2 Principais Métodos de Estimativa de Consumo de Energia em Alto Nível

Para estimar o consumo de energia de um processador existem várias técnicas nos mais variados níveis de abstração. As técnicas apresentadas a seguir fazem parte do conjunto de alto nível de abstração. É sabido que a precisão é menor em alto nível quando comparadas às técnicas de baixo nível. Isso porque processam menos informação mas, por essa mesma causa, são mais rápidas ao gerar os resultados. Quando utilizam esse nível os projetistas não estão preocupados apenas com a precisão, o valor absoluto do consumo de energia do dispositivo, mas com o valor relativo do mesmo, facilitando a escolha entre as alternativas de implementação existentes em um nível mais alto e economizando tempo de projeto. Procura-se, então, uma relação custo-benefício, na qual a precisão é sacrificada pela eficiência, possibilitando-se fazer ajustes ainda nos estágios iniciais do projeto.

2.2.1 Nível Arquitetural

No nível arquitetural, as primitivas são os blocos funcionais como os somadores, os multiplexadores, controladores, entre outros. As dificuldades de estimativa neste nível de abstração se devem ao fato de os níveis mais baixos do sistema não estarem ainda especi-

ficados no início do projeto. Este nível é dividido em dois principais métodos: o analítico e o empírico.

Métodos Analíticos

O método analítico tem como característica a tentativa de fazer a relação do consumo de energia com informações fundamentais, como a capacitância física e a atividade do circuito. Esse método é ainda subdividido em dois outros modelos: o baseado em complexidade e o baseado em atividade.

O modelo baseado em complexidade sustenta-se no fato de que a complexidade de um circuito pode ser grosseiramente descrita em termos de circuitos mais simples. Por exemplo, um multiplicador 16X16 pode ser descrito utilizando-se uma combinação de NANDs de 2 portas, estima-se então o consumo do multiplicador com base no consumo dessa porta lógica. Um problema desta abordagem é que, ao se reduzir a complexidade de um circuito a um conjunto de combinações de uma porta lógica, apenas ela é utilizada como referência para o consumo de todas as partes do circuito, não se levando em conta alguns ítens importantes como estilos de circuito, estratégias e técnicas de *layout*, entre outros. Uma ferramenta que utiliza essa técnica é o CES (Chip Estimation System) [19], que se utiliza da expressão 2.2 para estimar o consumo:

$$P = \sum_{i \in fns} GE_i \left(E_{typ} + C_L^i V_{dd}^2 \right) f A_{int}^i \tag{2.2}$$

onde GE_i é o número de gates equivalentes de cada bloco i, E_{typ} é o consumo médio pelo gate equivalente i (quando ativo), C_L^i é sua capacitância média, V_{dd}^2 é o quadrado da tensão de alimentação, f é a freqüência e A_{int}^i é a porcentagem de *gates* que chaveiam a cada ciclo de *clock* para o bloco i. Essa aproximação é particularmente imprecisa para circuitos especializados como memórias.

Liu e Svensson [16] melhoraram a precisão do trabalho [19] aplicando uma estimativa customizada para alguns tipos de circuitos (memórias, interconexões, etc...). Por exemplo, para calcular a estimativa de consumo de uma memória é utilizada a fórmula 2.3.

$$P_{mem} = \frac{2_k}{2} \left(c_{int} l_{col} + 2^{n-k} C_{tr} \right) V_{dd} V_{swing} f \tag{2.3}$$

onde 2_k é o número de células na linha, c_{int} é a capacitância por comprimento, l_{col} é o comprimento, 2^{n-k} é o número de células na coluna, C_{tr} é a capacitância mínima, V_{dd} é a tensão de alimentação, V_{swing} é a tensão de chaveamento e f é a freqüência do dispositivo.

Já o modelo baseado em atividade utiliza a atividade do circuito, sua entropia. Esse modelo resolve alguns dos problemas citados acima e consiste em simular o circuito para as entropias de entrada e saída dos blocos funcionais e assim calcular o consumo médio do circuito. Najm [20] chegou à seguinte conclusão:

$$P\ \alpha\ Capacitancia\ X\ Atividade\ \alpha\ Area\ X\ Entropia \tag{2.4}$$

o consumo de energia é proporcional ao produto da capacitância com a atividade e também proporcional ao produto da área com a entropia do sistema. Sua metodologia consistiu em executar uma simulação RTL[4] do circuito de forma a medir as entropias de entrada e saída dos blocos funcionais e utilizando esses dados para estimar a área e a entropia total e calcular o consumo de energia.

Essa abordagem também tem seus problemas, pois considera a capacitância uniformemente distribuída por toda a área do circuito, o que não é verdade; e outras informações como a temporização não são levadas em conta para o cálculo da entropia.

Métodos Empíricos

No método empírico, ao invés de se tentar relacionar o consumo dos componentes com os parâmetros fundamentais, o consumo é medido por macro-modelos. Macro-modelos são modelos de consumos de implementações de blocos funcionais. A maioria das abordagens utilizadas pelas ferramentas atualmente se utiliza de métodos empíricos e esse método pode ser dividido em três categorias: os de atividade fixa, os que levam em conta dados estatísticos das atividades dos sinais e os baseados nas transições de entrada.

A primeira categoria leva em consideração uma atividade fixa dos sinais, o que não ocorre na realidade, podendo ocasionar erros consideráveis na estimativa. O cálculo é feito baseado na equação 2.5.

$$P = \sum_{i \in blocos} k_i G_i f_i \tag{2.5}$$

[4]RTL: Register Transfer Level.

onde cada bloco funcional i é caracterizado por uma constante k PFA[5], uma medida G de complexidade do hardware e uma freqüência de ativação f. Por exemplo, um multiplicador tem uma complexidade relacionada ao quadrado do tamanho da palavra de entrada, então $G_{mult} = N^2$. A freqüência de ativação é simplesmente a freqüência com a qual as multiplicações são feitas no algoritmo, f_{mult}. E, finalmente, a constante PFA k_{mult} é extraída empiricamente de projetos anteriores (extraídos de artigos ou dados próprios). No caso analisado por [23] o multiplicador consome cerca de 15fW/bit^2-Hz para uma tecnologia de 1.2μm com uma tensão de 5V. Então a estimativa de consumo de energia do multiplicador é dada pela equação 2.6.

$$P_{mult} = k_{mult} N^2 f_{mult} \tag{2.6}$$

A segunda categoria, baseada nas atividades dos sinais, já é uma melhora do modelo anterior. Aqui as atividades dos sinais são monitoradas durante a simulação, garantindo uma maior precisão. A maioria das ferramentas atuais baseia-se nesse tipo de estimativa, onde assume-se que o consumo de energia (real) de cada componente já tenha sido empiricamente estimado por simulações anteriores. Uma ferramenta comercial chamada WattWatcher/Architect [3] baseia-se nesse tipo de estimativa e ainda consegue propagar probabilisticamente os sinais para níveis inferiores, sendo capaz de analisar uma arquitetura de 120.000 *gates* em apenas 34 minutos.

Uma última categoria não se baseia nas estatísticas de entrada, mas nas transições das entradas [18]. Essa estimativa supõe que exista um modelo de energia para cada unidade funcional, uma tabela contendo a potência consumida para cada transição na entrada. Detectando-se transições na entrada muito parecidas e padrões de energia, consegue-se diminuir o tamanho da tabela através de *clustering*. Um simulador de energia totalmente baseado nessa técnica é o SimplePower [32].

Os métodos empíricos são mais precisos em relação aos métodos analíticos por terem relação mais forte com o hardware. Uma das desvantagens destes métodos é o tempo gasto pelos projetistas para caracterizar um modelo, embora seja feito apenas uma vez na maioria dos casos.

[5]PFA: Power Factor Approximation.

2.2.2 Nível Comportamental

Quanto mais se eleva o nível, mais trabalhosa a estimativa de consumo de energia se torna pois menos informação sobre o sistema se sabe [12]. A estimativa no nível comportamental é baseada na freqüência de acesso dos recursos. Quanto mais um recurso é ativado, mais energia aquele recurso irá consumir. As técnicas comportamentais podem ser divididas em duas principais linhas: a predição estática e a predição dinâmica. A principal diferença entre as duas é que, enquanto a estática estima a freqüência de acesso aos diferentes recursos do hardware, a predição dinâmica retira esses dados de uma simulação, com a entrada fornecida pelo usuário.

A principal tarefa da predição estática é estimar a freqüência de acessos aos recursos de hardware, analisando a descrição comportamental da função sendo implementada. Uma ferramenta que utiliza esse tipo de abordagem é a HYPER-LP [2, 17]. A expressão 2.7 demonstra como é estimado o consumo de energia nessa ferramenta, onde f_r é a freqüência de acesso ao recurso r, C_{r0} é a capacitância do recurso quando ativado e V_{dd} é a tensão de entrada.

$$P = \sum_{r \in recursos} f_r C_r V_{dd}^2 \tag{2.7}$$

A predição dinâmica retira os dados de uma simulação do dispositivo, com as entradas dadas pelo usuário. Após a obtenção dessas informações, os dados são ligados a um modelo análogo ao de predição estática. Por necessitar obter os dados de uma simulação, a predição dinâmica é um pouco mais lenta e trabalhosa do que a estática, porém mais precisa.

2.2.3 Nível de Instruções

No nível de instruções associa-se o consumo de energia à execução de cada instrução individualmente. O consumo total é calculado, basicamente, a partir da média do consumo da repetição de cada uma das instruções e da energia dissipada entre cada par de instruções, devido a chaveamentos de circuitos. Existe uma dificuldade em obter-se esses dados, já que é necessária uma análise de toda a arquitetura do conjunto de instruções e uma análise de cada possível par, ou conjunto, de instruções, podendo tornar-se um mecanismo

inviável para processadores com muitas instruções diferentes. A seção 2.3 demonstra com mais detalhes as ferramentas e métodos utilizados por esse nível de abstração.

2.2.4 Nível de Sistema

No nível de sistema, o objetivo é fornecer uma ferramenta que analise o sistema. Essa ferramenta é bastante útil para identificar afunilamentos do sistema antes de perder tempo tentando otimizar, equivocadamente, outras partes. Esse é o mais alto nível entre os níveis de abstração mostrados aqui e são direcionadas para estimar o consumo de SoCs[6].

Na realidade nenhuma das estratégias de alto nível utilizadas para estimar o consumo de energia de um dispositivo possui um alto nível de precisão absoluta, mas consegue captar, generalizar e estimar o consumo com um erro aceitável, que é o verdadeiro objetivo das estimativas de alto nível.

2.3 Estimativa de Energia em Nível de Instrução

Os microprocessadores atuais são sistemas extremamente complexos constituídos por vários blocos funcionais internos. No entanto, essa complexidade está escondida por detrás de uma abstração muito mais simples, o conjunto de instruções do processador. Tendo essa idéia como base, é bastante lógico considerar o consumo de energia destas instruções, já que cada instrução envolve processos e atividades específicas por entre os vários blocos internos do processador.

O problema de estimativa de consumo de energia em nível de instrução é um problema relativamente recente [27]. Desde a sua proposta até os dias atuais, o problema tem sido abordado de muitas maneiras e propósitos diferentes.

A maneira mais utilizada é a execução de cada uma das instruções repetidamente, os chamados *stimuli*, do qual se obtém o consumo médio de cada instrução. Durante esse processo a corrente elétrica média é medida e guardada em uma tabela de custos base e pode ser estimada também por classes de instruções.

[6]SoC: System-on-Chip.

Tiwari et al. [27, 28] explorou a estimativa de consumo de energia em nível de instruções utilizando modelos simples como a média de energia consumida por instrução, derivada de medidas experimentais. Em seus trabalhos também foi considerada a energia consumida entre as instruções, o consumo intra-instruções. Eles também propuseram um método simples e prático para medir a corrente. Ao testar um programa contendo um conjunto repetido de instruções em um processador cuja alimentação foi isolada do resto do sistema, a forma da corrente é captada e, sendo periódica, sua média pode ser facilmente calculada. Essa técnica muito simples foi utilizada para obter dados sobre o consumo de um processador 486DX2. Variações dessa técnica foram utilizadas para medir o consumo de energia de outros processadores como o ARM7TDMI e o Motorola DSP 56100 em outros projetos.

Russel [26] considerou uma média de consumo de energia igual para todas as instruções do ISA baseado na observação de que para certas classes de processadores a diferença de consumo de energia entre as instruções era mínima.

Em [11] os autores propuseram estimar o consumo de energia considerando apenas as instruções que são executadas após um NOP.

O custo base de cada instrução, no entanto, pode ser camuflado por vários fatores da arquitetura do processador. Cada um deles gera um valor que difere do custo base, podendo ser o efeito de um *cache-miss* ou um efeito de um *stall* de sistema ou interrupção, entre outros. Além desses são discutidos em [21] o efeito causado por diferentes registradores fontes e destinos, o efeito dos valores de operandos, o efeito das instruções condicionais e o efeito dos diferentes tipos de endereçamentos de uma mesma instrução.

A maior dificuldade para ser implementada em cada tipo de processadores é o tamanho do conjunto de instruções de cada um deles. Isto decorre do fato de estar em um nível mais alto de abstração, pois normalmente um processador comum possui centenas de instruções. No nível mais baixo, seria possível isolar cada um dos blocos funcionais utilizados pelo núcleo do processador, medir o consumo em cada um deles e saber quais deles são utilizados por cada instrução.

Em suma, uma vez construída a tabela de custos base, o cálculo da estimativa de

consumo de energia em nível de instrução pode ser simplificado e dado pela fórmula 2.8

$$P = \frac{\sum (n_i.X_i)}{N} \tag{2.8}$$

onde n_i representa o número de vezes que a instrução foi executada, X_i o consumo médio (custo-base) dessa instrução e N o número total de instruções executadas.

Em [13] é descrita uma série de propriedades necessárias para que o modelo de consumo de energia possa ser utilizado como base para otimização em código de programa. São elas:

- **Precisão (Accuracy):** O modelo deve ser capaz de estimar o consumo de energia precisamente;
- **Simplista (Simplicity):** O modelo deve ser construído utilizando propriedades simples, visíveis ao nível de instruções;
- **Responsável (Accountability):** O modelo deve ser capaz de identificar a significância de cada fator que afeta o consumo de energia;
- **Reutilizável (Retargetability):** O modelo deve ser generalista o suficiente para poder ser utilizado em vários processadores.

Além destas características, é necessário observar certos fatores para minimizar o consumo de energia. Esses fatores podem ser garantidos utilizando um compilador especializado ou otimizado para esse fim. São eles:

- **Instruções utilizadas:** Durante a fase de seleção de código, o modelo precisa escolher seqüências de instruções de uma série de alternativas. Além disso, para o modelo ser altamente preciso, é necessário que todas as instruções sejam incluídas nos testes.
- **Fluxo das instruções:** Quando duas instruções subseqüentes requerem o uso de unidades funcionais diferentes, estas são ativadas ou desativadas de acordo com a necessidade. A energia consumida por essas mudanças de estado podem ser minimizadas otimizando-se o agendamento, ou seqüência das instruções de maneira a

utilizar uma mesma unidade funcional pelo período mais longo possível. Esse tipo de melhoramento precisa levar em conta as dependências dessas instruções.

- **Hierarquia de memória:** Em sistemas com duas ou mais memórias, o compilador precisa decidir a alocação dos objetos na memória. Essas diferenças devem ser consideradas quando o consumo de energia for medido.

- **Parâmetros do modelo:** Para que os requerimentos acima sejam alcançados, não é possível apenas utilizar dados de planilhas. É necessário que se façam medições. Estas medições devem ser simples o suficiente para serem feitas sem que conhecimentos da estrutura interna sejam requeridos.

2.4 Ferramentas

Várias ferramentas têm sido desenvolvidas para estimar o consumo de energia em processadores e sistemas embarcados [5, 7, 10]. Estas têm o objetivo de tornar a medição uma tarefa mais simples, e várias destas ferramentas envolvem otimizações em códigos descritos para certas arquiteturas específicas.

2.4.1 encc

O encc [7] é um programa desenvolvido pelo Grupo de Sistemas Embarcados do Departamento de Ciência da Computação da Universidade de Dortmund, na Alemanha e contém, entre outros, um *backend* para o processador Leon e um arquivo de configuração com informações de consumo de energia das instruções do processador ARM7TDI.

O encc foi também desenvolvido para gerar código para o processador Leon (Seção 3.1). Utilizando este modelo de processador, é possível sintetizá-lo e determinar suas atividades através de simulações. Essas atividades são a medida para a energia dissipada pelo processador.

A figura 2.3 mostra o método de uso da ferramenta e como sua base de dados foi calculada.

[7] Energy Aware C Compiler

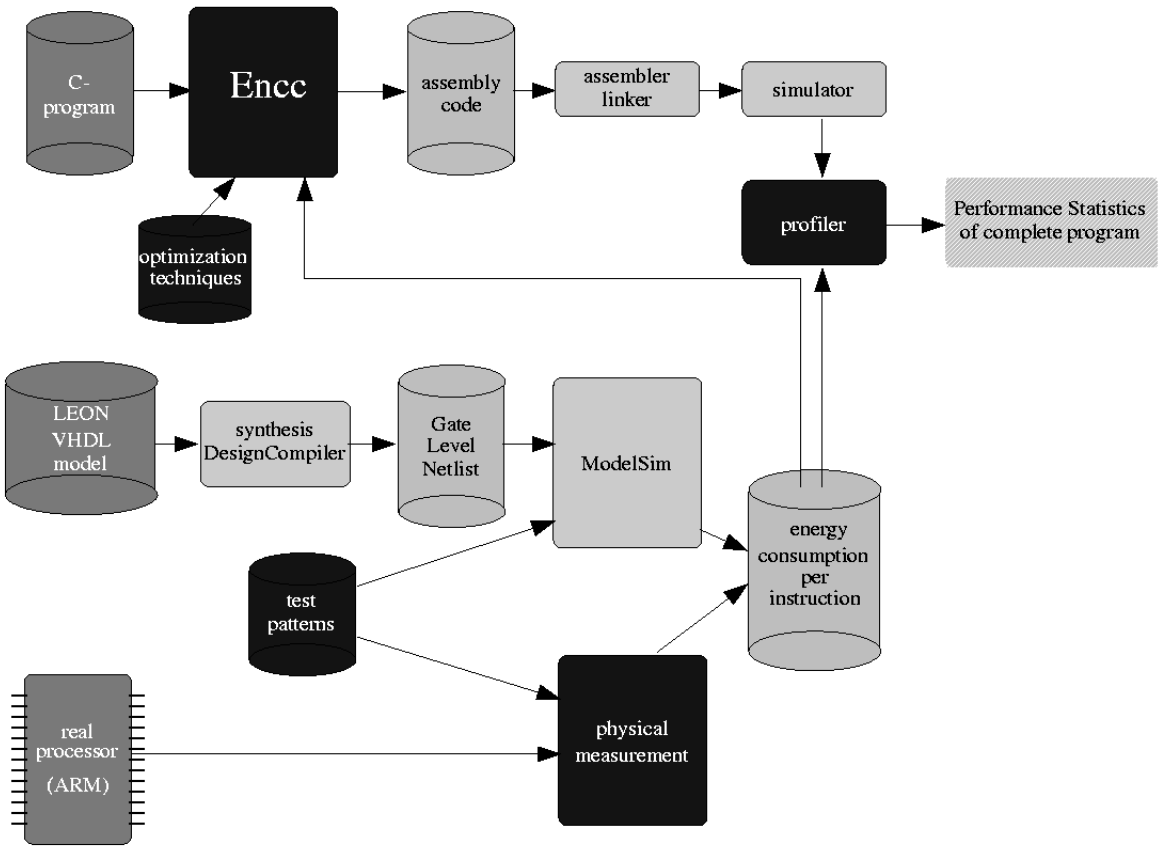


Figura 2.3: encc.

2.4.2 Orinoco

Orinoco [22] é uma ferramenta de otimização de código para consumo de energia. É voltado para uso de códigos em SoCs que necessitam de economia de energia. É uma das ferramentas que analisa o nível eletrônico de sistema[8], nível no qual pode ser alcançada a maior redução de consumo de energia de um sistema.

O Orinoco suporta processos descritos tanto em ANSI C quanto em SystemC, e sua estimativa é utilizada para otimizar algoritmos, caminhos de dados, recursos utilizados, interconexões, acessos a memória, entre outros, podendo gerar uma economia de energia de até 75% em relação ao projeto original. Um exemplo de nível de uso do Orinoco pode ser visto na figura 2.4.

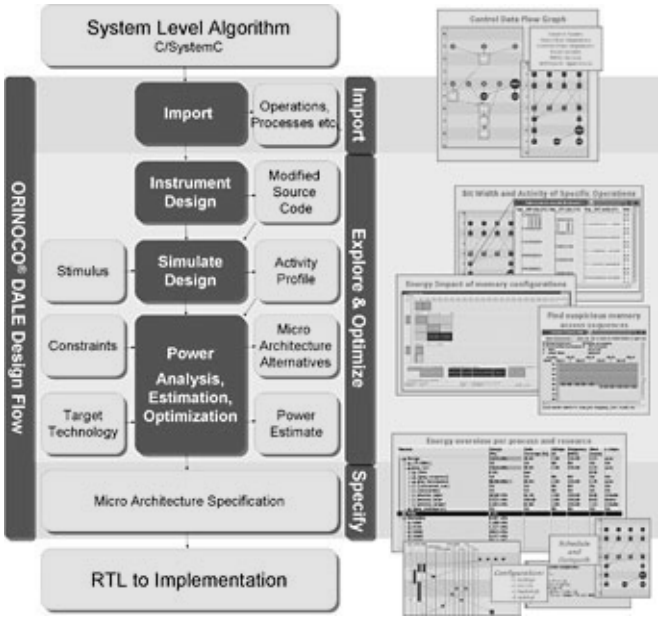


Figura 2.4: Fluxo de uso do Orinoco

2.4.3 PowerScope

PowerScope [24] é uma ferramenta (API) que descreve a energia utilizada por uma aplicação. Ele mapeia o consumo de energia na estrutura do programa do mesmo modo

[8]ESL: Eletrnonic System Level

que o processador mapeia como o programa irá ser executado. O PowerScope utiliza também um programa que analisa os dados mapeados e gera uma descrição de uso por processos e procedimentos.

Em um de seus experimentos o PowerScope ajudou a reduzir em 46% a energia utilizada por uma aplicação de visualização de vídeo.

2.4.4 PowerPlay

A ferramenta PowerPlay, da Altera, foi desenvolvida para análise em nível de sistema de suas FPGAs[9]. A abordagem utilizada é chamada de *spreadsheet-like*, composta por um conjunto de fórmulas simplificadas de potência. Essas fórmulas são usadas com o intuito de se obter o consumo médio de energia através de parâmetros estimados fornecidos (exemplo: tamanho e tipo de blocos lógicos, freqüência, atividade de chaveamento, etc...). A figura 2.5 mostra um resumo do fluxo de uso da ferramenta.

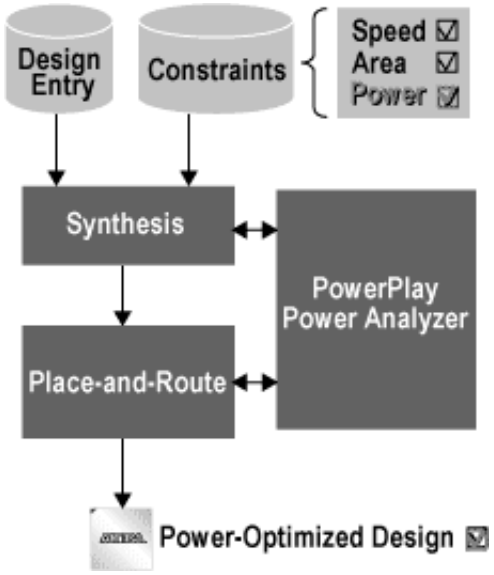


Figura 2.5: Fluxo de uso do PowerPlay.

[9]FPGA: Field-Programmable Gate Array.

2.4.5 XPower

A ferramenta XPower, da Xilinx, foi desenvolvida para análise em nível arquitetural e é utilizada pela Xilinx para estimar o consumo de energia em suas FPGAs. O cálculo da estimativa se baseia na observação de que o consumo dinâmico em circuitos CMOS se dá principalmente (mas não apenas) pela atividade de chaveamento do mesmo. A ferramenta tem um modelo de capacitância para os vários elementos da FPGA (exemplo: LUTs, segmento de roteamento, RAM, etc...) e, juntamente com os dados fornecidos pelo usuário (taxa de atividade, freqüência, etc...) e com os dados da tecnologia utilizada, obtém o consumo de energia. O fluxo de uso da ferramenta pode ser visto na figura 3.3.

2.4.6 PowerCompiler

PowerCompiler é uma ferramenta da Synopsys que serve para estimar e otimizar o consumo de energia nas FPGAs e ASICs[10], bastando que se disponibilizem as bibliotecas contendo as informações da tecnologia, e suporta diferentes níveis de abstração como RTL e *gate-level*. Uma das otimizações aplicadas por esta ferramenta é o chamado *clock-gating*, que consiste basicamente em inibir o clock nas regiões do circuito onde seu uso é desnecessário. Seu fluxo de uso pode ser visto na figura 3.4.

No PowerCompiler a estimativa é modelada por três componentes: o consumo estático, a potência de chaveamento (obtida através das simulações) e a potência estática (dada pelas bibliotecas da tecnologia), como descrito na seção 3.2.2.

2.4.7 Atividade de chaveamento

A maioria das ferramentas de estimativa de consumo de energia utilizam arquivos que contém atividades de chaveamento como entrada. Esses arquivos descrevem quando e quantas vezes os sinais mudam de fase. Com base nessas informações é possível aumentar a precisão da estimativa do consumo de energia.

As atividades de chaveamento podem ser adquiridas de formas diferentes. As mais comuns são:

[10] ASIC: Application-Specific Integrated Circuit.

- **Simulação feita pelo usuário**: Nessa forma, ao realizar a simulação, a atividade de chaveamento é anotada e guardada em um arquivo. Os arquivos mais comuns são VCD [11], ASF [12] e SAIF [13]. Os arquivos VCDs são utilizados pela maioria das ferramentas, os ASFs são utilizados pelas ferramentas da Altera e os SAIFs são um padrão da Synopsys.

- **Entrada manual**: Nessa forma o usuário configura a taxa manualmente pela ferramenta.

Caso não haja nenhuma entrada exterior, a ferramenta passa a utilizar o seu padrão global. Nessa forma a ferramenta assume uma taxa padrão que normalmente é 50% para cada sinal.

A tabela 2.1 mostra uma visão mais geral das ferramentas apresentadas e seus respectivos níveis de estimativa de consumo de energia.

	Nível de Abstração			
	Arquitetural	Comportamental	Instrução	Sistema
encc				•
Orinoco	N/A			
PowerScope		•		•
Xpower	•			
PowerPlay	•			•
PowerCompiler	•			

Tabela 2.1: Ferramentas de estimativa de consumo de energia e seus níveis de abstração.

2.5 ArchC

ArchC [25] é uma ADL, linguagem de descrição de arquiteturas, desenvolvida no Laboratório de Sistemas de Computação do Instituto da Computação da Universidade Estadual de Campinas.

[11] VCD: Value Change Dump
[12] ASF: Signal Activity File
[13] SAIF: Switching Activity Interchange Format

ArchC é baseado em uma linguagem de descrição de sistemas, SystemC. Com esta ferramenta é possível aos projetistas avaliarem rapidamente novas idéias na área de arquitetura de computadores como: projeto de processadores, hierarquia de memória e outros aspectos. Dentre os aspectos que podem ser descritos em ArchC está o comportamento de um conjunto de instruções.

O algoritmo 2.5.1 mostra um exemplo da descrição comportamental de uma instrução modelada em ArchC. Para o processador SPARCv8 a instrução *add* escreve no registrador *rd* a soma dos registradores *rs1* e *rs2* e atualiza o pc[14].

Algoritmo 2.5.1 Comportamento da função *add* em ArchC

```
void ac_behavior(add_reg)
{
    writeReg(rd, readReg(rs1) + readReg(rs2));
    update_pc(0,0,0,0,0);
}
```

A escolha de uso da ArchC neste projeto se deve a vários fatores como, por exemplo, o fato de ela ser desenvolvida nessa instituição e estar disponível para implementações de novos recursos, o que a torna diferente de seus atuais concorrentes e, como trabalha em um nível mais alto que simuladores RTL ou gate-level, as simulações em ArchC também são mais rápidas, economizando horas de trabalho.

Neste capítulo foram apresentados os conceitos básicos de consumo de energia, foram abordados vários métodos de como estimar esse consumo passando pelos níveis arquitetural, comportamental, de instruções e de sistema, juntamente com seus métodos analíticos, empíricos, estáticos ou dinâmicos. Também foram apresentadas algumas das ferramentas existentes no mercado e o ArchC, que será utilizado neste trabalho. O próximo capítulo apresentará a parte prática do trabalho, como ferramentas e processadores utilizados, ferramentas desenvolvidas, entre outros.

[14]PC: Program Counter

Capítulo 3

Integração de Estimativa de Consumo de Energia ao ArchC

Neste capítulo são apresentados os aspectos práticos desse trabalho como o processador e as ferramentas utilizadas, os meio de captação de dados, a metodologia utilizada e a ferramenta desenvolvida. Por causa da dificuldade de encontrar processadores em VHDL[1] ou outra linguagem sintetizável para que a atividade de chaveamento possa ser anotada para calcular seu consumo de energia e possuir um modelo desse processador descrito em ArchC funcional, foi utilizado apenas um processador, o SPARCv8, com duas implementações diferentes em VHDL.

3.1 Processador

Para esse projeto foram utilizados duas implementações de um processador SPARCv8[2] [29]. O SPARC é um processador RISC[3] de 32 bits e em sua versão mínima possui 72 instruções implementadas.

Seus principais tipos de dados são inteiro de 32 bits e ponto-flutuante (padrão IEEE 754) de 32, 64 ou 128 bits. A parte lógica do processador possui uma unidade operações

[1]VHDL: VHSIC(Very High Speed Integrated Circuit) Hardware Description Language.
[2]SPARC: Scalable Processor ARChitecture Version 8.
[3]RISC: Reduced Instruction Set Computer.

de inteiros (IU), uma unidade de ponto flutuante, um co-processador opcional; cada um com seus próprios registradores de uso específico. Todos os registradores, salvo algumas exceções, são de 32 bits, e a IU[4] controla a maioria das operações do processador, contém os registradores de propósito geral e de controle, além de executar as instruções aritméticas e calcular os endereços de acesso à memória dos LOADs e STOREs. A IU também mantém os PCs[5] e controla o fluxo de execução da unidade de ponto flutuante e do co-processador. Sendo assim, a IU pode ser considerado o núcleo principal do processador.

Esse processador foi escolhido por já haver um modelo funcional em ArchC e possuir implementações em VHDL como o Leon2 e Leon3 sob licença LGPL.

Existe ainda um compilador específico para as arquiteturas Leon2 e Leon3 disponibilizado por Gaisler [8]. Esse suporte torna a escolha desses processadores bastante vantajosa pois não é necessário gastar tempo para modificar um compilador com as especificações dos processadores de teste. Ambos utilizam uma versão alterada do sparc-elf.

3.1.1 LEON2

O Leon2 [14] é um processador totalmente descrito em VHDL sintetizável. Suas características principais são:

- ISA do SPARCv8;
- Pipeline de 5 estágios;
- Unidades de multiplicação, divisão e MAC em hardware;
- Caches de dados e instrução separadas;
- Caches configuráveis: 1-4 conjuntos de 1-64 kbytes/cada, com algoritmo de substituição aleatório, LRR[6] ou LRU[7];
- Periféricos como UARTs, timers, controladores de interrupção e portas de E/S de 16 bits “on-chip”;

[4]IU: Integer Unit
[5]PC: Program Counter
[6]LRR: Least Recently Read.
[7]LRU: Least Recently Used.

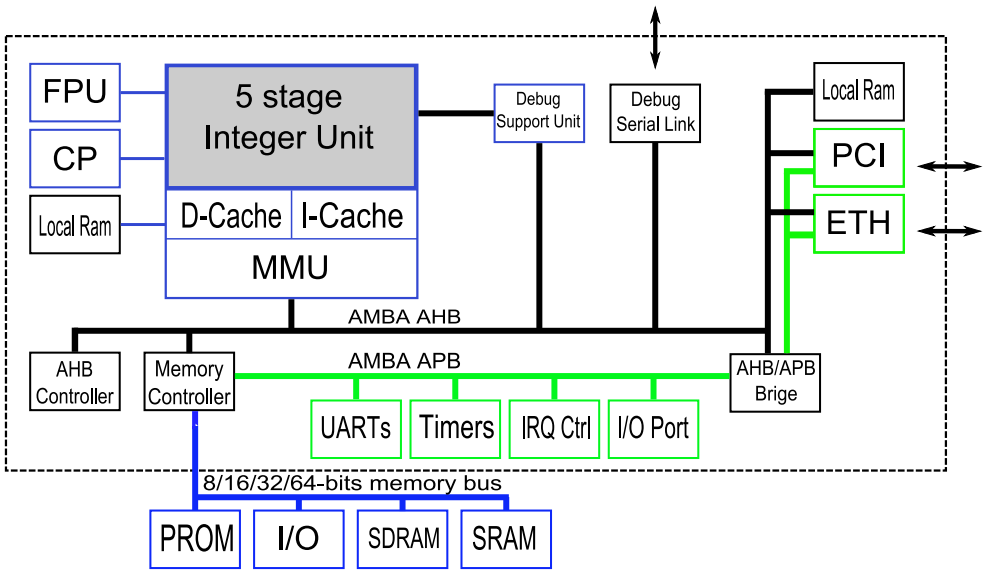


Figura 3.1: Diagrama Leon2 com destaque para o IU.

- Controladores de SDRAM e memória;
- Unidade de ponto flutuante padrão IEEE 754 de alta performance;

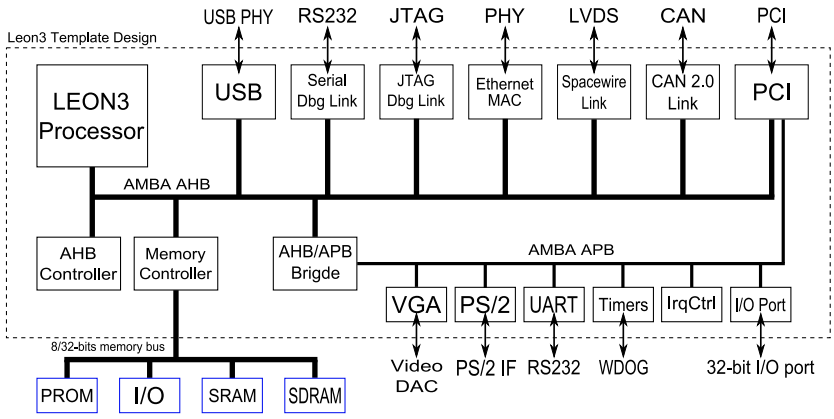
A figura 3.1 mostra um diagrama de blocos do Leon2. Para esse projeto foram captadas apenas a atividade de chaveamento do IU, destacada na figura.

3.1.2 LEON3

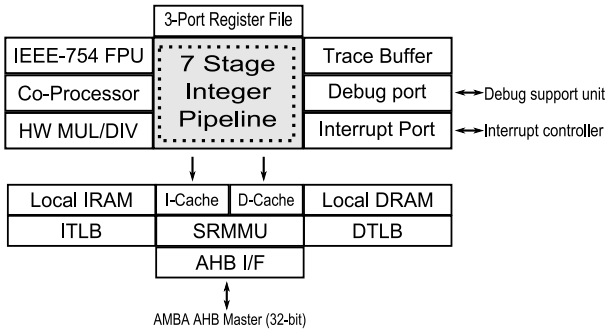
O Leon3 [15] é uma versão aprimorada do Leon2, cujo suporte foi encerrado pela Gaisler [8] no início do ano de 2007, e faz parte do GRLIB, um conjunto de *IP-cores*[8] desenvolvidos para uso em projetos de SoC's. Suas principais características são:

- ISA do SPARCv8;
- Processador com pipeline de 7 estágios;
- Unidades de multiplicação, divisão e MAC em hardware;
- Unidade de ponto flutuante padrão IEEE-754;

[8]IP-cores: Módulo ou função integrável à FPGA.



(a) Leon3



(b) Leon3 Core

Figura 3.2: Diagrama do Leon3 com destaque para o IU.

- Cache de dados e instruções separadas;
- Caches configuráveis: 1-4 conjuntos de 1-256 kbytes/cada com algoritmo de substituição aleatório, LRR ou LRU;
- Suporte à SMP[9];
- Freqüência em FPGA: até 150 MHz; Freqüência em ASIC: até 400 MHz.

A figura 3.2 mostra o diagrama do Leon3. Para esse projeto foram captadas apenas a atividade de chaveamento do IU 3.2(b).

[9]SMP: Symmetric Multi-processor.

3.2 Ferramentas

As ferramentas utilizadas nesse trabalho foram as ferramentas disponíveis no laboratório e disponibilizadas pela fabricante da FPGA para estimativa de consumo de energia para cada tecnologia utilizada. Para FPGAs Xilinx Virtex4 e Spartan3, a ferramenta de estimativa utilizada foi o XPower, da própria Xilinx, disponibilizada juntamente com o ISE. Para ASICs, foi utilizado o PowerCompiler da Synopsys.

3.2.1 XPower

O XPower é uma ferramenta criada pela Xilinx que estima o consumo de energia em suas FPGAs como Spartan3 e Virtex4. FPGAs são circuitos reconfiguráveis utilizados para a implementação de circuitos digitais. As FPGAs da Xilinx são compostos basicamente por três tipos de componentes: os blocos de entrada e saída (IOB), os blocos lógicos configuráveis (CLB) e as chaves de interconexão (Switch Matrix). Os blocos lógicos são dispostos de forma bidimensional, as chaves de interconexão são dispostas em formas de trilhas verticais e horizontais entre as linhas e as colunas dos blocos lógicos.

A utilização do XPower é dada da seguinte maneira. Inicialmente, um dispositivo descrito em VHDL ou Verilog para o qual se deseja estimar a média de consumo de energia é escolhido. Esse modelo é então simulado em alguma ferramenta de simulação, e sua atividade de chaveamento é anotada. Neste projeto foi utilizado o ModelSim da Mentor. Com a atividade de chaveamento em mãos, o dispositivo é então sintetizado pelas ferramentas disponibilizadas pela fabricante da FPGA, que nesse caso foi o ISE WebPack, da Xilinx. A atividade de chaveamento anotada não tem influência sobre a síntese do dispositivo. É feito nessa ordem por pura escolha do usuário. Se houver a disponibilidade de mais computadores, essa tarefa pode ser feita em paralelo.

Os processadores Leon2 e Leon3 disponibilizam junto com seu modelo em VHDL, scripts que automatizam quase todos os passos do fluxo mostrado na figura 3.3. O dispositivo, “Design-Entry”, em questão é a descrição VHDL do processador. Após verificar o projeto pela simulação funcional, é construído uma netlist lógica do processador, a qual é mapeada para um conjunto de componentes característico da FPGA alvo e o último

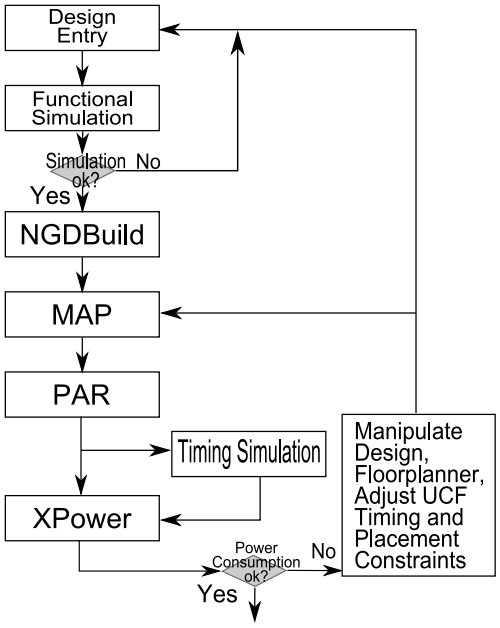


Figura 3.3: Fluxo de uso do XPower

passo é o mapeamento desses componentes para a FPGA, fazendo também as conexões entre os CLBs. Após ser mapeada para a FPGA, o XPower usa esse modelo como uma das entradas. A outra entrada é o arquivo contendo a atividade de chaveamento.

A ferramenta utiliza como entrada os arquivos que contém a atividade de chaveamento e os arquivos que contêm o modelo sintetizado. A figura 3.3 mostra o fluxo de uso do XPower.

A estimativa do consumo de energia fornecida pela ferramenta pode ser de várias formas. Uma delas é o Power Report em modo texto e o valor considerado nesse trabalho é o consumo total, indicado no texto 3.2.1.

3.2.2 PowerCompiler

Power compiler é uma ferramenta da Synopsys para estimar consumo de energia para ASICs. ASICs, ao contrário das FPGAs, são circuitos construídos para executar uma tarefa específica, ou seja, são circuitos dedicados e não podem ser reconfigurados. Isso reduz imensamente os atrasos geradas pelas interconexões comumente presentes nas FPGAs

Texto 3.2.1 Parte do relatório do XPower

```
--------------------------------------------------------------------
Release 8.2i - XPower SoftwareVersion:I.31
Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.
Design:       E:\Projeto\teste\leon_xst.ncd
Preferences:  E:\Projeto\leon.pcf
VCD File:     E:\Projeto\teste\saida_call_tbgen.vcd
Part:         2vp7fg456-7
Data version: ADVANCED,v1.0,05-28-03

XPower and Datasheet may have some Quiescent Current differences.
This is due to the fact that the quiescent numbers in XPower are based on
measurements of real designs with active functional elements reflecting real
world design scenarios.



Power summary:                          I(mA)    P(mW)
--------------------------------------------------------------------
Total estimated power consumption:                 520 <-- Consumo estimado utilizado
                                ---
                      Vccint 1.50V:       61       92
                      Vccaux 2.50V:      167      418
                      Vcco25 2.50V:        4       11
                                ---
       .
       .
       .
           Quiescent Vccint  1.50V:       35       53
           Quiescent Vccaux  2.50V:      167      418
           Quiescent Vcco25  2.50V:        1        3
              Startup Vccint  1.5V:      500
              Startup Vccaux  2.5V:      250
              Startup Vcco25  2.5V:      100
                                ---
Package power limits, ambient 25C:              3333
                          250 LFM:              4196
                          500 LFM:              4615
     .
     .
     .
```

ou arquiteturas não dedicadas.

O PowerCompiler faz parte do DesignCompiler, também da Synopsys, e seu uso e a metodologia utilizada são apresentados nas figuras 3.4 e 3.5. O DesignCompiler sintetiza um dispositivo descrito em VHDL ou Verilog em um ASIC. Para isso são necessárias bibliotecas que contém os dados e especificações da tecnologia alvo.

Nesse projeto a tecnologia alvo para os processadores foi a TSMC .25μm. Uma vez sintetizado, essa informação é usada como entrada para o PowerCompiler que também necessita de um arquivo contendo a atividade de chaveamento do modelo. Caso não seja

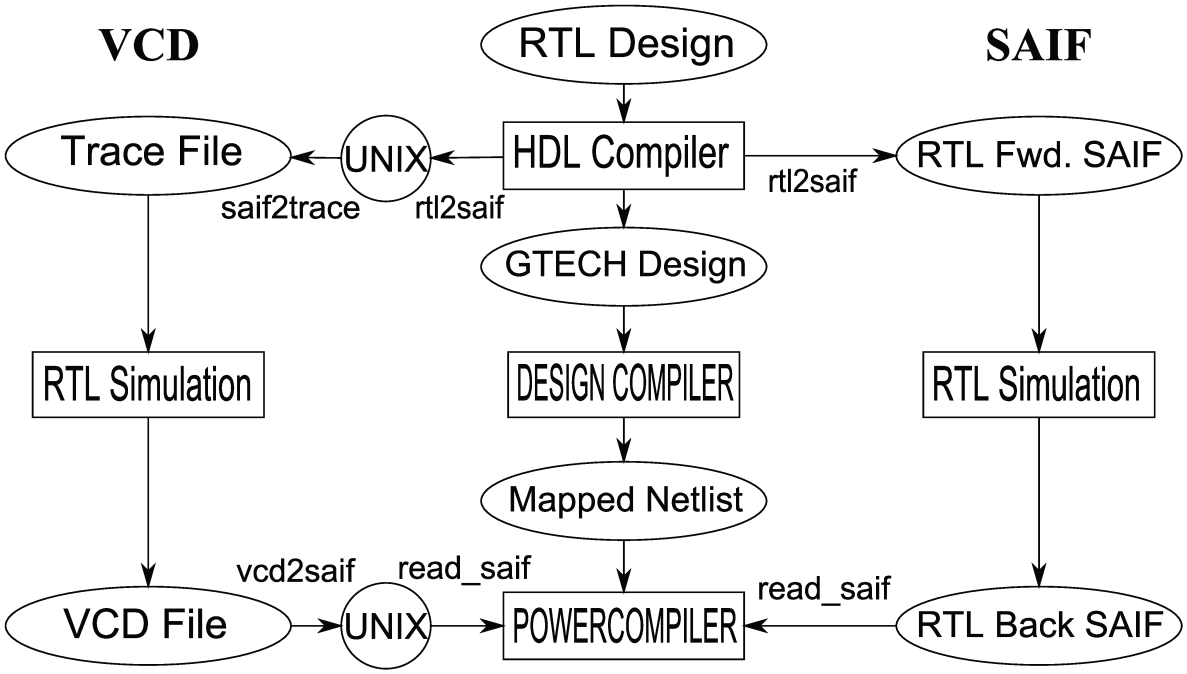


Figura 3.4: Fluxo de uso do PowerCompiler

fornecida a atividade de chaveamento a ferramenta utiliza o padrão que é de 50%. As atividades de chaveamento para o PowerCompiler também foram coletadas utilizando o ModelSim da Mentor.

A figura 3.4 apresenta a utilização do PowerCompiler. Os dois caminhos laterais podem ser utilizados para captar as atividades de chaveamento do dispositivo. A maior diferença é o método de captação e de construção dos arquivos. Enquanto no VCD todos os sinais são anotados por unidade de tempo, no SAIF apenas são anotados os instantes em que algum sinal mudou de estado. Isso faz com que os SAIFs sejam bem menores que os VCDs.

A figura 3.5 apresenta a metodologia utilizada neste projeto para estimar o consumo de energia dos programas testes. Note que foi utilizado apenas o lado direito da figura 3.4. Após obtida as atividades de chaveamento, o caminho central é tomado. A Gaisler fornece juntamente com os Leon2 e Leon3 scripts que automatizam todo o processo até o PowerCompiler.

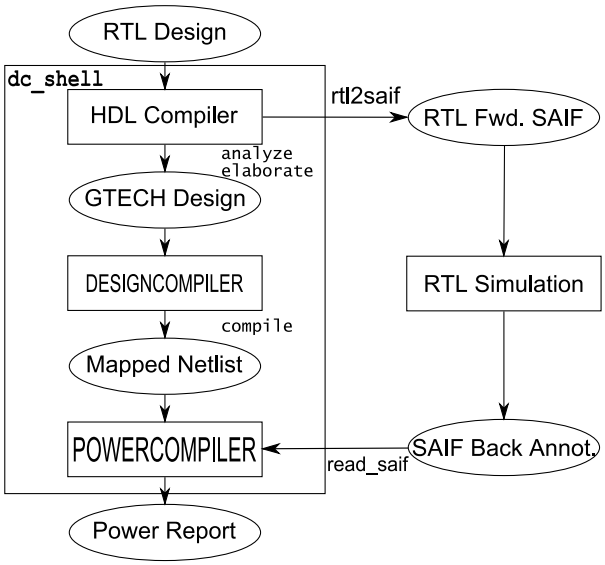


Figura 3.5: Metodologia de uso do PowerCompiler

Os power-reports gerados pelo PowerCompiler são como o modelo do texto 3.2.2 e a estimativa de consumo utilizada por esse trabalho é indicada no texto.

3.3 Metodologia

Nesse projeto foram utilizados os métodos propostos por [7, 21, 26, 27, 28] que utilizam programas alvo para gerar os estímulos necessários. Neste projeto, cada programa contém um núcleo com 100 instruções alvo, e esse núcleo é rodado 1.000 vezes, totalizando 100.000 instruções alvo executadas por simulação. Um gráfico validando o número de instruções executadas por esses programas pode ser visto na figura 4.1 do próximo capítulo.

As instruções alvo para esse projeto foram escolhidas após simulação dos benchmarks Mediabench e MiBench. Após simular os programas com o ArchC 2.0, com suporte a estatísticas ligado e essas guardadas em arquivos, o número de instruções utilizadas foram colocadas em uma tabela, somadas e classificadas em ordem decrescente de uso. A partir daí foram escolhidas as instruções que mais foram executadas. Parte da tabela utilizada na escolha das instruções é mostrada no texto 3.3.1.

Após a escolha das instruções e confecção dos programas, esses foram então compilados

Texto 3.2.2 Relatório do PowerCompiler

```
****************************************
Report : power
-analysis_effort high
Design : leon
Version: X-2005.09-SP4
Date   : Wed Jun 20 19:17:10 2007
****************************************


Library(s) Used:

    vtvtlib25 (File: /home/josuema/P/projeto/leon2/leon2/vtvtlib25.db)


Operating Conditions: nom_pvt   Library: vtvtlib25
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
leon                  05x05              vtvtlib25


Global Operating Voltage = 2.5
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ps
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1mW


  Cell Internal Power  =   1.1680  W   (68%)
  Net Switching Power  = 544.0558 mW   (32%)
                         ---------
Total Dynamic Power    =   1.7120  W  (100%) <-- Consumo estimado utilizado

Cell Leakage Power     =   6.6462 uW
```

utilizando o sparc-elf disponibilizado pela Gaisler. Nesse compilador foi utilizada a opção '-mv8' para gerar instruções da versão 8 do SPARC. Caso essa opção não seja utilizada, o compilador utiliza a versão 7 como padrão. A versão 8 do SPARC dá suporte às novas instruções diretamente. Um dos exemplos é o suporte à instrução de multiplicação ('mul') utilizada. Depois de compilados, foram simulados no Leon2 e Leon3 em tecnologias diferentes, como Virtex4, Spartan3 e em TSMC .25μm, utilizando o ModelSim, de onde são gerados os arquivos contendo as atividades de chaveamento nos formatos SAIFs e VCDs.

Com os arquivos de atividades de chaveamento em mãos, essas informações foram repassadas para o PowerCompiler e para o XPower para serem geradas as estimativas

Texto 3.3.1 Parte da somatória dados da saída estatística do ArchC após simulação do MediaBench e MiBench em ordem decrescente.

```
[ArchC 2.0] Printing statistics from instruction or_reg:    3098659546
[ArchC 2.0] Printing statistics from instruction or_imm:    1694779916
[ArchC 2.0] Printing statistics from instruction ld_imm:    1510735531
[ArchC 2.0] Printing statistics from instruction add_reg:   1509612327
[ArchC 2.0] Printing statistics from instruction subcc_imm: 1361816541
[ArchC 2.0] Printing statistics from instruction srl_imm:   1351721320
[ArchC 2.0] Printing statistics from instruction add_imm:   1332784107
[ArchC 2.0] Printing statistics from instruction subcc_reg: 1214230685
[ArchC 2.0] Printing statistics from instruction st_imm:    943804441
[ArchC 2.0] Printing statistics from instruction sethi:     938418778
[ArchC 2.0] Printing statistics from instruction and_imm:   789807418
[ArchC 2.0] Printing statistics from instruction be:        734652605
[ArchC 2.0] Printing statistics from instruction bleu:      675137290
[ArchC 2.0] Printing statistics from instruction std_imm:   660026488
[ArchC 2.0] Printing statistics from instruction sll_imm:   613318564
[ArchC 2.0] Printing statistics from instruction bne:       560639513
[ArchC 2.0] Printing statistics from instruction ba:        512593082
[ArchC 2.0] Printing statistics from instruction ldd_imm:   498868098
[ArchC 2.0] Printing statistics from instruction and_reg:   425264952
[ArchC 2.0] Printing statistics from instruction jmpl_imm:  345076699
[ArchC 2.0] Printing statistics from instruction call:      344521495
[ArchC 2.0] Printing statistics from instruction save_imm:  315992979
(...)
```

de consumo de energia. Essas estimativas foram então colocadas em uma tabela base, contendo cada uma das instruções e suas respectivas estimativas de consumo.

A figura 3.6 mostra em parte a metodologia utilizada para captação de dados no projeto e a tabela 4.1 da seção 4.1 mostra a estimativa de consumo de energia por classes de instruções.

Essa metodologia segue as propostas de [13] descritas na seção 2.3. Segundo as características descritas pode se dizer que a metodologia utilizada é responsável, pois estima o consumo de energia em nível de instruções com um erro médio de $\pm 6\%$ mesmo não sendo capaz de identificar a significância de todos os fatores que afetam o consumo de energia do processador, resultados que podem ser visualizados no capítulo 4. É também simplista,

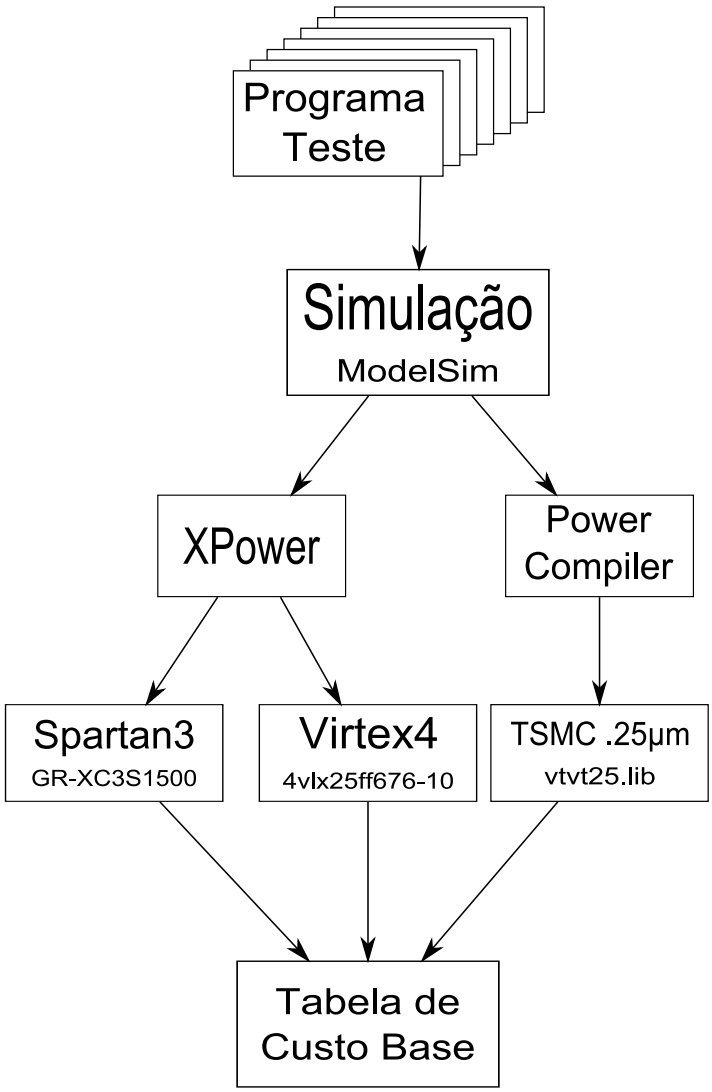


Figura 3.6: Metodologia utilizada para captação de dados

A função *function(X, Y)* tem como objetivo utilizar um meio mais automatizado para obter a quantidade de energia consumida por uma instrução *X* em uma tecnologia *Y*.

Como existe uma diferença de consumo para cada tecnologia ou arquitetura utilizada, implementar a idéia mostrada no algoritmo 3.4.1 (que é usar um registrador global e um valor estático) é pouco viável, pois para cada tecnologia ou arquitetura diferenciada seria necessário percorrer todos os comportamentos das instruções para modificar o custo-base utilizado. A idéia apresentada no algoritmo 3.4.2 resolve parte desse problema, porém, nesses dois casos, a maior dificuldade é modificar o conteúdo dos comportamentos de todas as instruções.

O ArchC, em sua versão 2.0 [1], faz uma varredura e cálculo de estatísticas como número de instruções executadas e quantidade de vezes que cada instrução foi executada em um determinado programa. Com esses dados é possível utilizar um arquivo contendo os dados de consumo de energia de cada instrução como entrada e utilizar esses valores para o cálculo, modificando o menos possível a estrutura interna do ArchC. Nesse caso, a modelagem do processador e do comportamento das instruções continua a mesma. Uma vez pronta, não é necessário modificar a parte comportamental. Para ter acesso aos dados de consumo de uma nova tecnologia basta utilizar um novo arquivo de entrada contendo os custos base.

3.5 acpower

O ***acpower*** é uma ferramenta desenvolvida nesse trabalho levando em conta as idéias apresentadas nos capítulos anteriores tais como ser simples, rápida e fácil de ser utilizada. E também leva em conta a dificuldade existente para estimar o consumo de energia se a ferramenta de estimativa estiver internamente ligada ao simulador. O ***acpower*** leva em consideração o número de instruções utilizado, quantas vezes cada instrução foi utilizada, o custo base de cada instrução e a freqüência de operação.

O ***acpower*** é um *script* descrito em Python que utiliza o ArchC que, em sua nova versão já disponibiliza as estatísticas necessárias para fazer o cálculo da estimativa de consumo de energia em nível de instrução, através de um arquivo que contém a tabela de

custos base e um arquivo com as saídas estatísticas do ArchC. Como a saída estatística do ArchC é feita como as saídas de erro do sistema (em C++: cerr<<), para obtê-las é utilizado o comando shell do linux: 2>&1 | tee [arquivo contendo saída estatística].

Uma vez obtidas as estatísticas de um determinado programa, não é necessário re-simulaá-lo para obter sua estimativa de consumo, economizando tempo precioso de simulação e computação.

Para funcionar corretamente, os arquivos de entrada do ***acpower*** devem seguir algumas características que serão mostradas na seção 3.5.1.

3.5.1 Arquivos de Entrada

Um exemplo das entradas utilizadas pelo ***acpower*** é mostrado nos textos 3.5.1 e 3.5.2 a seguir. A saída estatística do ArchC contém a maioria das informações necessárias para o cálculo da estimativa de consumo de energia em nível de instruções, pois descreve detalhadamente o número total de instruções executadas e quantas vezes cada instrução foi executada. Já o arquivo que contém todas as instruções e suas respectivas médias de consumo completa as informações necessárias para a estimativa.

O texto 3.5.2 representa o arquivo de custos base por instrução em sua forma básica.

O cabeçalho é formado por informações do consumo de energia. As palavras reservadas iniciam-se com o símbolo '#' e são:

- **PROCESSOR :** Descreve o processador utilizado.
  Ex: SPARCv8, SPARCv8 Leon2, OR1k, OR1200, etc...
- **DEVICE :** Descreve a tecnologia utilizada para a estimativa.
  Ex: SPARTAN3 3s1500fg456-4, TSMC .25, etc...
- **POWER :** Descreve a unidade dos valores do consumo de energia.
  Ex: mW, W, MG, etc...
- **FREQ :** Indica a freqüência utilizada na captação dos dados.
  Ex: 50, 75, 1200, ... ou 'default'

Texto 3.5.1 Parte da saída estatística do ArchC

```
            SystemC 2.1.v1 --- Oct 29 2006 15:13:41
        Copyright (c) 1996-2005 by all Contributors
                    ALL RIGHTS RESERVED
ArchC: Reading ELF application file: mediabench/mpeg2/bin/mpeg2encode

ArchC: -------------------- Starting Simulation --------------------

ArchC Warning: This version of fstat should not be called!
ArchC Warning: Please, recompile your target application with an updated libac_sysc.
Encoding frame 0 .................
Encoding frame 1 .................
Encoding frame 2 .................
Encoding frame 3 .................
ArchC: -------------------- Simulation Finished --------------------
SystemC: simulation stopped by user.
ArchC: Simulation statistics
    Times: 3481.42 user, 3.15 system, 3513.98 real
    Number of instructions executed: 10834233858
    Simulation speed: 3112.01 K instr/s

[ArchC 2.0] Printing GLOBAL statistics from processor module sparcv8:
     INSTRUCTIONS : 10834233124
     SYSCALLS : 734
[ArchC 2.0] Printing INSTRUCTION statistics from processor module sparcv8:
[ArchC 2.0] Printing statistics from instruction call:
     COUNT : 115695311
[ArchC 2.0] Printing statistics from instruction nop:
     COUNT : 31380216
[ArchC 2.0] Printing statistics from instruction sethi:
     COUNT : 466441373
[ArchC 2.0] Printing statistics from instruction ba:
.
.
.
```

- **SCALE :** Indica a unidade de freqüência utilizada.
  Ex: Hz, KHz, MHz, etc...

- **a :** Fator de correção(α).
  Ex: 0,7; 0,8 etc...

O último item α existe, pois o crescimento do consumo em relação à freqüência não é linear, algo que pode ser visto nas figuras 4.2 e 4.3. Caso não haja especificidade da freqüência utilizada na hora da captação dos dados da tabela de custos base, é utilizada a palavra 'unknown'.

Nesse arquivo foi determinado que qualquer comentário é iniciado com '%' e comentários só podem ser inseridos no início de cada linha. Como mostrado em [26], esse

Texto 3.5.2 Parte do arquivo que contém os consumos das instruções

```
% ---- Inicio do cabeçalho ----
% Comentários iniciam-se com "%"
#PROCESSOR = LEON3 SPARCv8(2)
#DEVICE = SPARTAN3 3s1500fg456-4
#POWER = mW
#FREQ = 50
#SCALE = MHz
#a = 0.982
% ---- fim do cabeçalho -----
ADD(add_reg, addcc_reg, addx_reg, addxcc_reg, addxcc_imm, addx_imm, addcc_imm, add_imm)=
204
AND(and_reg, andcc_reg, andn_reg, andncc_reg, and_imm, andcc_imm, andn_imm, andncc_imm)
% o símbolo '=' pode ser omitido
204
BRANCH(ba, bn, bne, be, bg, ble, bg, bgu, bleu, bcc, bcs, bpos, bneg, bvc, bvs)
204
CALL(call)
202
.
.
.
```

trabalho também identificou uma certa semelhança entre instruções do mesmo tipo, por essa razão o arquivo de custos base utiliza-se de classes de instruções para o cálculo. Cada linha do arquivo inicia-se com o nome da classe seguido das instruções pertencentes a ela entre parênteses separadas por vírgulas. Para facilitar a visualização e a confecção do arquivo, os valores do consumo de energia de cada classe de instruções são colocados uma linha abaixo.

3.5.2 Funcionamento

O ***acpower*** foi desenvolvido para funcionar das seguintes maneiras:

- **2 entradas:**

  ```
  $ acpower arquivoDeCustosBase arquivoSaidaDoArchC
  ```

  Nesse modo, o ***acpower*** calcula o consumo de energia baseado nos arquivos de entrada, o arquivo de custos base e o arquivo contendo as estatísticas, e dá o resultado em tela e em um arquivo saída padrão (power_report.acpwr).

- **3 entradas:**

```
$ acpower arquivoDeCustosBase arquivoSaidaDoArchC novaFrequencia
```

  Nesse modo, o ***acpower*** calcula o consumo de energia baseado nos arquivos de entrada, recalcula a estimativa para a nova freqüência dada e dá o resultado em tela e em um arquivo saída padrão, chamado power_report.acpwr.

- **4 entradas:**

```
$ acpower arquivoCustoBase arquivoSaidaArchC novaFrequencia arquivoSaida
```

  Nesse modo, o ***acpower*** calcula o consumo de energia baseado nos arquivos de entrada, recalcula a estimativa para a nova freqüência dada e dá o resultado em tela e em um arquivo saída com o nome customizado.

Caso a freqüência no arquivo de custo base seja 'unknown', a nova freqüência dada pelo usuário será multiplicativa. Por exemplo, caso seja dado o valor 2 na entrada e no arquivo de custos base a freqüência seja 'unknown', o ***acpower*** irá calcular o resultado para o dobro da freqüência base. Caso a freqüência inserida pelo usuário seja igual à da configuração original, o ***acpower*** irá desconsiderar a nova freqüência.

A figura 3.7 mostra um diagrama de fluxo de uso do ***acpower***, com seus conjuntos de entrada e sua saída.

A vantagem desse modelo para o ***acpower*** é que não é necessária a re-simulação de um programa para obter sua estimativa de consumo de energia. Uma vez que se tenham as estatísticas em mãos, é possível, apenas mudando o arquivo de entrada (que contém as informações do consumo de energia de uma outra tecnologia ou arquitetura), obter a nova estimativa de consumo quase instantâneamente.

Nesse capítulo foram apresentados os aspectos práticos desse trabalho como o processador e as ferramentas utilizadas, os meio de captação de dados, a metodologia utilizada

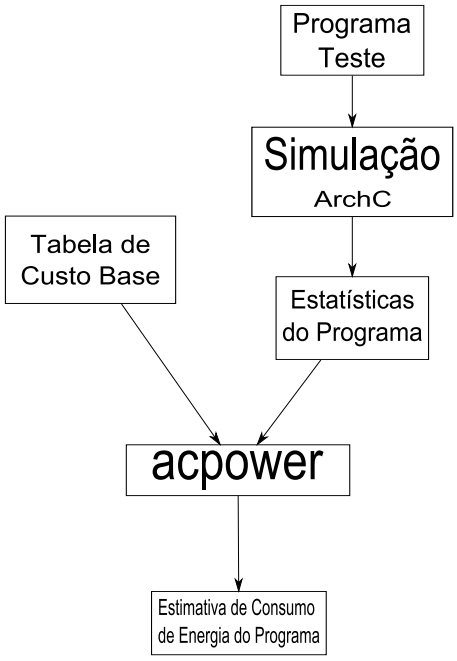


Figura 3.7: Fluxo de uso do ***acpower***

e a ferramenta desenvolvida. Também foi mostrado o funcionamento e a confecção do arquivo de custos base utilizada pela ferramenta. No próximo capítulo serão apresentados os resultados e as validações do sistema desenvolvido.

Capítulo 4

Resultados Experimentais

Nesse capítulo serão apresentados os resultados experimentais obtidos com o uso do ArchC para estimativa de consumo de energia, apresentado no capítulo anterior. Para a avaliação desses resultados foram utilizados como base de comparação os resultados obtidos pelo uso do PowerCompiler [9].

A alimentação do PowerCompiler foi feita através da simulação e captação de atividade de chaveamento utilizando o ModelSim 6.1b [4]. Foi utilizado o mesmo conjunto de programas para validar os testes. Todas as medições de tempo foram feitas utilizando-se o mesmo computador, com as seguintes configurações: processador Pentium IV HT 2.8 GHz 1024 MB RAM. A máquina em questão foi isolada durante os testes de desempenho.

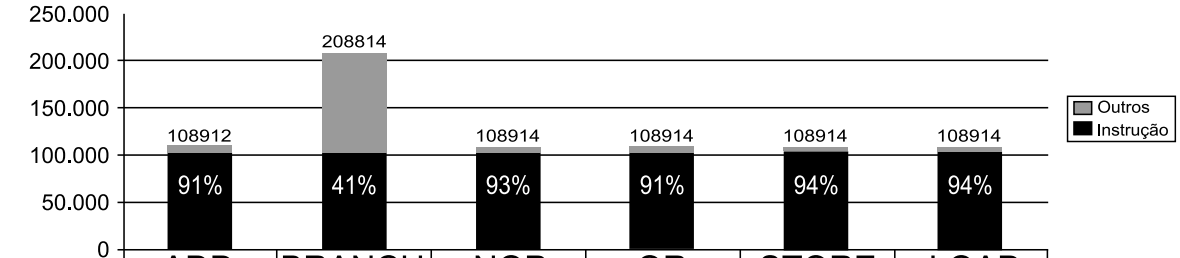
4.1 Validação dos programas utilizados

Para validar os programas de testes foi utilizado o recurso de estatística do ArchC. O núcleo dos programas testes é um laço que contém 100 instruções de um mesmo tipo. Este laço é executado 1.000 vezes, totalizando 100.000 execuções de uma mesma instrução, mostrado pelo algoritmo 4.1.1.

Nesse algoritmo é mostrado o programa base para a instrução ADD. Todos os outros programas base tem o mesmo cabeçalho e rodapé, onde .LL2 e .LL5 são responsáveis pela iteração e, main e .LL3 são as instruções de controle de início e fim de programa. Para os testes das demais instruções a mudança se limita às instruções do núcleo do programa.

Em .LL2, a instrução cmp %g1,999 é responsável pelas iterações.

O laço tem o tamanho suficiente e é executado um número de vezes suficiente para que o *overhead* das instruções de controle seja mínimo. Como pode ser visto no gráfico da figura 4.1, os núcleos dos programas são responsáveis por ±90% das instruções executadas pelo programa. Com exceção da instrução BRANCH que, para cada desvio, necessita de um NOP para funcionar corretamente.



	ADD	BRANCH	NOP	OR	STORE	LOAD
Outros	9012	106915	7012	9013	6014	6014
Instrução	99900	101899	101902	99901	102900	102900

Figura 4.1: Validação do método utilizando estatísticas de alguns dos programas de teste

Com base nessas informações é possível afirmar que o responsável pelo consumo do programa é a instrução que está sendo executada no núcleo.

A tabela 4.1 mostra o consumo de energia por classe de instrução para cada um dos processadores, Leon2 e Leon3, em cada uma das diferentes tecnologias, com freqüências próximas à 50MHz (52 MHz em FPGA, Virtex4 e Spartan3; e 50 MHz em ASIC, TSMC .25μm) obtidas pelas ferramentas PowerCompiler e XPower. As instruções pertencentes à cada classe utilizada são:

- **ADD:** add_reg, addcc_reg, addx_reg, addxcc_reg, addxcc_imm, addx_imm, addcc_imm, add_imm.
- **AND:** and_reg, andcc_reg, andn_reg, andncc_reg, and_imm, andcc_imm, andn_imm, andncc_imm.
- **BRANCH:** ba, bn, bne, be, bg, ble, bg, bgu, bleu, bcc, bcs, bpos, bneg, bvc, bvs.

Algoritmo 4.1.1 Exemplo de um dos núcleos dos programas base utilizados

```
   .file "call.s"
   .section ".text"
   .align 4
   .global main
   .type main, #function
   .proc 04

main:
   !#PROLOGUE# 0
   save %sp, -120, %sp
   !#PROLOGUE# 1
   mov 1, %g1
   st %g1, [%fp-20]
   mov 1, %g1
   st %g1, [%fp-12]

.LL2: // Variáveis de controle
   ld [%fp-12], %g1
   cmp %g1, 999 // Comparação
   bg .LL3
   nop

.LL4: // Início do núcleo
   add %g0, %g1, %g2
   add %g1, %g2, %g3
   add %g2, %g3, %g4
   add %g3, %g4, %g5
   add %g4, %g5, %g6
   .
   .
   .
   add %l0, %l1, %l2
   add %l1, %l2, %l3
   add %l2, %l3, %l4
   add %l3, %l4, %l5 // Fim do´núcleo

.LL5:
   ld [%fp-12], %g1
   add %g1, 1, %g1 // Iteração
   st %g1, [%fp-12]
   b .LL2
   nop

.LL3: // Saída
   mov 0, %g1
   mov %g1, %i0
   ret
   restore
   .size main, .-main
   .ident "GCC: (GNU) 3.4.4"
```

- **CALL:** call.
- **CMP:** cmp.
- **DIV:** udiv_reg, udivcc_reg, sdiv_reg, sdivcc_reg, udiv_imm, udivcc_imm, sdiv_imm, sdivcc_imm, stb_imm.
- **JMP:** jmpl_reg, jmpl_imm.
- **LOAD:** ldsb_reg, ldsh_reg, ldub_reg, lduh_reg, ld_reg, ldd_reg, ldsb_imm, ldsh_imm, ldub_imm, lduh_imm, ld_imm, ldd_imm, ldstub_imm, ldstub_reg.
- **MOV:** mov.
- **NOP:** nop.
- **OR:** or_reg, orcc_reg, orn_reg, orncc_reg, or_imm, orcc_imm, orn_imm, orncc_imm.
- **RESTORE:** restore_reg, restore_imm.
- **SAVE:** save_reg, save_imm.
- **SETHI:** sethi.
- **SHIFT:** sll_reg, srl_reg, sra_reg, sra_imm, srl_imm, sll_imm.
- **SMUL:** smul_reg, smulcc_reg, mulscc_reg, smul_imm, smulcc_imm, mulscc_imm.
- **STORE:** sth_imm, st_imm, std_imm, stb_reg, sth_reg, st_reg, std_reg.
- **SUB:** sub_reg, subcc_reg, subx_reg, subxcc_reg, subxcc_imm, subx_imm, subcc_imm, sub_imm.
- **UMUL:** umul_reg, umulcc_reg, umul_imm, umulcc_imm.
- **XOR:** xor_reg, xorcc_reg, xnor_reg, xnorcc_reg, xor_imm, xorcc_imm, xnor_imm, xnorcc_imm.
- **UNIMP:** trap_imm, trap_reg, unimplemented, wry_imm, rdy, swap_imm, wry_reg, swap_reg.

	Resultados - Consumo por Classes de Instruções				
	Leon2			Leon3	
	FPGA(mW)		ASIC(W)	FPGA(mW)	ASIC(W)
	Spartan3	Virtex4	TSMC .25μm	Spartan3	TSMC .25μm
add	182	507	1,60	204	1,51
and	182	507	1,74	204	1,48
branch	181	507	1,68	204	1,51
call	171	501	1,64	204	1,52
cmp	182	507	1,65	204	1,53
div	179	505	1,65	205	1,53
jump	195	514	1,71	216	1,48
load	242	539	1,67	225	1,52
mov	182	507	1,66	204	1,53
nop	182	507	1,66	204	1,51
or	182	507	1,68	204	1,53
restore	182	507	1,64	204	1,51
save	182	507	1,64	204	1,51
sethi	182	507	1,64	204	1,50
shift	182	507	1,63	204	1,50
smul	164	497	1,64	201	1,54
store	242	539	1,67	225	1,52
sub	182	507	1,59	204	1,57
umul	164	497	1,64	201	1,50
xor	182	507	1,63	204	1,52
unimpl	0	0	0	0	0

Tabela 4.1: Consumo aferido por classe de instrução.

4.2 Validação das ferramentas

Para validar as ferramentas foi utilizado um mesmo programa teste, apenas variando a freqüência do processador. Essa validação utilizou a ferramenta PowerCompiler, tecnologia TSMC 0.25 e variação de freqüência de 25 a 200 MHz. O gráfico 4.2, valida a ferramenta pois, mantendo a capacitância do sistema, a tensão de funcionamento e o *toggle rate*, a freqüência de operação é a responsável pelo consumo de energia do sistema, como já descrito na fórmula 2.1 da seção 2.1, que para facilitar sua visualização é repetida abaixo.

$$P_{din} = C_L V_{dd}^2 f T_{Rate} \tag{4.1}$$

A figura 4.2 mostra o crescimento do consumo do programa da instrução ADD em comparação com uma linha base, de crescimento linear. Já a figura 4.3 demonstra esse crescimento para um conjunto de programas tais como ADD, LOAD e Hello World, também em comparação com uma reta de crescimento linear.

Caso o crescimento fosse linear, a equação 2.1 poderia ser reduzida à seguinte equação onde a constante K substituiria $C_L V_{dd}^2 T_{Rate}$.

$$P = \frac{f_{new}}{f_{old}} K \tag{4.2}$$

Nas figuras 4.2 e 4.3 a linha tracejada representa a função 4.2, como na teoria. Na prática, porém, foi necessária a criação de um fator de correção α no ***acpower***, já descrito na seção 3.5.1.

A equação 4.3 foi construída após os resultados experimentais e implementada no ***acpower*** para se calcular a estimativa de consumo de energia em uma freqüência diferente da estabelecida no arquivo de entrada.

Esses resultados foram obtidos utilizando como programas testes o programa ADD, LOAD e o Hello World! e todos utilizaram como ferramenta o PowerCompiler, como tecnologia o TSMC .25μm e como freqüência de testes 25, 50, 75, 100, 125, 150, 175 e 200 MHz, como podem ser vistos nas figuras 4.2 e 4.3.

Essa função pode ser visualizada pela reta pontilhada no gráfico da figura 4.3. Para a tecnologia TSMC .25μm o valor de α é 0,76689883. Esse fator ainda não foi calculado para as FPGAs.

$$P = \frac{f_{new}}{f_{old}} \alpha K \tag{4.3}$$

4.3 Avaliação do ArchC e acpower

Nessa seção serão demonstrados os resultados da utilização do ArchC e do ***acpower***.

A tabela 4.2 mostra a diferença entre os tempos de execução de alguns programas, utilizando ModelSim e ArchC, simulando um processador SPARCv8 e os mesmos programas quando compilados em C e executados em uma máquina qualquer. Os programas

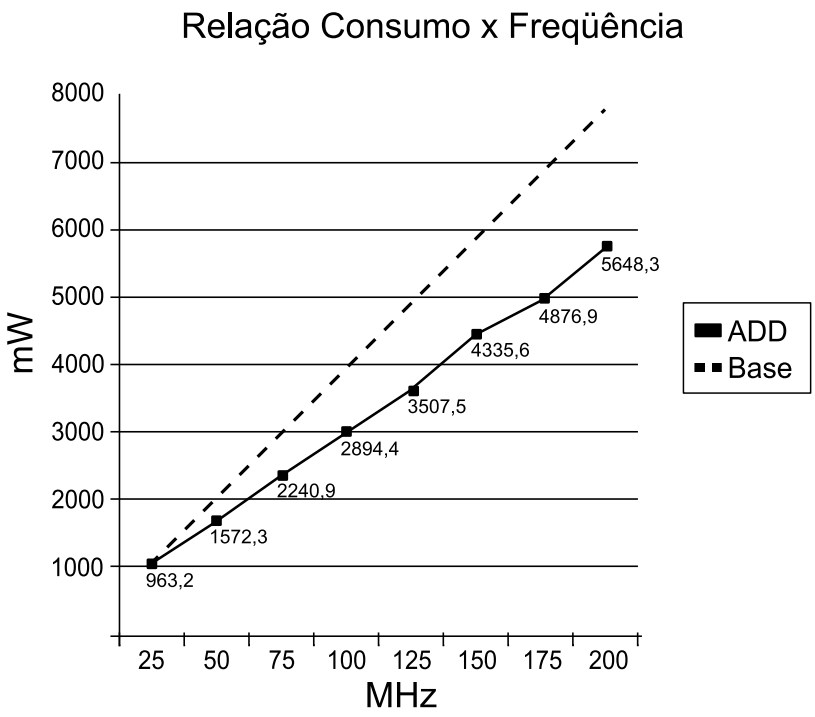


Figura 4.2: Comparativo do crescimento linear e real do consumo de energia.

utilizados como teste são programas bem básicos como Hello World! e um programa que multiplica duas matrizes 20x20 com valores predefinidos. Esses dois programas serão utilizados como base para comparação de alguns outros resultados mostrados abaixo. Foram escolhidos por serem simples e de rápida simulação, mesmo assim a simulação do programa de matrizes demorou 1 hora para finalizar no ModelSim. Como teste final de eficiência, foi utilizado o programa qsort_small, presente no MediaBench ou no MiBench. Para a utilização desse programa foram necessárias algumas modificações como por exemplo a utilização de um vetor com apenas 128 palavras inclusas no código fonte, ao invés da leitura de arquivo utilizado pelo programa original. Para a captação de tempo foi utilizado o comando *time*. O campo SpeedUp(X) mostra quantas vezes o ArchC foi mais rápido que o ModelSim na simulação do processador.

Como pôde ser visto, a diferença de tempo de simulação é considerável e o crescimento é não-linear. Esse crescimento depende do nível de simulação utilizado. Quanto mais alto o nível, mais rápida é a simulação. Esse é um dos fatores positivos em se utilizar de uma

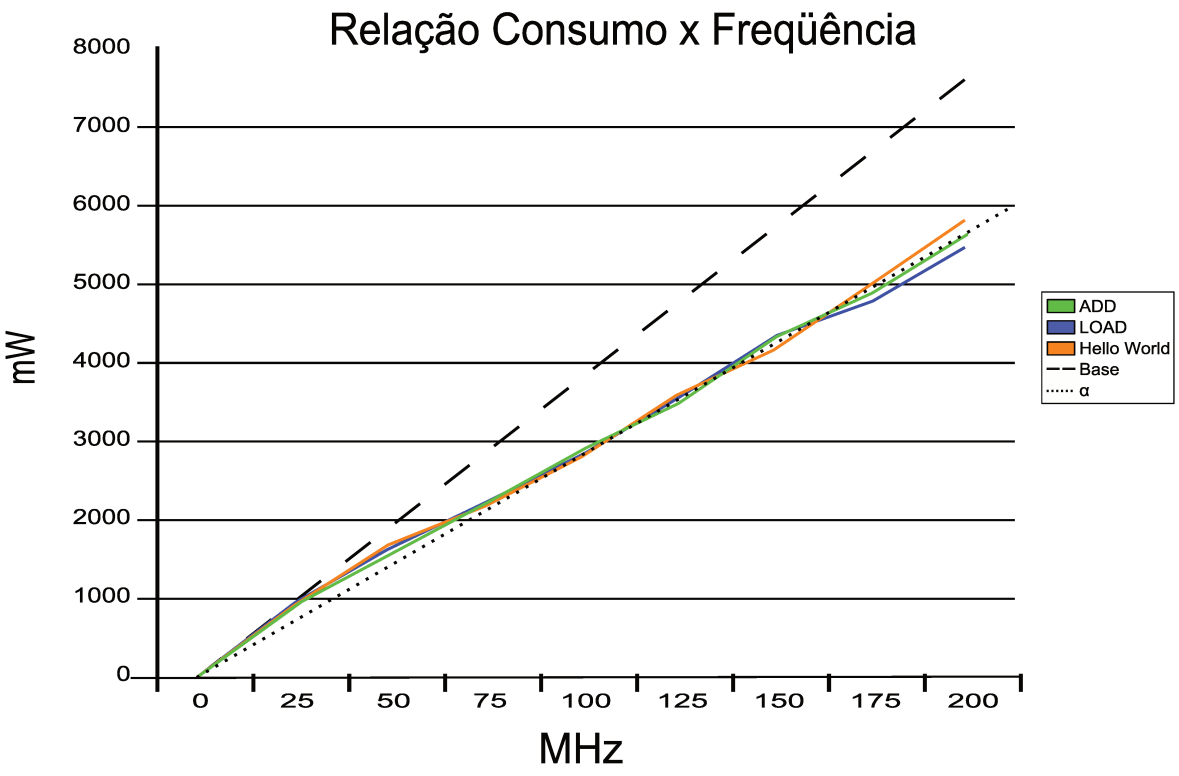


Figura 4.3: Comparativo do crescimento linear, real e utilização do fator α.

ADL para simulação e do nível de instruções para o cálculo da estimativa de consumo de energia.

A tabela 4.3 compara os tempos gastos com a simulação para captação da atividade de chaveamento, utilizando ModelSim, com a estimativa de consumo de energia, utilizando o PowerCompiler, com a simulação no ArchC e o com a estimativa utilizando o ***acpower***. Observando a tabela é possível reafirmar que quanto mais alto o nível de abstração, mais rápida será sua simulação e resultado. Devido ao tempo gasto com o qsort modificado, este não foi re-simulado utilizando ModelSim para Leon2 nem para captação de VCDs.

A tabela 4.4 faz uma última comparação entre o tempo gasto com a estimativa no método convencional e o tempo gasto com o ArchC (simulação + ***acpower***). É possível afirmar que o tempo economizado utilizando o ArchC para estimar o consumo de energia compensa o erro médio de ±5%, como fator de comparação, o PowerCompiler utilizado

	Tempo de execução do programa			
	C	ModelSim	ArchC	SpeedUp(X)
qsort (modificado)	0,016	2,5 dias	3,288	65693
Matriz	0,002	3540	0,507	6982
Hello World	0,002	28	0,255	109

Tabela 4.2: Tempo de execução (em segundos).

	Tempo de Simulação (s)					
	Leon2		Leon3		ArchC	
	MSim	PCompiler	MSim	PCompiler	Simulação	acpower
qsort	-	-	2,5 dias	1857	3,288	0,031
Matriz	3540	630	1680	1860	0,507	0,031
Hello World	28	612	18	1860	0,255	0,031

Tabela 4.3: Tempo de simulação (em segundos).

como base tem erros médios de ±20%. Como o qsort modificado não foi re-simulado para captação de VCDs, não há resultado comparativo utilizando XPower. Os resultados apresentados aqui são do Leon2, utilizando tecnologia TSMC .25μm, 50 MHz, no PowerCompiler e Virtex4, 52 MHz, no XPower. Para o qsort modificado, o único resultado disponível é do PowerCompiler, utilizando Leon3 a 50 Mhz na tecnologia TSMC .25μm.

	Resultados Experimentais						
	Método Convencional				acpower		ERRO
	XPower		PCompiler				
	Cons(mW)	T(s)	Cons(mW)	T(s)	Cons(mW)	T(s)	%
qsort	-	-	1646	1857	1534	0,031	6,8
Matriz	-	-	1644	630	1606	0,031	2,3
Hello World	-	-	1666	612	1652	0,031	0,8
Matriz	507,3	90	-	-	497,6	0,031	1,9
Hello World	506,5	72	-	-	513,3	0,031	1,3

Tabela 4.4: Comparativo entre tempos e resultados de estimativa de consumo de energia das ferramentas.

A tabela 4.5 mostra os consumos médios dos benchmarks Mediabench e Mibench. Os programas comuns aos dois benchmarks não foram re-simulados.O consumo médio do Leon3 em Virtex4 não pôde ser calculado devido à versão do ISE utilizada. As bibliotecas de modelos Virtex4 disponíveis no WebPack não eram grandes o suficiente para sintetizá-

lo.

Para obter as estatísticas do lame large foram necessários 7 horas de simulação em ArchC.

Essa tabela apresenta apenas os resultados obtidos pelo uso do ***acpower***. A maior característica do ***acpower*** é que, uma vez tendo o modelo de consumo de energia e a saída estatística do ArchC, ele não necessita de uma re-simulação para calcular o consumo para um mesmo programa. Apenas se troca o arquivo que contém as novas informações ou a freqüência de operação do dispositivo, e se tem a nova estimativa. Assim, o tempo total para calcular toda a tabela utilizando ***acpower*** foi de apenas 70 segundos. Se o ***acpower*** estivesse atrelado internamente ao ArchC, seriam necessárias 35 horas de simulação apenas para calcular a estimativa do programa lame large. Analisando a tabela 4.2 é possível estimar que seriam necessários mais de 14 dias (ou mais de 420 horas) para obter a atividade de chaveamento do mesmo programa utilizando o Modelsim.

Uma característica interessante apontada pela tabela é o fato do consumo na FPGA ser menor do que o apresentado pela ASIC. Como a freqüência utilizada pelas FPGAs são compatíveis com a utilizada pela ASIC, $\pm$50 MHz, essa diferença se deve ao fato da tecnologia utilizada pelos fabricantes. Enquanto o TSMC é fabricado utilizando .25μm e as PFGAs Virtex4 e Spartan3 são fabricadas utilizando tecnologia 90nm.

Nesse capítulo foram apresentados os resultados práticos desse trabalho, as validações e resultados das ferramentas utilizadas e desenvolvidas, comparações de desempenho e resultados comparativos para pequenos programas.

	Resultados Experimentais - Benchmarks				
	Leon2			Leon3	
	FPGA(mW)		ASIC(W)	FPGA(mW)	ASIC(W)
	Spartan3	Virtex4	TSMC .25μm	Spartan3	TSMC .25μm
MediaBench					
adpcm coder	182,36	499,12	1,61	201,39	1,50
adpcm decoder	194,96	532,91	1,73	215,08	1,59
adpcm timing	186,14	509,40	1,64	205,55	1,52
cjpeg	190,50	503,21	1,60	204,22	1,49
djpeg	192,61	514,70	1,64	209,43	1,52
gsm toast	191,49	509,28	1,62	206,77	1,51
gsm untoast	187,04	509,63	1,63	205,88	1,52
mpeg encoder	192,83	508,76	1,64	206,54	1,50
mpeg decoder	190,40	509,47	1,65	206,52	1,51
pegwit decript	197,88	513,71	1,64	208,98	1,51
pegwit encript	196,72	513,74	1,64	208,79	1,51
pegwit generate	195,21	511,60	1,63	207,81	1,51
MiBench					
basicmath	188,45	507,16	1,64	205,28	1,51
bitcount large	183,50	503,55	1,63	203,34	1,50
bitcount small	183,57	503,58	1,63	203,37	1,50
quicksort large	201,82	525,07	1,68	213,75	1,55
quicksort small	202,80	528,49	1,69	215,10	1,56
susan corners	194,15	502,95	1,60	204,80	1,48
susan edges	189,95	506,53	1,62	205,33	1,50
susan smoothing	193,30	514,74	1,64	209,05	1,53
lame large	189,67	505,88	1,63	205,08	1,50
lame small	189,31	505,95	1,63	205,04	1,50
djikstra large	199,53	516,46	1,65	210,33	1,52
dijkstra small	199,13	516,14	1,65	210,16	1,52
patricia large	193,20	510,74	1,64	207,28	1,51
patricia small	193,11	510,75	1,64	207,27	1,51
stringsearch large	193,29	512,79	1,64	208,04	1,52
stringsearch small	193,28	512,76	1,64	208,02	1,52
crc32 large	195,46	495,66	1,58	202,39	1,45
crc32 small	195,46	495,67	1,58	202,39	1,45

Tabela 4.5: Resultados do acpower para programas do Mediabench e Mibench.

Capítulo 5

Conclusões e Trabalhos Futuros

Este trabalho apresentou o ***acpower***, uma ferramenta de estimativa de consumo de energia em nível de instrução, adicional à saída estatística do ArchC, que utiliza-se de técnicas de estimativa de consumo de energia em um nível mais alto de abstração do que as ferramentas mais conhecidas no mercado.

Nessa dissertação foram apresentados os resultados finais do projeto, obtidos com o desenvolvimento do ***acpower***, uma ferramenta para estimativa de consumo de energia em nível de instruções para processadores modelados em ArchC e comparações entre os tempos de execução dos métodos tradicionais e os da ferramenta desenvolvida, juntamente com seu erro.

5.1 Contribuições

Uma das contribuições desse trabalho é o ***acpower***, uma ferramenta simples, porém eficiente para re-estimar o consumo de energia de um determinado programa sem ter o trabalho de re-simulação, economizando tempo útil a algum projeto. Além de ser mais rápido, apresenta um erro consideravelmente baixo em relação ao método convencional.

A utilização do ***acpower*** só é possível graças ao advento de linguagens de descrição de arquiteturas, as ADLs, que, ao elevarem o nível de abstração, diminuem o tempo de simulação, em relação às linguagens de descrição de hardware, as HDLs.

Outra contribuição desse trabalho é que a ferramenta de estimativa de consumo de

energia desenvolvida não onera o tempo de simulação do ArchC.

5.2 Trabalhos Futuros

Ao final deste trabalho sobram algumas pendências como por exemplo, levar a ferramenta de estimativa de consumo de energia para dentro do ArchC, para que a primeira simulação não seja infrutífera. Um outro trabalho futuro seria automatizar a saída estatística do ArchC para que essas não se percam por descuido ou esquecimento do usuário.

Por enquanto, o maior trabalho subseqüente seria construir mais tabelas de custo base para os demais processadores descritos em ArchC, como o MIPS e o PowerPC, entre outros. Essas tabelas podem ser construídas de outras formas como, por exemplo, a utilização de datasheets ou a utilização de resultados já obtidos e disponíveis em artigos da área.

Referências Bibliográficas

[1] ArchC 2.0. *http://www.archc.org*, acessado em 04/2007.

[2] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen. Optimizing power using transformations. In *IEEE Transactions on CAD, Vol. 14, No. 1*, pages 12–31, 1995.

[3] Sente Corp. *WattWatcher Product Sheet*. Chelmsford, MA.

[4] Mentor Graphics Corporation. *ModelSim SE 6.1b Quick Reference Guide*, 2005.

[5] Vijay Degalahal and Tim Tuan. Methodology for high level estimation of fpga power consumption. In *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 657–660, New York, NY, USA, 2005. ACM Press.

[6] encc. *http://ls12-www.cs.uni-dortmund.de/research/encc/*, acessado em 10/2005.

[7] Jerry Frenkil. Tools and methodologies for low power design. In *DAC '97: Proceedings of the 34th annual conference on Design automation*, pages 76–81, New York, NY, USA, 1997. ACM Press.

[8] Gaisler. *http://www.gaisler.com/*, acessado em 06/2007.

[9] Synopsys Inc. *PowerCompiler User Guide, X-2005.09-SP4 edition*, 2004.

[10] K. Keutzer, O. Coudert, and R. Haddad. What is the state of the art in commercial eda tools for low power? In *ISLPED '96: Proceedings of the 1996 international symposium on Low power electronics and design*, pages 181–187, Piscataway, NJ, USA, 1996. IEEE Press.

[11] B. Klass, D. E. Thomas, H. Smith, and D. F. Nagle. Modeling inter-instruction energy effects in a digital signal processor, 1998.

[12] Paul Landman. High-level power estimation. In *ISLPED '96: Proceedings of the 1996 international symposium on Low power electronics and design*, pages 29–35, Piscataway, NJ, USA, 1996. IEEE Press.

[13] Sheayun Lee, Andreas Ermedahl, and Sang Lyul Min. An accurate instruction-level energy consumption model for embedded risc processors. *SIGPLAN Not.*, 36(8):1–10, 2001.

[14] Leon2. *http://www.gaisler.com/leonmain.html*, acessado em 04/2006.

[15] Leon3. *http://www.gaisler.com/leonmain.html*, acessado em 04/2006.

[16] D. Liu and C. Svensson. Power consumption estimation in cmos vlsi chips. In *IEEE Journal of Solic- State Ciruicts*, pages 663–670, 1994.

[17] R. Mehra and J. Rabaey. Behavioral level power estimation and exploration. In *Proceedings of the International Workshop on Low-Power Design*, pages 197–202, 1994.

[18] Huzefa Mehta, Robert Michael Owens, and Mary Jane Irwin. Energy characterization based on clustering. In *DAC '96: Proceedings of the 33rd annual conference on Design automation*, pages 702–707, New York, NY, USA, 1996. ACM Press.

[19] K. Muller-Glaser, K. Kirsch, and K. Neusinger. Estimating essential design characteristics to support project planning for asic design management. In *ICCAD-91 Digest of Technical Papers*, pages 148–151. ACM Press, 1991.

[20] Farid N. Najm. Towards a high-level power estimation capability. In *ISLPED '95: Proceedings of the 1995 international symposium on Low power design*, pages 87–92, New York, NY, USA, 1995. ACM Press.

[21] S. Nikolidis and Th. Laopoulos. Instruction-level power consumption estimation od embedded processors for low-power applications, 1999.

[22] Orinoco. *http://www.chipvision.com/*, acessado em 10/2005.

[23] S. Powell and E. Chau. Estimating power dissipation of vlsi signal processing chips: The pfa technique. In *VLSI Signal Processing IV*, pages 250–259, 1990.

[24] PowerScope. *http://www.cs.cmu.edu/˜ jflinn/pscope.html*, acessado em 10/2005.

[25] Sandro Rigo, Guido Araujo, Marcus Bartholomeu, and Rodolfo Azevedo. Archc: A systemc-based architecture description language. In *SBAC-PAD '04: Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04)*, pages 66–73, Washington, DC, USA, 2004. IEEE Computer Society.

[26] J. Russell and M. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *ICCD '98: Proceedings of the International Conference on Computer Design*, page 328, Washington, DC, USA, 1998. IEEE Computer Society.

[27] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Trans. Very Large Scale Integr. Syst.*, 2(4):437–445, 1994.

[28] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. Instruction level power analysis and optimization of software. *J. VLSI Signal Process. Syst.*, 13(2-3):223–238, 1996.

[29] SPARC V8. *www.sparc.org/standards/V8.pdf*, acessado em 10/2005.

[30] Xilinx. *XPower Tutorial - FPGA Design*. v. 1.3. 07/2002.

[31] Xilinx. *ISE 8.0 WebPack*, 2005.

[32] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *DAC '00: Proceedings of the 37th conference on Design automation*, pages 340–345, New York, NY, USA, 2000. ACM Press.