PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA EXPLORAÇÃO DO ESPAÇO DE PROJETO DE PROCESSADORES COM HARDWARE DE PONTO FLUTUANTE EM FPGAS

TACIANO ARES RODOLFO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Ney Laert Vilar Calazans

Porto Alegre, Brasil
2010

R695e Rodolfo, Taciano Ares.

Uma exploração do espaço de projeto de processadores com hardware de ponto flutuante em FPGAS / Taciano Ares Rodolfo. – Porto Alegre, 2010.

109 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Ney Laert Vilar Calazans.

1. Informática. 2. Arquitetura de Computador. 3. FPGA.

I. Calazans, Ney Laert Vilar. II. Título.

CDD 004.22

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Uma Exploração do Espaço de Projeto de Processadores com Hardware de Ponto Flutuante em FPGAs", apresentada por Taciano Ares Rodolfo, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas Embarcados e Sistemas Digitais, aprovada em 15/03/10 pela Comissão Examinadora:

Prof. Dr. Ney Laert Vilar Calazans – Orientador — PPGCC/PUCRS

Prof. Dr. Fernando Gehm Moraes – PPGCC/PUCRS

Prof. Dr. Eduardo Augusto Bezerra – UFSC

Prof. Dr. Luigi Carro – UFRGS

Homologada em...08/.06../..2012, conforme Ata No. ..012... pela Comissão Coordenadora.

p/

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

“*Conhecerás a verdade e a verdade vos libertará.*”

Jesus Cristo

AGRADECIMENTOS

Agradeço primeiro a Deus, este ser superior que buscamos compreender dentro das nossas parcas possibilidades, por nos ter concedido mais esta oportunidade para aprender, crescer e superar limites.

Gostaria muito de agradecer a todos da minha família. Aos meus pais, Moacir e Sandra, pelo amor incondicional e pelo suporte, financeiro e psicológico, e aos meus irmãos, Michel e Lucas, pelo apoio que me deram durante o período do mestrado.

Quero agradecer especialmente a minha esposa Carina por ter compreendido e suportado tudo o que passou decorrente daquilo que não pude dar durante toda a gravidez e durante todo o primeiro ano de vida do nosso amado filho Augusto: a minha presença! A ela peço que aceite minhas sinceras desculpas por minha ausência. Tenho a certeza de que o futuro reserva muitos momentos de felicidades a nós três.

Ao meu avô Antônio, que sempre considerei como meu segundo pai, agradeço por sempre ter me incentivado aos estudos e a retidão de caráter. Também agradeço por seu auxílio sempre tão desapegado e solícito.

As minhas avós Izaura e Zaíra, que não se encontram mais entre nós, agradeço pelo amor recebi das duas desde o berço. Aquele amor de vó sempre tão sincero e puro, que não nos cobra nada em troca de tudo. Aqui registro minha homenagem em memória de ambas. Tenho a certeza de que onde quer que estejam torcem muito por mim, como sempre fizeram em suas vidas.

Agradeço muito a minha tia Terezinha, cujo auxílio permitiu que eu chegasse até aqui. Meu obrigado por ter me recebido em sua residência sem nada me exigir. Também gostaria de pedir desculpas pelo incômodo e transtornos que a minha presença trouxe para a sua vida.

Ao meu tio Clóvis agradeço por ter me recebido em Porto Alegre. Sempre tão interessado, atencioso e preocupado com a minha vida, que posso afirmar que me senti quase como se fosse um filho seu. Também agradeço a sua família pelos momentos alegres e leves que tornaram a minha solidão na capital menos áspera.

Ao professor Ney Calazans agradeço sobremaneira pela orientação e atenção. Além disso, gostaria de registrar aqui meu agradecimento por ter me recebido em seu grupo de pesquisa GAPH no ano de 2002 quando entrei na PUCRS. Este fato permitiu que eu somasse muitos conhecimentos que vieram posteriormente a me auxiliar nos desafios que encontrei durante o mestrado.

Ao professor Fernando Moraes que compartilhou conosco momentos de orientação e outros de auxílio, principalmente durante o período da graduação, o que eu acho justo registrar nestas linhas dada a quantidade e qualidade do auxílio prestado.

Ao CNPQ pelo financiamento do meu mestrado.

Aos colegas de mestrado, Alzemiro, Samuel e Luciano, que compartilharam comigo diversos momentos, desde os mais alegres de confraternização até os mais difíceis e estressantes do mestrado. Aos diversos colegas do GAPH: Guindani, Edson, Ewerson, Ost, Rafael e a tantos outros de quem tenho a certeza de que me perdoarão pelo esquecimento de lhes agradecer.

A todos novamente meu *muito obrigado*!

UMA EXPLORAÇÃO DO ESPAÇO DE PROJETO DE PROCESSADORES COM HARDWARE DE PONTO FLUTUANTE EM FPGAS

RESUMO

Circuitos aritméticos são parte fundamental de sistemas digitais, uma vez que cada porção de informação processada por estes deve ter sido codificada previamente sob a forma de números, e que a aritmética é a forma por excelência de proceder à manipulação sistemática de números. Existe uma grande quantidade de esquemas de codificação usados em sistemas digitais, mas três formas de representação se sobressaem por serem usadas na maioria maciça das situações: números sem sinal, números inteiros e a representação de ponto flutuante. Os dois primeiros são mais simples e mais universais, mas algumas aplicações exigem o recurso à faixa estendida de valores e à precisão incrementada de representações de ponto flutuante. Embora o uso de hardware de ponto flutuante em FPGAs tenha sido por muito tempo considerado inviável ou relegado ao uso apenas em dispositivos e plataformas de alto custo, esta não é mais a situação atual. Este trabalho descreve o processo de projeto, a implementação física e uma avaliação preliminar de unidades de processamento de ponto flutuante de precisão simples em hardware para uma arquitetura de processador MIPS. Exploram-se várias implementações completas que têm a forma de coprocessadores fortemente acoplados. Estes coprocessadores ocupam apenas 4% de um FPGA de tamanho médio, enquanto o processador em si ocupa 3% do mesmo dispositivo. O processo de exploração do espaço de soluções de projeto descrito aqui considera as figuras de mérito área, desempenho e potência e considera variações na escolha da ferramenta de síntese, do método de geração a unidade de ponto flutuante e questões arquiteturais tais como estratégias de uso de relógios. Os experimentos conduzidos mostram reduções de mais de 20 vezes na contagem do número de ciclos de relógio do processador, para módulos de aplicação típicos que usam ponto flutuante de forma intensiva, quando comparado com processamento de representações de ponto flutuante emulado em software.

Palavras Chave: hardware de ponto flutuante; FPGA; exploração do espaço de projeto; projeto GALS; prototipação; processador embarcado

AN EXPLORATION OF THE DESIGN SPACE OF PROCESSORS WITH FLOATING POINT HARDWARE IN FPGAS

ABSTRACT

Arithmetic circuits are a fundamental part of digital systems, since every piece of information processed by them must first be encoded as numbers, and arithmetic is the ultimate way to systematically manipulate numbers. There exists a large number of available number encoding schemes, but three of these stand as useful in most situations: unsigned, integer and floating point. The first two are simpler and more universal, but some applications do require the recourse to the extended range of values, and the increased precision of floating point representations. Although the use of floating point hardware in FPGAs has long been considered unfeasible or relegated to use only in expensive devices and platforms, this is no longer the case. This work describes the design process, the implementation and a preliminary evaluation of single-precision floating point hardware units for an instance of the MIPS processor architecture. It explores several fully-fledged implementations that have the form of strongly coupled coprocessors. These coprocessors take as little room as 4% of a medium-sized FPGA, while the processor CPU may take only 3% of the same device. The space exploration process described here values area, performance and power metrics and considers variations on the choice of synthesis tool, floating point unit generation method and architectural issues such as clocking schemes. The conducted experiments show reductions of more than 20 times in clock cycles count for typical floating point application modules, when compared to the use of software-emulated floating point processing.

Keywords: floating point hardware; FPGA; design space exploration; GALS design; prototyping; embedded processor

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ANSI	**American National Standards Institute**
ASIC	***Application Specific Integrated Circuit***
bc1f	**branch if condition 1 false**
bc1t	**branch if condition 1 true**
BUFGCE	**Global Clock Buffer with Clock Enable**
CAD	**Computer-Aided Design**
CI	**Circuito Integrado**
CISC	**Complex Instruction Set Computer**
CMOS	**Complementary Metal-Oxide-Semiconductor**
CP0	**Coprocessor 0**
CP1	**Coprocessor 1**
CP2	**Coprocessor 2**
CP3	**Coprocessor 3**
CPU	**Central Processing Unit**
DCM	**Digital Clock Manager**
DFS	**Dynamic Frequency Scaling**
DLL	**Delay Locked Loop**
DDR	**Double Data Rate**
DSP	**Digital Signal Processor**
FPGA	**Field Programmable Gate Array**
FPU	**Floating Point Unit**
FIFO	**First In First Out**
FIR	**Finite Impulse Response**
FPU	**Floating Point Unit**
GALS	**Globally Asynchronous Locally Synchronous**
GAPH	**Grupo de Apoio ao Projeto de Hardware**
GCC	**GNU Compiler Collection**
GNU	**GNU is Not Unix**
GPS	**Global Positioning System**
HDR	**High Dynamic Range**
HFP	**Hardware Floating Point**
HW	**Hardware**
IEEE	**Institute of Electrical and Electronics Engineers**

IIR	**Infinite Impulse Response**
JTAG	***Joint Test Action Group***
LP	**Low Power**
LVDS	***Low Voltage Differential Signaling***
lw	***load word***
lwc1	***load word coprocessor 1***
MAC	**Media Access Control**
MARS	**MIPS Assembler and Runtime Simulator**
MIPS	**Microprocessor without Interlocked Pipeline Stages**
mfc1	**move from coprocessor 1**
mtc1	**move to coprocessor 1**
NaN	**Not a Number**
PDA	**Personal Digital Assistant**
PLL	**Phase Locked Loop**
QNaN	**Quiet Not a Number**
RAM	**Random Access Memory**
RISC	**Reduced Instruction Set Computer**
SDF	**Standard Delay Format**
SDRAM	**Synchronous Dynamic Random Access Memory**
SNaN	**Signaling Not a Number**
sw	**store word**
SW	**Software**
swc1	**store word coprocessor 1**
TSV	***Tabulated* Separated Value**
VHDL	**VHSIC Hardware Description Language**
VHSIC	**Very High Speed Integrated Circuits**
VLSI	**Very Large Scale Integration**

SUMÁRIO

1. INTRODUÇÃO

A crescente demanda por aparatos eletrônicos diminutos, eficientes e com múltiplas funcionalidades motiva pesquisas em busca de novas metodologias de desenvolvimento de sistemas embarcados. Sistemas embarcados geralmente realizam tarefas pré-definidas com requisitos bastante específicos. Desempenho mínimo, máxima área em silício, máximo consumo de energia, confiabilidade mínima e, principalmente, custo máximo, são restrições que devem ser levadas em consideração na pesquisa e no projeto de sistemas embarcados. Tais requisitos são baseados fortemente na aplicação alvo de um sistema, que pode envolver características tais como computação em tempo real, processamento de imagens e de áudio, processamento digital de sinais, entre outros.

Muitas aplicações, estimuladas pelo aumento na demanda por novas funcionalidades e aperfeiçoamento de outras, exigem cada vez mais o processamento de ponto flutuante. O atual estado da arte de dispositivos VLSI tornou o custo da área em silício mais acessível. Por esta razão, coprocessadores de ponto flutuante vêm sendo adicionados ao hardware de processadores embarcados, aumentando o seu poder computacional. Isto vem acontecendo até mesmo em processadores para aparelhos eletrônicos móveis, que são alimentados por fontes de energia bastante limitadas. Certamente neste caso, o máximo consumo de energia deve ser uma das principais restrições do sistema embarcado.

Com o avanço dos dispositivos VLSI, os FPGAs tornaram-se uma opção atraente para o projeto de sistemas digitais. Até pouco tempo atrás, estes dispositivos eram deixados em segundo plano, devido à baixa disponibilidade de área e ao seu baixo desempenho. Hoje em dia, sua flexibilidade e a atual oferta de módulos de hardware, disponibilizados sob a forma de núcleos sintetizáveis, diminuem o tempo de projeto e permitem alcançar rapidamente os requisitos de um sistema embarcado. Processadores podem ser conectados a dezenas de outros módulos de hardware, inclusive coprocessadores de ponto flutuante. Este trabalho tira proveito deste fato e procura explorar extensamente o espaço de projeto de um processador embarcado capaz de executar aplicações que fazem uso de números em aritmética de ponto flutuante.

1.1. Colocação do Problema

Aparatos eletrônicos agregam cada vez mais capacidade de processar aplicações complexas, tais como as aplicações multimídia [JEO99, DSP09, RTC09]. Correspondendo a amplo segmento das aplicações embarcadas, aplicações multimídia utilizam recursos que podem exigir o emprego de aritmética computacional que suporte uma ampla faixa de representação dos números e que garanta, ao mesmo tempo, a precisão de cálculos numéricos e o desempenho global do sistema embarcado. Nestes termos, entre as alternativas de aritmética existentes, a aritmética de ponto flutuante apresenta-se como a mais indicada para dar suporte satisfatoriamente a grupos específicos de aplicações.

No passado, a maioria maciça de sistemas embarcados dotados de processadores programáveis utilizava processadores comerciais que não possuíam hardware dedicado para executar operações de aritmética de ponto flutuante. Isto se deve principalmente ao acréscimo na área em silício do processador que tal unidade traria, onerando o seu custo final, o que sempre tem impacto altíssimo em sistemas embarcados. Além disso, este acréscimo também pode acarretar um potencial aumento na potência dissipada e/ou no consumo de energia, restringindo sua aplicabilidade. Desta forma, outras soluções eram empregadas tais como a emulação em software das instruções de ponto flutuante e a aritmética de ponto fixo. Contudo, estas soluções apresentam limitações que acabam por restringem o seu uso em diversos sistemas [INA96].

1.2. Motivação

A utilização de unidades de ponto flutuante em hardware no projeto de dispositivos móveis depende fortemente dos requisitos da aplicação [INA96, PAP04, TEX08], tais como o *deadline* das operações de tempo real, precisão nos cálculos realizados, desempenho, etc. Como comentado anteriormente, a utilização destas unidades aumenta a área em silício de um processador, o que por sua vez pode aumentar, de forma significativa, a dissipação de potência e/ou consumo de energia do mesmo.

O acréscimo na ocupação da área em silício é hoje um problema menor, dada a evolução da chamada lei de Moore [SAL06]. Esta dita que a cada dezoito a vinte e quatro meses duplica-se a densidade de dispositivos VLSI, tornando o custo relativo da área em silício cada vez mais baixo. Por outro lado, a dissipação de potência e o consumo de energia são cada vez mais aspectos críticos de projeto.

Diversas pesquisas recentes exploram a eficiência energética e a eficiência térmica [MOY01, VEN05]. Estudos de eficiência energética permitem minimizar o consumo de energia e prolongar o tempo de funcionamento dos sistemas embarcados. Estudos de eficiência térmica garantem o funcionamento correto do sistema sem comportamentos espúrios provocados pela eventual geração de calor em excesso no circuito integrado devido à dissipação de potência.

Pode-se afirmar que o uso de unidades de ponto flutuante em processadores embarcados é cada dia mais viável. Certamente há implicações a serem avaliadas antes de se optar pelo uso de tais unidades. Apesar de apresentar vantagens, sua dissipação de potência e seu consumo de energia, se não tratados adequadamente, podem restringir o uso de tais unidades. Este fato tem sido a motivação de diversos estudos, este trabalho não constituindo exceção.

1.3. Objetivos

Partindo da motivação, este trabalho possui os seguintes objetivos estratégicos:

- dominar o uso de processadores embarcados e a tecnologia de uso de unidades de ponto flutuante;
- dominar técnicas de projeto de sistemas digitais que visem a redução da dissipação de potência e do consumo de energia em processadores embarcados;
- explorar o espaço de projeto de um processador embarcado com relação à unidade de ponto flutuante.

Para alcançar os objetivos estratégicos, derivam-se os seguintes objetivos específicos:

- propor e implementar coprocessadores de ponto flutuante compatíveis com o conjunto de instruções MIPS-I de processadores embarcados;
- implementar organizações “processador/coprocessador de ponto flutuante” em hardware, incluindo versões síncrona e não-síncrona e explorar a dissipação de potência e consumo de energia destas organizações;
- disponibilizar um fluxo de projeto e validação para o desenvolvimento sistemas *GALS* dotados de pelo menos uma técnica de redução do consumo de energia e dissipação de potência em FPGAs;
- simular, prototipar e validar em FPGAs as diversas organizações “processador/coprocessador de ponto flutuante” implementadas;
- avaliar métricas de ocupação de área, estimativa da frequência de operação destas organizações, potência dissipada e energia consumida;

- desenvolver aplicações que fazem uso de aritmética de ponto flutuante para demonstrar o uso dos sistemas desenvolvidos;
- avaliar, através da execução das aplicações desenvolvidas, o desempenho, a dissipação de potência e o consumo de energia das diversas organizações implementadas.

Note-se que este trabalho não tem como objetivo o projeto de unidades de processamento de números de ponto flutuante em hardware. Utilizam-se implementações destas unidades disponíveis no domínio público e/ou geradores automáticos de hardware.

1.4. Contribuições

As contribuições deste trabalho compreendem:

- a implementação de diversos coprocessadores de ponto flutuante compatíveis com a família MIPS-I, fazendo reuso de unidades de ponto flutuante de código aberto e de unidades implementadas com o auxílio de uma ferramenta de geração de hardware;
- a implementação de organizações “processador/coprocessador de ponto flutuante”, incluindo versões não-síncronas que possibilitam o emprego de frequências de operação mais altas que as empregadas nas organizações síncronas;
- a implementação de uma organização “processador/coprocessador de ponto flutuante” não-síncrona dotada de uma técnica que visa a redução da dissipação de potência e do consumo de energia;
- a disponibilização de um fluxo de trabalho e de ferramentas, que permitem o cálculo da potência dissipada por um FPGA e do consumo de energia do mesmo;
- a avaliação do impacto da adoção de um coprocessador de ponto flutuante em um processador embarcado prototipado em FPGA (análise do desempenho, ocupação de área, dissipação de potência e consumo de energia adicionais).

1.5. Estrutura do Restante do Trabalho

O restante deste documento está estruturado da seguinte forma. O Capítulo 2 apresenta alguns conceitos fundamentais, incluindo o padrão IEEE-754, que normatiza o uso da aritmética de ponto flutuante, a arquitetura MIPS-I, base do processador empregado neste trabalho, e conceitos básicos sobre dissipação de potência e consumo de energia. O Capítulo 3 descreve o estado da arte no uso de unidades de ponto flutuante em processadores embarcados, bem como um breve estado da arte em técnicas para a redução do consumo de energia e dissipação de potência. Além disso, apresentam-se os estados da arte em técnica GALS de projeto e de interfaces de comunicação assíncrona. O Capítulo 4 apresenta os materiais e métodos utilizados na pesquisa proposta. Apresentam-se o processador e as unidades de ponto flutuante empregadas neste trabalho. O Capítulo aborda ainda os métodos e recursos usados para medição da dissipação de potência e consumo de energia. O Capítulo 5 apresenta a implementação dos coprocessadores propostos. A implementação das organizações oriundas da integração destes coprocessadores ao processador adotado é assunto do Capítulo 6. Neste Capítulo também se apresentam as implementações das organizações não-síncronas. As simulações realizadas, a prototipação destas em FPGAs, bem como os métodos utilizados para validação de todas as organizações implementadas, são o tema do Capítulo 7. O Capítulo 0 apresenta os resultados experimentais, incluindo estimativas de ocupação de área e da frequência de operação, métricas de desempenho e medidas de dissipação de potência e de consumo de energia. Por fim, o Capítulo 9 conclui esta Dissertação, tecendo considerações finais sobre os resultados obtidos e dando direções para trabalhos futuros.

2. CONCEITOS BÁSICOS

Este Capítulo apresenta alguns conceitos básicos necessários ao desenvolvimento do presente trabalho. A Seção 2.1 apresenta o padrão IEEE-754, a norma praticamente universal para a representação de números em ponto flutuante. A Seção 2.2 apresenta, em termos gerais, as principais características da arquitetura MIPS-I de processadores programáveis. Por fim, a Seção 2.3 apresenta alguns conceitos fundamentais sobre dissipação de potência e consumo de energia que valem para dispositivos CMOS.

2.1. O Padrão IEEE-754

O padrão IEEE-754 [AME85] foi criado pela ANSI/IEEE e tem como objetivo normatizar a representação binária de números racionais. Tal representação é conhecida como *ponto flutuante*. Neste padrão definem-se regras a seguir para a implementação de hardware e software, incluindo formatos de representação, operações aritméticas e de conversão, modos de arredondamento situações de exceção e seus respectivos tratamentos.

Existem dois formatos principais de representação de números de ponto flutuante com estrutura fixa: o formato de precisão simples, de 32 bits e o formato de precisão dupla, de 64 bits. Existem também extensões para estes dois formatos (os formatos variáveis) que não serão abordados no presente trabalho. A Figura 1 ilustra os dois formatos fixos da norma 754.

1	8	23
s	e	m

Precisão simples – 32 bits

1	11	52
s	e	m

Precisão dupla – 64 bits

Figura 1 – Formatos fixos de precisão simples e dupla.

Os dois formatos possuem os mesmos campos que, porém apresentam tamanhos distintos. O campo “s” é utilizado como bit de sinal do número como um todo, sendo ‘0’ para representar um número positivo e ‘1’ para um número negativo. O campo “e” é o expoente e “m” é a mantissa, ou parte fracionária, do número racional a ser representado. Vale ressaltar que valor do expoente é um número representado em notação polarizada, uma notação que representa uma faixa de inteiros entorno de 0, positivos e negativos. Portanto, não é necessário armazenar o sinal do mesmo. A notação polarizada facilita operações de comparação entre números, incluindo comparações de magnitude e de igualdade. A notação polarizada também é conhecida como notação por excesso de valor. Nesta notação, o menor valor que o expoente pode assumir é representado pelo vetor “e” contendo apenas zeros, correspondendo ao valor mais negativo possível na notação polarizada. Em precisão simples os limites dos valores do expoente são dados subtraindo 127 do binário puro associado ao vetor de bits “e” em precisão dupla subtrai-se 1023. Também conforme a norma, a mantissa tipicamente representa a parte fracionária de um número normalizado. Trata-se de um número cujo valor está no intervalo de racionais [1,2) (fechado à esquerda e aberto à direita) e seu bit mais significativo não é representado, sendo sempre ‘1’. Este bit também é conhecido como *hidden bit*. A normalização garante um máximo de precisão nas operações aritméticas.

A partir dos campos “s”, “e” e “m” dos formatos, é possível derivar a Equação 1 que representa um número real normalizado em ponto flutuante. A variável *PESO* utilizada nesta equação é a polarização e tem valor 127 na representação de precisão simples e 1023 em precisão dupla.

$$num = (-1)^S \cdot (1 \cdot m) \cdot 2^{e-PESO} \tag{1}$$

A Tabela 1 apresenta cinco casos de valores que podem ser representados no padrão pelos formatos de precisão simples e dupla.

Tabela 1 – Valores representados nos formatos de precisão simples e dupla.

	Precisão Simples			**Precisão Dupla**		
	Expoente 8 bits	**Mantissa 23**	**Valor**	**Expoente 11 bits**	**Mantissa 52 bits**	**Valor**
Not-a-Number	e = 255	m ≠ 0	NaN	e = 2047	m ≠ 0	NaN
Infinito	e = 255	m = 0	$(-1)^s = \infty$	e = 2047	m = 0	$(-1)^s = \infty$
Racional	0<e<255	m	$(-1)^s \cdot(1.m)\cdot 2^{e-127}$	0<e<2047	m	$(-1)^s \cdot(1.m)\cdot 2^{e-1023}$
Não normalizado	e = 0	m ≠ 0	$(-1)^s \cdot(0.m)\cdot 2^{-126}$	e = 0	m ≠ 0	$(-1)^s \cdot(0.m)\cdot 2^{-1022}$
Zero	e = 0	m = 0	$(-1)^s \cdot(0)$	e = 0	m = 0	$(-1)^s \cdot(0)$

Além de valores do tipo “Racional”, utilizado para a representação de números racionais, o padrão permite a representação de valores especiais. “Not-a-Number”, detalhado mais adiante, é utilizado para indicar resultados inválidos. É utilizado também no tratamento de exceções. “Infinito” é utilizado nos resultados de operações que ultrapassam os limites superiores definidos pelo padrão. Números representados como “Não normalizado” não empregam o *hidden bit*. Deste modo, é possível representar valores menores do que os limites inferiores definidos pelo padrão. “*Zero*” é utilizado para representar o valor 0, que este não pode ser normalizado. Todos os bits de sua mantissa e expoente são preenchidos com ‘0’. A Tabela 2 apresenta os diferentes formatos normatizados pelo padrão e os valores para os parâmetros que os definem.

Tabela 2 – Diferentes formatos e valores para os parâmetros que os definem.

	Precisão simples	**Precisão simples estendida**	**Precisão dupla**	**Precisão dupla estendida**
Bits de precisão	24	≥ 32	53	≥ 64
e max	127	≥ 1023	1023	≥ 16383
e min	-126	≤ -1022	-1022	≤ -16382
Polarização (*PESO*)	127	Depende	1023	Depende
Total de bits	32 exatamente	Variável, ≥ 43, < 64	64 exatamente	Variável, ≥ 79

A norma IEEE-754 provê operações aritméticas (adição, subtração, multiplicação, divisão e raiz quadrada) além de operações de conversão (de ponto flutuante para inteiro e vice-versa, entre formatos de representação) e de comparação (igualdade, maior e menor que, etc.).

Como resultados de operações aritméticas realizadas em equipamentos reais não podem possuir uma precisão infinita, é sempre necessário truncar, ou arredondar, tais resultados. A norma define quatro modos de arredondamento: para zero, onde se arredonda para o valor imediatamente anterior, ou menor, em casos de resultados positivos e para o valor imediatamente posterior, ou maior, em resultados negativos; para cima e para baixo onde se arredonda o resultado para o maior e menor valor relativo respectivamente; e para o mais próximo que arredonda para o valor imediatamente acima ou abaixo dependendo da proximidade com o resultado. Este último é o modo recomendado pela norma.

São previstas no padrão cinco exceções: “operação inválida”, “divisão por zero”, “*overflow*”, “*underflow*” e “valores inexatos”. Todas estas exceções dever ser sinalizadas quando detectadas e medidas devem ser adotadas para os seus respectivos tratamentos, como sinalização em qualificadores, disparo de interrupções, ou ambos.

A exceção “operação inválida” aplica-se a operações aritméticas para as quais não existe correto possível, tais como divisão por zero e raiz quadrada de um número negativo. O resultado de tais operações é definido como NaN, ou *Not-a-Number*. Existem dois tipos de NaN: QNaN e SNaN, *quiet* e *signaling* respectivamente. SNaN indica operação inválida somente se um dos operandos for SNaN. SNaN é um tipo especial de operando, como números complexos por exemplo. As demais operações inválidas devem sempre ser sinalizadas QNaN. A exceção “divisão por zero” ocorre quando um número não-zero e finito é dividido pelo número zero, resultando em um valor não representável. A exceção “*overflow*” é sinalizada sempre que o resultado de uma operação excede o valor máximo que pode ser representado pelo expoente. A exceção "*underflow*" ocorre quando o valor do resultado é tão pequeno que não pode ser representado de forma normalizada a não ser como o número 0, ainda que este não seja o resultado correto da operação. Isto causa perda de precisão quando o resultado vem a ser utilizado em outra operação aritmética. A exceção "valores inexatos" ocorre nos resultados das operações devido aos arredondamentos e às restrições do expoente e de precisão.

2.2. Organização MIPS-I

O projeto MIPS [STA98] começou a ser desenvolvido em meados de 1981 na Universidade de Stanford por uma equipe de doutorandos e de mestrandos chefiada por John Hennessy. Em 1984, convencido do potencial do processador, Hennessy licenciou-se da universidade durante um ano para tornar-se um dos fundadores da MIPS Computer Systems, empresa responsável pelo desenvolvimento dos primeiros microprocessadores comerciais MIPS. O primeiro MIPS comercial, o R2000, foi lançado em 1985. Mais tarde, no ano de 1992, a MIPS Computer Systems foi adquirida pela Silicon Graphics Computer Systems.

No inicio da década de 90 o MIPS começou a ser licenciado para terceiros. O licenciamento se mostrou próspero devido à simplicidade destes processadores. Sendo mais eficiente a um custo mais acessível, passou a ser utilizado em diversos dispositivos que até então utilizavam processadores de arquitetura CISC. Empresas como a Silicon Graphics, NEC e Acer, adotaram os processadores MIPS em seus desktops. Porém, devido à competição de processadores Pentium pela Intel e devido à falta de suporte ao MIPS por parte de Microsoft em seus sistemas operacionais, os processadores MIPS caíram em desuso em computadores. Já em dispositivos embarcados a situação é outra. Com o licenciamento, o MIPS tornou-se amplamente empregado no mercado de sistemas embarcados. Versões da sua arquitetura têm sido usadas em diversos dispositivos, tais como: impressoras, televisores, consoles de jogos, PDAs, set-top boxes, entre outros. Com o sucesso do MIPS, recentemente várias gerações de processadores são oferecidas na forma de núcleos sintetizáveis para o projeto de soluções embarcadas. Devido ao conhecimento generalizado por parte de projetistas da arquitetura MIPS e devido ao amplo ferramental de desenvolvimento, o MIPS permanece ainda hoje como um dos processadores embarcados mais utilizados na indústria.

MIPS é acrônimo para *Microprocessor without Interlocked Pipeline Stages*, ou microprocessador sem estágios de *pipeline* bloqueados. Atualmente existem 5 arquiteturas MIPS, compatíveis entre si, denominadas MIPS-I, MIPS-II, MIPS-III, MIPS-IV, e MIPS-32/64. No contexto deste trabalho, aborda-se somente a arquitetura MIPS-I, a mais simples. Esta arquitetura costuma empregar uma organização de memória do tipo Harvard e seu *hardware* constitui-se de uma unidade de processamento e por funcionais, ou coprocessadores, conforme ilustra a Figura 2. Sua unidade de processamento é um processador RISC do tipo *load*/*store* com todas as suas instruções lógicas e aritméticas operando apenas sobre registradores do banco. Existem 32 registradores no banco e são designados com R0 a R31 ou $0 a $32. O registrador $0 não é propriamente um registrador, mas a constante 0. O registrador $31 é usado de forma especializada por uma instrução (jal) para armazenar implicitamente o endereço de retorno de uma subrotina. Tipicamente, organizações MIPS-I empregam um *pipeline* de 5 estágios: busca, decodificação, execução, acesso à memória e escrita no banco de registradores. Neste *pipeline*, se não houver conflitos entre instruções executando simultaneamente, a cada ciclo de relógio conclui-se a execução de uma delas.

A organização MIPS-I foi projetada para operar com até quatro coprocessadores. O coprocessador denominado CP0, de uso obrigatório, é utilizado para controle do sistema, realizando tarefas como o gerenciamento de memória, controle de interrupções e exceções, prever diagnósticos, etc. Geralmente este coprocessador é implementado juntamente com o processador. O coprocessador denominado CP1 é reservado para o processamento em hardware de números em ponto flutuante. Este coprocessador deve ser compatível com o padrão IEEE-754 e deve possuir seu próprio banco de registradores de 32 posições com uma largura de 32 bits, designados com $f0 a $f31. Os coprocessadores CP2 e CP3 podem ser utilizados na implementação de funcionalidades adicionais. Na Figura 2 pode-se observar a organização da família MIPS-I com seu processador, memória, e coprocessadores CP0 e CP1.

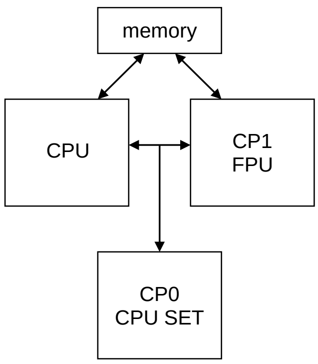

Figura 2 – Organização da família MIPS-I.

O MIPS-I possui no total 111 instruções onde o código objeto de cada uma destas ocupa exatamente 32 bits. São 21 instruções aritméticas, 8 instruções lógicas, 8 de manipulação de bits, 12 de comparação, 25 de controle de fluxo (saltos), 15 de leitura de memória, 10 de escrita na memória, 8 de movimentação de dados entre processador e coprocessador e 4 instruções miscelâneas. Existem três formatos de instrução: imediato, salto e registrador. A Figura 3 mostra os formatos e sua codificação. Os três formatos têm um campo em comum: o campo de *opcode*, utilizado para identificar a instrução no estágio de decodificação do processador. Os formatos imediato e registrador ainda têm em comum os campos *rs* e *rt*, utilizados para a especificação dos registradores fonte e destino.

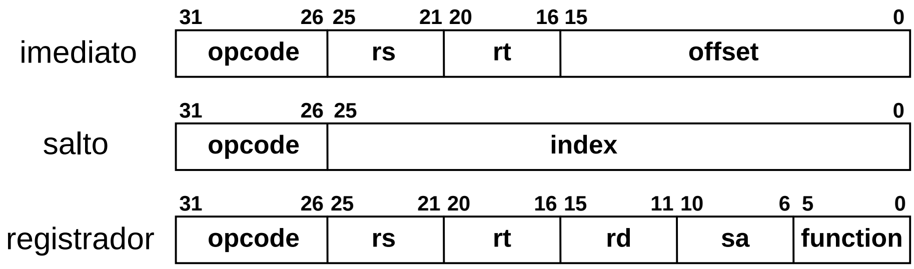

Figura 3 – Formatos básicos das instruções inteiras do MIPS-I.

Nas instruções com o formato imediato o campo *offset* pode conter valores entre 0 a 65535 para variáveis do tipo inteiro sem sinal ou valores entre -32768 a 32767 para inteiro com sinal. Nas instruções com o formato salto o campo *index* é utilizado como inteiro com sinal para as instruções de saltos relativos ou é utilizado, através de seus cinco bits menos significativos, para a seleção de um dos 32 registradores de propósito geral. O formato registrador é utilizado apenas em instruções lógico-aritméticas. O campo *rd* seleciona outro registrador destino. O campo *sa* determina o número de bits a deslocar nas operações de deslocamento. Finalmente, o campo *function* é utilizado para especificar a operação específica em instruções lógico-aritmeticas, uma vez que neste formato todas as instruções possuem *opcode* idêntico (000000 em binário).

2.3. Dissipação de Potência e Consumo de Energia

O projeto de sistemas digitais tradicionalmente contempla o desempenho, em particular de processadores. Com o crescente aumento da densidade dos dispositivos VLSI é possível desenvolver sistemas cada vez mais complexos que permitem atender a demanda por alto desempenho. Porém, altos níveis de desempenho de processadores podem trazer consigo altos consumo de energia e dissipação de potência. Tão altos, que a densidade da potência dissipada e concomitante geração de calor vêm rapidamente alcançando níveis comparáveis a reatores nucleares [VEN05]. Este fato acaba por diminuir a confiabilidade e a expectativa de vida dos circuitos integrados e também por aumentar os custos de arrefecimento.

Os altos níveis de dissipação de potência e de consumo de energia dos processadores têm sido determinantes no projeto de sistemas embarcados, principalmente em aparatos eletrônicos móveis, uma vez que estes utilizam fontes de energia limitadas, como baterias recarregáveis, por exemplo. Dissipação de potência tornou-se o terceiro eixo no espaço de otimização sendo adicionado às restrições de desempenho e ocupação de área em silício [VEN05]. Portanto, é necessário encontrar soluções de projeto para a minimização dos problemas ocasionados pela dissipação de potência.

Potência e energia são definidas em termos de trabalho que um sistema realiza. Energia é a quantidade total de trabalho que um sistema realiza ao longo de um determinado período de tempo, enquanto que potência é a taxa no qual o sistema realiza trabalho [VEN05].

$$P = W/T \tag{2}$$

$$E = P/T \tag{3}$$

As Equações 2 e 3 apresentam os termos formais para o cálculo da potência e da energia de um determinado sistema. Nestas equações: P é potência; E é energia; T é o intervalo de tempo especifico; e W o total de trabalho realizado neste intervalo. Energia é medida em Joules e potência em Watts. No contexto deste trabalho, potência é a taxa de consumo de energia elétrica de um circuito quando está em funcionamento. Energia é a energia elétrica total que o circuito consome ao longo de um determinado período de tempo. Apesar dos conceitos estarem intrinsecamente relacionados, a distinção entre potência e energia é importante, uma vez que técnicas que reduzem a potência dissipada não necessariamente reduzem a energia consumida, e vice-versa [VEN05]. Em aparatos eletrônicos móveis geralmente se empregam técnicas de redução do consumo de energia, uma vez que este expediente aumenta a duração da carga de suas baterias. Por exemplo, considere-se a tecnologia atual de baterias recarregáveis, como as Níquel-Metal Híbrido, disponíveis em tamanhos “AA” com capacidade de 1600 mA.h a uma tensão nominal de 1,2 volts. Muitos dispositivos móveis que utilizam um par destas baterias podem permanecer em funcionamento por aproximadamente quatro horas quando se dissiparem até 1 Watt de potência média. Para funcionarem durante um mês sem recargas, as técnicas de redução empregadas devem fazer esta dissipação cair para 5 mW [MOY01].

3. ESTADO DA ARTE

Este Capítulo apresenta revisões do estado da arte relacionadas à pesquisa proposta. A Seção 3.1 apresenta uma revisão sobre o espectro de possíveis soluções para sistemas que necessitam de processamento de ponto flutuante. Na Seção 3.2 apontam-se exemplos para demonstrar como é cada vez maior a utilização de unidades de ponto flutuante em dispositivos embarcados e/ou móveis. A Seção 3.3 apresenta alguns estudos que utilizam FPGAs para a implementação de soluções que empregam ponto flutuante em hardware. Em seguida, a Seção 3.4, apresenta técnicas que visam à redução da dissipação de potência e do consumo de energia em circuitos CMOS. Finalizando o Capítulo, a Seção 0 discute técnica GALS de projeto e interfaces assíncronas de comunicação que podem ser utilizadas nesta técnica.

3.1. Processamento de Ponto Flutuante

No passado, uma unidade de ponto flutuante em hardware representava um custo adicional considerável em relação à unidade de processamento bem como ao sistema como um todo, tornando sua utilização restrita. Porém, o atual estado da arte da fabricação de circuitos integrados aumentou consideravelmente o desempenho e a densidade de circuitos integrados (CIs), tornando relativamente muito menos dispendioso o seu uso. Portanto, pode-se considerar que em um futuro próximo, unidades de ponto flutuante estarão presentes em qualquer sistema que possa se beneficiar com o seu uso. Isto inclui diversas classes de sistemas embarcados, até mesmo aparatos eletrônicos móveis, se o problema da dissipação de potência e do consumo de energia forem adequadamente controlados. Para colocar o trabalho em perspectiva, a Figura 4 apresenta o espectro de opções de projeto para sistemas que possam se beneficiar do uso de representações de ponto flutuante.

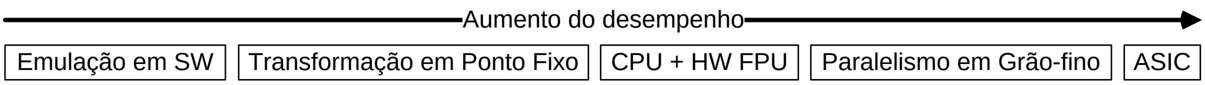

Figura 4 – Espectro de opções de projeto para soluções em ponto flutuante.

Cada opção possui benefícios e desvantagens e a escolha de uma determinada opção deve basear-se fortemente nos requisitos da aplicação alvo do sistema embarcado [INA96, PAP04, TEX08].

A emulação em software é a opção mais direta, uma vez que o compilador simplesmente transforma as operações de ponto flutuante em um conjunto equivalente de manipulações utilizando números inteiros. Esta é a opção mais empregada em processadores embarcados. Entretanto, o principal problema é seu baixo desempenho, que provém do código de máquina extenso e lento gerado pelo compilador.

Transformar uma especificação de ponto flutuante em ponto fixo pode resolver o problema do desempenho da emulação em software, ao custo de se reduzir a portabilidade. Em ponto fixo não existe um campo de expoente e a posição da vírgula (fixa) deve ser armazenada e gerenciada separadamente. Sua vantagem é a facilidade de implementação, bem como a eficiência das operações aritméticas, uma vez que seu formato, similar à representação de números inteiros, é bastante simples. Disto resulta menor área em silício e alto desempenho, já que o hardware para implementar operações aritméticas inteiras possui um porte bastante reduzido se comparado com outras opções. Ponto fixo é uma solução atraente para sistemas embarcados de baixa dissipação de potência, sendo utilizada em processadores de diversos fabricantes. Todavia, ponto fixo logo perde suas vantagens quando os números a serem manipulados possuem uma diferença muito grande em suas magnitudes. Quando isso ocorre são necessários grandes deslocamentos nas mantissas dos números e diversos incrementos ou decrementos na posição da vírgula (correspondente ao expoente em ponto flutuante), antes da execução de qualquer operação. Este fato quando ocorre, acaba degradando significativamente o desempenho global do sistema [INA96]. Além disso, apesar de apresentar uma precisão satisfatória, sua reduzida faixa de representação de números pode acabar restringindo o uso em determinadas aplicações.

A próxima opção, adotada na pesquisa proposta, é o uso de um processador dotado de uma unidade de ponto flutuante em hardware. Esta opção pode aumentar consideravelmente o desempenho de um sistema. Existem estudos que afirmam que esta opção oferece um ganho da ordem de 20 a 100 vezes em relação à emulação em software, dependendo da operação empregada [DSP08]. Outros estudos afirmam ainda que processadores embarcados comerciais executando aplicações, tais como a computação de um filtro FIR, conseguem um aumento de desempenho em aproximadamente duas ordens de grandeza [EET01]: 4,8 Mflops (milhões de operações em ponto flutuante por segundo) para a emulação em software e 380 Mflops para um processador dotado de uma unidade de ponto flutuante em hardware. Apesar do desempenho proporcionando por esta opção, a adoção de uma unidade de ponto flutuante pode acarretar um aumento significativo na ocupação de área do processador e também em sua dissipação de potência.

Problemas específicos, tais como radares e simulações de dinâmica molecular, podem se beneficiar de forma significativa utilizando FPGAs para a implementação de paralelismo em grão-fino. No entanto, esta opção pode levar a um grande tempo de projeto e a redução da flexibilidade e da portabilidade.

Por fim, os virtualmente infinitos graus de liberdade do projeto de Circuitos Integrado para Aplicações Específicas (do inglês, *Application Specific Integrated Circuits* ou ASICs). Nesta opção se obtém as maiores otimizações na área, desempenho e dissipação de potência. Contudo, poucas aplicações embarcadas estão habilitadas a enfrentar os custos de projeto e fabricação de máscaras oriundos do fluxo de projeto ASIC.

3.2. Utilização Comercial de Ponto Flutuante

Como comentado anteriormente, o atual estado da arte em fabricação de circuitos integrados tornou menos dispendioso o emprego de unidades de ponto flutuante. Por este motivo, é possível perceber o crescente uso de tais unidades em processadores embarcados comerciais.

Máquinas digitais de uso profissional estão empregando a tecnologia HDR (do inglês, *High Dynamic Range*) [MAN07], ou Grande Alcance Dinâmico, que utiliza ponto flutuante para representar precisamente os níveis de luminosidade de suas fotografias. Com esta técnica, por exemplo, é possível alcançar 100.000 níveis de luminosidade, ao contrário dos 200 níveis atuais [HDR09]. O software embarcado de câmeras digitais, utilizado para o tratamento de imagens que empregam a técnica HDR, exige o processamento maciço de aritmética de ponto flutuante.

Navegadores de GPS, incluindo aparelhos topográficos, estão empregando processadores embarcados que possuem unidades de ponto flutuante. Desta forma, podem realizar com rapidez e alta precisão os cálculos responsáveis, não somente pela navegação, mas também pela geração de mapas 3D [EET09, ZII09].

Codificadores e decodificadores de áudio foram desenvolvidos para utilizarem ponto fixo ou emulação de ponto flutuante e trabalham satisfatoriamente utilizando tais representações. Porém, ponto flutuante vem sendo cada vez mais considerado para áudio de alta fidelidade. Sua maior precisão evita a propagação de erros entre os diversos estágios de codificação e decodificação, e sua ampla faixa de representação evita distorções harmônicas oriundas da digitalização do sinal de áudio [DSP09, RTC09].

Consoles de jogos de ultima geração estão empregando unidades de ponto flutuante para os cálculos realizados nas diversas técnicas utilizadas para a geração da parte gráfica de jogos, tais como a renderização de polígonos, efeitos de *morphing*, mapeamento de texturas, efeitos de luz e sombra, entre outros [JEO99]. A console portátil PSP [EXT09] da Sony possui um circuito integrado denominado Allegrex. Sua organização inclui um processador baseado no MIPS-32 R4000, uma unidade de ponto flutuante, uma unidade de ponto flutuante vetorial, e dois processadores adicionais para as aplicações do tipo multimídia e DSP. A console Playstation 3 [PS309], também da Sony, possui em sua plataforma de hardware um acelerador gráfico NVIDIA modelo RSX, responsável por cálculos de ponto flutuante maciços, e um processador Cell da IBM. Este processador é constituído de um processador PowerPC e de 8 SPEs, ou elementos de processamento sinergético, que também podem efetuar cálculos de ponto flutuante. O desempenho resultante para esta console é de 218 Gflops [PS309].

Fabricantes de processadores específicos para notebooks e laptops, disponibilizam modelos que trazem em seu hardware unidades de ponto flutuante, tais como o Atom da Intel [CPU09a], Nano da Via [VIA09] e Semprom da AMD [CPU09b]. A ARM disponibiliza, para o projeto de dispositivos do tipo *handheld*, processadores com barramento de dados que permitem a utilização de seu coprocessador de ponto flutuante FPA10-A [HAR93].

3.3. Ponto Flutuante em FPGAs

No passado, FPGAs foram evitados no projeto de hardware de ponto flutuante devido à baixa disponibilidade de área e ao seu baixo desempenho. Atualmente, FPGAs estão sendo considerados mais do que uma plataforma de prototipação. Seus fabricantes têm disponibilizado modelos de baixo custo e baixo consumo de energia [XIL09a, ALT09] para a indústria de eletrônicos, que os tem utilizado cada vez mais em sistemas embarcados fabricados em pequenas quantidades. Além disso, diversas pesquisas acadêmicas e científicas estão empregando este tipo de dispositivo para o processamento maciço de cálculos em ponto flutuante. As propostas de [CHO09] e [BEA06] produzem casos para justificar a inclusão de unidades de ponto flutuantes embarcadas em FPGAs sob a forma de núcleos de hardware dedicados. No entanto, tais propostas parecem não ter tido espaço nos principais fabricantes de FPGAs. Quando os recursos de um FPGA eram mais escassos, alguns autores propuseram formatos de ponto flutuante configuráveis e bibliotecas de módulos de ponto flutuante. Em [DID02] os autores propõem um formato parametrizável, não compatível com o padrão IEEE-754, específico para o projeto DSP em FPGAs.

Em [BEL02] encontra-se a proposta de uma biblioteca de módulos parametrizáveis para a construção de operadores de ponto flutuante adaptados para a aplicação em questão. Outros trabalhos, tais como [JIE08], afirmam que um grande desempenho em computação de ponto flutuante pode ser obtido em FPGAs, projetando-se operadores básicos de ponto flutuante de forma otimizada. Tais trabalhos adotam o reuso de núcleos de hardware, abordagem igualmente usada neste trabalho. Finalmente, existem trabalhos que propõe a integração de processadores embarcados e unidades de ponto flutuantes em FPGAs. Esta proposta forma uma solução completa, tal como investigada na pesquisa proposta aqui. Em [PAP08] o autor integra instruções de ponto flutuante personalizadas ao processador *soft* Nios da Altera para a aceleração do processamento de MP3. Sua arquitetura é especifica para tal aplicação. Em [KAD07] aumenta-se o desempenho de um processador *soft* Microblaze com a integração de até oito coprocessadores de ponto flutuante, cada um controlado por um processador Picoblaze reconfigurável. A integração é baseada fortemente no uso do ambiente EDK. Além disso, a configuração dos controladores de coprocessadores exige codificação *em linguagem de montagem*. Outro trabalho [STE09] demonstra como acelerar computação de ponto flutuante a partir de uma unidade de ponto flutuante específica para um determinado FPGA acoplado ao processador embarcado PowerPC. Novamente, o ambiente EDK é utilizado, o que pode diminuir a flexibilidade da abordagem.

A pesquisa proposta no presente trabalho emprega um processador embarcado de código aberto, possibilitando a otimização da interface processador/coprocessador em baixo nível de abstração. O trabalho também explora o processador acoplado a várias unidades de ponto flutuante distintas, o que favorece o rápido projeto de blocos, dezenas dos quais podem ser prototipados em FPGAs.

3.4. Redução da Dissipação de Potência e do Consumo de Energia

A potência em circuitos digitais construídos com tecnologia CMOS pode ser dividida em três componentes principais [RAB96]: dinâmica, de curto-circuito e estática. A potência dinâmica é o resultado da carga e descarga das capacitâncias dos circuitos, sendo tradicionalmente a que mais contribui para o consumo total de um circuito integrado: em torno de 45% [AMD05]. A potência de curto-circuito está associada à transição dos níveis de tensão em uma porta lógica, quando ambos os transistores (N e P) conduzem simultaneamente por um breve período de tempo. Corresponde de 10% a 15% da potência dissipada [VEN05]. A potência estática é definida como aquela resultante da dissipação devido a corrente de fuga nos transistores e nas junções P-N (em inglês, *leakage current*). Portanto, existem três formas de atacar o problema da dissipação de potência em circuitos do tipo CMOS, uma para cada tipo de componente responsável por dissipação de potência. Os números citados aqui para o percentual de potência dissipada em cada componente devem ser colocados em perspectiva. Estes valores estão mudando drasticamente à medida que se avança em tecnologias profundamente submicrônicas e podem/devem inverter em breve.

Neste trabalho abordam-se somente as técnicas que reduzem a potência dinâmica de um circuito. As técnicas que visam a redução da potência de curto-circuito e da potência estática demandam soluções em nível de transistores e por este motivo não serão abordadas. A potência dinâmica é o resultado da carga e descarga das capacitâncias do circuito, e por esta razão a frequência com a qual tais capacitâncias são carregadas e descarregadas reflete diretamente na dissipação de potência. Portanto, a frequência de relógio aplicada no circuito, é um fator que contribui majoritariamente para a potência dinâmica. A potência dinâmica pode ser calculada aplicando-se a Equação 4 [RAB96].

$$P_{dinâmica} = \propto \cdot fc \cdot Cl \cdot Vcc^2 \tag{4}$$

Nesta equação, $\propto$ corresponde à probabilidade de chaveamento do total de portas lógicas do circuito, fc corresponde à frequência de relógio aplicada ao circuito, Cl corresponde à carga capacitiva total do circuito, e Vcc corresponde à tensão de alimentação aplicada ao circuito.

As técnicas que reduzem a potência dinâmica abordadas neste trabalho devem ser necessariamente prototipáveis em FPGAs, uma vez que se pressupõe o uso deste tipo de dispositivo na pesquisa proposta. Por este motivo, o presente trabalho aborda técnicas como *clock gating* e DFS.

Chaveamento de Relógio

A árvore de distribuição do sinal de relógio é responsável por mais de 40% da dissipação de potência de um circuito do tipo CMOS [DON03]. O chaveamento de relógio (do inglês, *clock gating*) é uma técnica bem conhecida e amplamente utilizada que desconecta o sinal de relógio de circuitos que se encontram inativos. Desta forma, diminui-se significativamente a atividade de chaveamento destes circuitos, reduzindo a dissipação de potência.

A Figura 5 ilustra duas formas de se implementar a técnica de chaveamento de relógio. A primeira (**a**) desconecta diretamente o sinal de relógio de sua árvore de distribuição. Esta implementação é utilizada principalmente em projetos de ASICs. A segunda implementação (**b**) utiliza multiplexadores para forçar o registrador a armazenar o seu próprio conteúdo. Desta maneira, a lógica combinacional permanece em seu estado atual, minimizando a atividade de chaveamento. Esta implementação é bastante utilizada em FPGAs, onde a árvore de distribuição do sinal de relógio é pré-definida. Nestes dispositivos, qualquer lógica adicional na rede de distribuição pode causar escorregamentos significativos no sinal de relógio e, como consequência, os tempos de *setup* e de *hold* dos registradores podem ser violados ou difíceis de controlar, acarretando riscos de mau funcionamento do circuito.

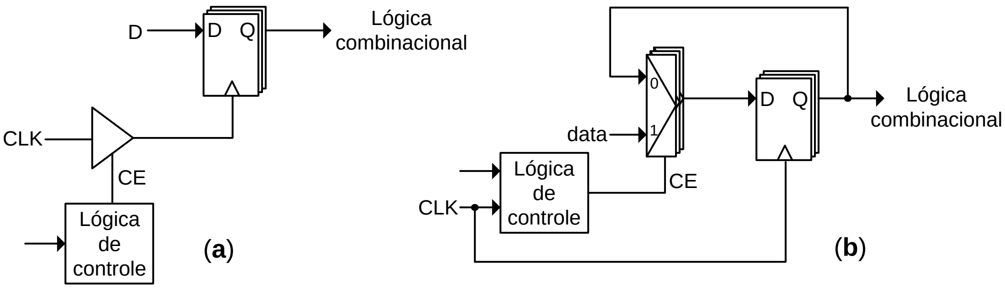

Figura 5 – Implementações da técnica de chaveamento de relógio [ZHA06].

A técnica de chaveamento de relógio pode ser aplicada em grão-pequeno ou em grão-grande, como se observa na Figura 6. Em grão-pequeno é possível obter um controle maior de unidades de hardware menores. Entretanto, existe um acréscimo de área proveniente da multiplicação de módulos necessários para o controle local do sinal de relógio. Além disso, dependendo do padrão da aplicação da técnica, as unidades de hardware podem não ser todas desativadas ao mesmo tempo. Por este motivo, a atividade de chaveamento pode ser maior em relação à técnica aplicada em grão-grande. Em grão-grande, obtém-se um controle menos detalhado, porém, com a vantagem de se desconectar o sinal de relógio de uma área de hardware maior, diminuindo de forma mais simplificada a atividade de chaveamento.

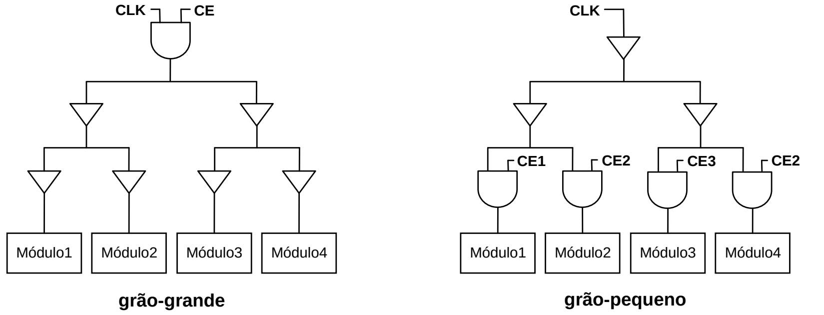

Figura 6 – Técnica de chaveamento de relógio em grão-grande e grão-pequeno [LUO05].

Variação Dinâmica da Frequência de Operação

A técnica de variação dinâmica da frequência de operação (do inglês, *dynamic frequency scaling*) tem como objetivo controlar a frequência do sinal de relógio de um sistema digital em tempo de execução. Desta forma, é possível adaptar (diminuir ou aumentar) sua frequência conforme os requisitos das aplicações que o sistema executa, considerando também as mudanças no ambiente onde se insere o sistema.

Aplicações de tempo real geralmente são de baixa latência, uma vez que seus cálculos precisam ser realizados adequadamente dentro do intervalo de um tempo disponível. Muitas vezes esta baixa latência é obtida aplicando-se uma alta frequência de operação no sistema digital. De forma contrária, aplicações que não exigem alto desempenho computacional permitem a diminuição da frequência de operação do sistema digital. Sabe-se que o aumento da frequência de operação pode acarretar aumento na dissipação de potência (ver Equação 4) e consequentemente aumentar o consumo de energia. A diminuição da frequência de operação, apesar de reduzir o consumo de energia, acarreta a degradação do desempenho das aplicações.

O desafio no projeto de sistemas embarcados dotados da técnica DFS tem sido a obtenção de um método de controle dinâmico que, dados os requisitos das aplicações e as mudanças em seu ambiente, defina a frequência de operação que permita o melhor compromisso possível entre a latência das aplicações e o consumo de energia do sistema.

A Figura 7 ilustra um método geral para a variação dinâmica da frequência do sinal de relógio de um sistema. Neste método, um monitor analisa constantemente o estado atual do sistema, que pode incluir, por exemplo, exame da carga de trabalho do processador [YUH09], da carga da bateria de alimentação ou da temperatura do circuito integrado [JON06].

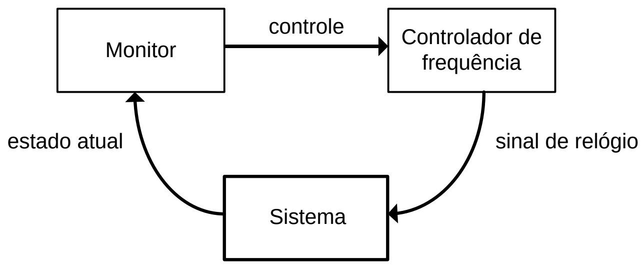

Figura 7 – Possível método para o controle da frequência do sinal de relógio na técnica DFS [YUH09].

Conforme estes dados, o monitor define uma nova frequência e envia sinais de controle para o controlador de frequência, que por sua vez altera a frequência do sinal de relógio do sistema. Vale ressaltar ainda que considerações acerca da implementação do controlador de frequência devem ser realizadas, uma vez que o a frequência do sinal de relógio deve ser alterada adequadamente, a fim de prevenir qualquer prejuízo no funcionamento e no desempenho do sistema.

Com o emprego de técnicas DFS a frequência de operação pode ser diminuída, por exemplo, quando a bateria apresentar um baixo nível de carga. Desta forma, o consumo de energia é reduzido, prolongando-se o tempo de funcionamento do sistema embarcado. Quando a temperatura do circuito integrado alcançar um limite crítico, a diminuição da frequência evita que o circuito integrado apresente um mau funcionamento, proveniente do calor provocado pela alta dissipação de potência. Ou ainda, quando a carga de trabalho do processador estiver baixa, pode-se diminuir a frequência para economizar a carga da bateria.

Projeto GALS e Interfaces Assíncronas

O uso de um sinal de relógio global no projeto de sistemas digitais é a principal característica do *estilo síncrono* de projeto. Entretanto, com o constante avanço dos dispositivos VLSI, problemas como a distribuição, escorregamento e dissipação de potência do sinal de relógio, são cada vez mais difíceis de se resolver. Conforme [ITR05], a complexidade de sistemas VLSI coloca em risco paradigmas consolidados, entre eles o uso do estilo síncrono, pois os circuitos resultantes tendem a se tornar menos confiáveis, devido aos limites de potência e ao custo para manter sua robustez, em relação ao esforço de projeto exigido.

Com o objetivo de superar as limitações do estilo síncrono de projeto, diversas pesquisas estão retomando o interesse no desenvolvimento de circuitos assíncronos [AMD05, SPA02, COR00], onde não se assume o pressuposto de discretização do tempo, isto é, o tempo é tratado como uma variável contínua [SPA02]. Esta abordagem reduz os problemas de escorregamento e de dissipação de potência do sinal de relógio [CAL98]. Porém, o projeto de sistemas puramente assíncronos esbarra ainda na falta de ferramental adequado para a automatização do processo de desenvolvimento. Devido às limitações do estilo síncrono de projeto, a falta de ferramentas para suportar circuitos totalmente assíncronos, e a grande popularidade do estilo síncrono de projeto, alguns trabalhos propõem soluções intermediárias entre os dois paradigmas. O objetivo é manter as ferramentas do projeto síncrono e eliminar, ou simplificar, o uso de sincronização através do sinal de relógio. Entre tais propostas pode-se destacar o uso de sistemas globalmente assíncronos e localmente síncronos, do inglês, Globally Asynchronous, Locally Synchronous ou GALS [CHA84].

Como se observa na Figura 8, a aplicação da técnica GALS consiste no particionamento de um determinado módulo de hardware em diversos sub-módulos, ou ilhas síncronas, onde cada uma possui seu próprio relógio [CHA84, MUT00]. A transferência de informações entre tais sub-módulos é efetuada empregando-se uma interface de comunicação assíncrona [CHA84]. Como vantagem desta abordagem cita-se a redução dos problemas relacionados ao controle da distribuição e do escorregamento de um relógio global, uma vez que de cada sub-módulo possui seu relógio local. Outra vantagem em potencial é a redução do consumo de energia, já que frequência do sinal de relógio de sub-módulos que não executem tarefas críticas pode ser diminuída. A principal desvantagem desta técnica é que, dependendo da quantidade total de sub-módulos derivados do particionamento, pode ocorrer degradação no desempenho global do sistema devido à sobrecarga de comunicação entre estes [SEM04]. O projeto GALS, portanto, implica descobrir um particionamento que mantenha ao máximo as características e propriedades originais do sistema e também em definir qual método, ou interface, será utilizado para a troca de informação entre os domínios [SEM04].

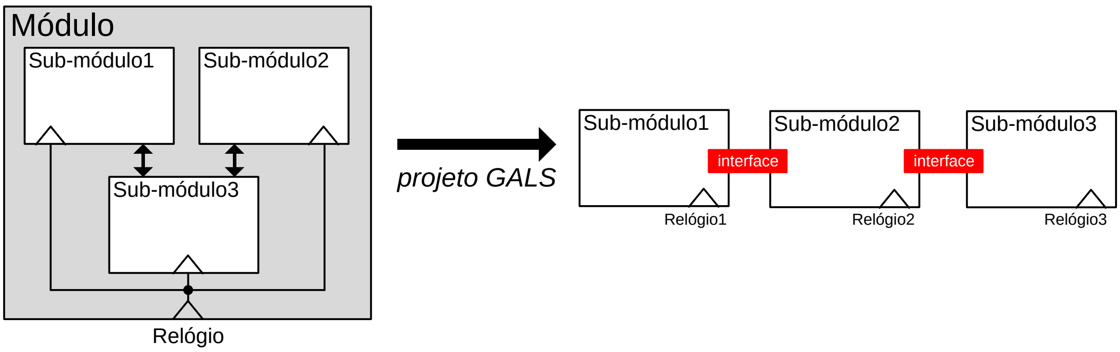

Figura 8 – Exemplo de aplicação da técnica GALS de projeto.

A literatura classifica os estilos de projeto GALS de acordo com os métodos utilizados para a transferência de dados entre os diversos sub-módulos [MES90]. Todos estes estilos devem prover formas de evitar fenômenos tais como a metaestabilidade [COM74] decorrente da diferença de frequência e/ou fase entre módulos que comunicam. Tais estilos são classificados em três categorias: relógio pausável, assíncrono e quase síncrono. O estilo relógio pausável faz uso de interfaces para a comunicação entre os sub-módulos que podem alongar (do inglês, *stretch*) ou pausar o sinal de relógio para que a transferência de dados ocorra. Nesta categoria pode-se citar as técnicas "*clock stretching*" [PON07] e "*gated clock*" [AMI07]. Estas técnicas utilizam componentes não-síncronos, tais como mutex, C-element e elementos de atraso. No estilo assíncrono são utilizados circuitos conhecidos como sincronizadores. Nesta categoria é possível identificar os seguintes tipos de sincronizadores:

- sincronizador *2FF* [GIN03, KIL07], ou "two-flop", que consiste na adição de dois registradores em todos os sinais de comunicação entre sub-módulos;
- sincronizador "phase control" [KIL07], que faz uso de circuitos especiais tais como *Phase Locked Loop* (PLL) ou *Delay Locked Loop* (DLL) para ajustar as fases das frequências de operação de sub-módulos que se comunicam;
- sincronizador baseado em FIFO (First-In-First-Out) e em protocolo de comunicação [KIL07], que transfere os dados entre sub-módulos.

No estilo quase síncrono existe uma relação de frequência e/ou fase bem conhecida entre os sinais de relógios dos sub-módulos. Neste caso não há necessidade do uso de sincronizadores, protocolos ou interfaces de comunicação.

A técnica GALS constitui uma abordagem natural no projeto de sistemas em um único chip (do inglês, *System-on-Chip*). Além de sanar alguns dos problemas do estilo síncrono de projeto, constitui uma alternativa interessante para a redução do consumo de energia [IYE02]. A perda de desempenho, causada pela sobrecarga de comunicação pode ser bastante minimizada se uma análise correta do particionamento do processador for realizada e se mecanismos especializados de comunicação forem empregados, tais como filas bem dimensionadas. Alguns estudos apontam uma degradação de 7% a 20% no desempenho geral de um sistema do tipo GALS devido à sobrecarga de comunicação [SEM04, IYE02]. Esta degradação é função da quantidade total de unidades funcionais de um sistema, ou seja, ocorre devido à taxa de comunicação entre as diferentes unidades entre as quais comunicações devem ser sincronizadas. Algumas técnicas são propostas para amenizar este problema, sendo possível reduzir até 94% da degradação [SEM04]. Independente da degradação ou não do desempenho, o emprego de técnicas GALS traz o benefício potencial de redução do consumo total de energia elétrica. Em [IYE02], os autores afirmam que uma redução de até 20% no consumo de energia é possível.

4. MATERIAIS E MÉTODOS

Este Capítulo descreve o conjunto de recursos usados, reutilizados ou produzidos de forma automática e empregado no trabalho aqui proposto Como elemento processador adotou-se a organização de código aberto Plasma, apresentada a seguir, na Seção 4.1. As Seções 4.2 e 4.3 apresentam respectivamente a unidade de ponto flutuante de código aberto FPU100 e unidades de ponto flutuante geradas com o auxílio da ferramenta CoreGen da Xilinx. Tais unidades foram adotadas na implementação de coprocessadores que permitem a exploração do espaço de projeto de um processador embarcado dotado de capacidade de processamento de ponto flutuante em hardware. A Seção 4.4 apresenta recursos e métodos utilizados para a medição de potência em FPGAs.

4.1. A Organização Plasma

Diversas instâncias de arquiteturas MIPS existem disponíveis para o projeto de sistemas embarcados. Tais módulos são distribuídos, por exemplo, como núcleos de hardware sintetizáveis, implementados em linguagens de descrição de hardware, tais como VHDL ou Verilog. Através do reuso destas distribuições pode-se facilmente modificar o projeto original ou adicionar outros núcleos de hardware ao processador, incluindo periféricos de comunicação, controladores de memória, e coprocessadores. Assim, diminui-se o tempo de projeto e se alcançam requisitos para um sistema embarcado.

A organização denominada Plasma [OPE08a] foi adotada neste trabalho. O Plasma é compatível com a organização MIPS-I, foi descrito em VHDL e possui código aberto. Diferentemente de muitos processadores RISC, sua organização de memória é do tipo *Von Neumann* e não *Harvard*, ou seja, instruções e dados compartilham a mesma unidade de memória e o mesmo barramento. Esta organização implementa todas as instruções da arquitetura MIPS-I, com exceção das instruções de *load* e *store* desalinhado, uma vez que tais instruções são patenteadas. O Plasma é disponibilizado com compilador C (GCC) e sistema operacional com suporte para aplicações em tempo real. Inclui periféricos tais como porta de comunicação serial bidirecional, controlador DDR SDRAM, interface de comunicação MAC Ethernet e interface Flash. Possui um temporizador e controlador de interrupções, porém nenhuma de suas versões possui unidade de gerenciamento de memória e tampouco coprocessador de ponto flutuante, ou CP1. Sua versão de coprocessador de controle, ou CP0, prevê somente as estruturas destinadas ao tratamento de algumas exceções e a habilitação das interrupções. Na Figura 9 observa-se o processador Plasma e sua interface de comunicação com a memória de instruções e dados.

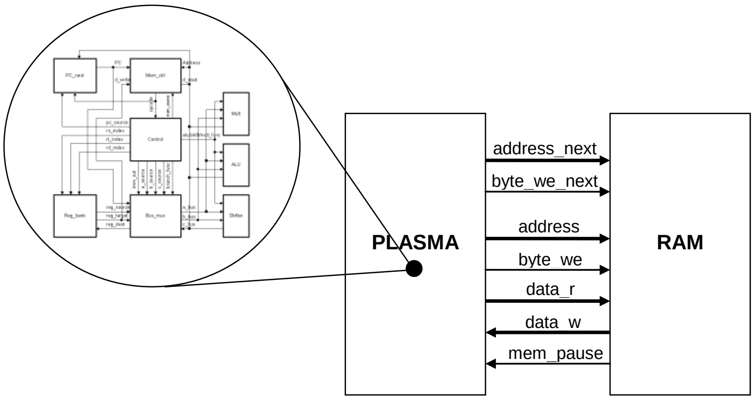

Figura 9 – Organização Plasma e sua interface de comunicação com a memória RAM [OPE08a].

4.2. Unidade de Ponto Flutuante FPU100

A FPU100 [OPE08b] é uma unidade de ponto flutuante de código aberto descrita em VHDL. Ela é compatível com o padrão IEEE-754, porém dá suporte somente ao formato de precisão simples. A FPU100 provê cinco operações aritméticas: adição, subtração, multiplicação, divisão e raiz quadrada. As operações de adição e subtração possuem uma latência de 7 ciclos de relógio, a multiplicação 12 e a divisão 35 ciclos de relógios. O código fonte responsável pela operação de raiz quadrada foi removido no escopo deste trabalho, por não fazer parte do conjunto de instruções de ponto flutuante da arquitetura MIPS-I. A unidade dá suporte aos quatro modos de arredondamento da norma e todas as exceções são assinaladas quando detectadas e seu tratamento é deixado para o software. Operações de comparação e de conversão entre ponto flutuante e inteiro, e vice-versa, não estão disponíveis. A Figura 10 mostra o diagrama de blocos do módulo FPU100.

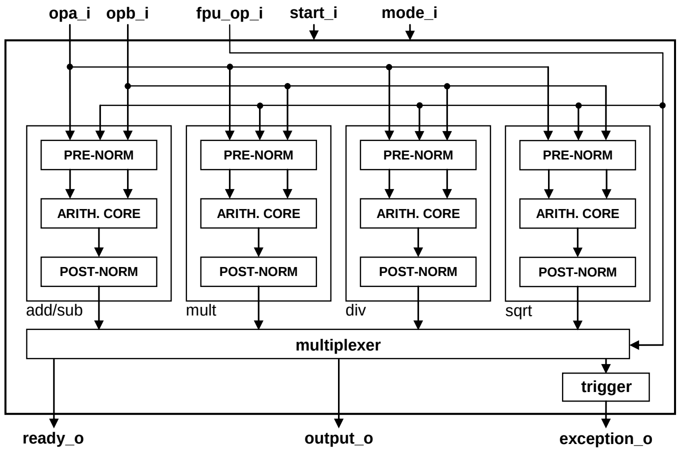

Figura 10 – Diagrama de blocos da unidade FPU100 [OPE08b].

Além dos sinais **clk_i** e **rst_i**, sua interface de entrada possui: dois sinais de 32 bits, **opa_i** e **opb_i**, que representam os operandos; o sinal **fpu_op_i** de 3 bits para a seleção da operação desejada; **rmode_i**, de 2 bits, utilizado para determinar o modo de arredondamento do resultado; e o sinal **start_i**, utilizado para iniciar a operação desejada. Sua interface de saída possui o sinal **output_o**, de 32 bits, para o resultado da operação, o sinal **ready_o**, que indica o término da operação, e 8 sinais correspondentes às exceções passíveis de ocorrência:

- **ine_o** sinaliza resultado com valor inexato;
- **overflow_o** indica que o resultado ultrapassou o limite superior de representação do padrão;
- **underflow_o** indica que o resultado ultrapassou o limite inferior de representação do padrão;
- **div_zero_o** indica a ocorrência de divisão por zero;
- **inf_o** e **zero_o** indicam exceções que não fazem parte de padrão 754 da IEEE e são utilizados na FPU100 para detectar mais rapidamente a ocorrência de resultados infinitos e resultados zerados, independente da maneira que tenham ocorrido;
- **qnan_o** e **snan_o** (Not-a-Number) indicam que pelo menos um dos operandos é inválido, ou um valor especial.

4.3. Unidades de Ponto Flutuante CoreGen

O software Coregen [XIL09b] é uma ferramenta de geração automática parametrizável de módulos de hardware específicos para FPGAs da Xilinx. Entre as parametrizações possíveis na geração dos módulos de hardware, há a definição de interfaces de dados, tamanho em bits de portas de entrada e saída, protocolos de comunicação, latência, etc. Com o auxílio desta ferramenta gerou-se módulos de hardware para adição, subtração, multiplicação e divisão em ponto flutuante, além de módulos para a conversão entre números de ponto flutuante e inteiros e de comparação de números em ponto flutuante. Geraram-se duas versões destes módulos no Coregen: uma versão com a latência mínima permitida pela ferramenta (1 ciclo de relógio) em todas as operações, e outra versão com a latência máxima permitida (16 ciclos de relógio para adição e subtração, 8 na multiplicação, 28 na divisão, 3 para comparação, e 6 ciclos de relógio para conversão). A partir da geração destes módulos implementaram-se duas unidades de ponto flutuante, uma unidade de latência mínima e outra de latência máxima. Apesar do CoreGen também permitir a parametrização do formato de precisão conforme a IEEE-754, todos os módulos foram gerados somente com precisão simples para permitir comparação com a unidade FPU100. Ainda com a finalidade de minimizar diferenças no coprocessador de ponto flutuante, as interfaces das duas unidades implementadas foram mantidas idênticas à unidade FPU100, excetuando-se os sinais **start_o** e **ready_i** na versão de latência mínima, uma vez que todas as suas operações são executadas em apenas um ciclo de relógio. Na Figura 11 pode-se visualizar o diagrama de blocos da unidade de ponto flutuante implementada com o auxílio da ferramenta CoreGen.

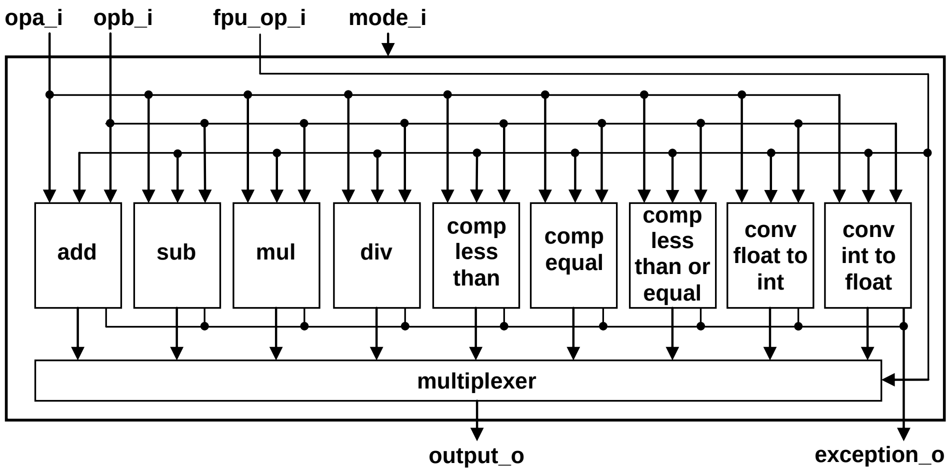

Figura 11 – Diagrama de blocos da unidade implementada com o auxílio da ferramenta Coregen.

4.4. Medidas de Potência em FPGAs

Esta Seção apresenta os fundamentos necessários para o cálculo prático da potência dissipada pelo FPGA durante a avaliação das diversas organizações prototipadas, bem como seus respectivos consumos de energia. A Seção 0 contém uma breve base teórica sobre um método que permite o cálculo da dissipação de potência de um FPGA e de seu consumo de energia. A Seção 0 apresenta o método específico aplicado e os recursos utilizados no presente trabalho, para a obtenção das medidas.

Método de Medida

A medição da dissipação de potência no presente trabalho baseia-se no método utilizado no trabalho de Becker & Huebner [BEC03], onde se estuda a avaliação da dissipação de potência de FPGAs para circuitos prototipados e também para circuitos reconfigurados dinamicamente. Este trabalho também analisa a potência dissipada pelo FPGA durante a reconfiguração, parcial ou completa, de um circuito previamente prototipado. Para calcular a potência dissipada pelo FPGA, estes autores utilizaram o sistema de medição apresentado na Figura 12.

Figura 12 – Sistema de medição utilizado em [BEC03].

Neste sistema, utiliza-se uma placa de prototipação, um osciloscópio e um computador. A placa de prototipação possui um *slot* para um resistor *shunt*, que é conectado em série com a fonte de alimentação do FPGA. Medindo a tensão instantânea sobre este resistor, e sendo conhecida a sua resistência com precisão, é possível calcular a potência instantânea (P_{inst}) dissipada pelo FPGA. Esta potência pode ser calculada através da Equação 5. Nesta Equação, V_{VCC} é tensão de alimentação do núcleo do FPGA, V_{shunt} é tensão medida sobre o resistor *shunt*, e R_{shunt} é a sua resistência.

$$P_{inst} = \frac{(V_{VCC} - V_{shunt}) \cdot V_{shunt}}{R_{shunt}} \tag{5}$$

Para calcular a potência média dissipada ($P_{média}$) por um circuito, prototipado no FPGA, durante um determinado período tempo, aplica-se a Equação 6. Esta Equação é baseada em uma integral definida aplicada sobre uma função contínua. No esquema proposta na Figura 12 utilizou-se o osciloscópio para acumular valores instantâneos de tensão (V_{shunt}) amostrados com uma determinada taxa ao longo do período T de tempo, onde $T = T_{fim} - T_{inicio}$. Assim, os valores de potência média realmente computados são aproximados e atingem o valor da Equação 6 apenas para uma taxa de amostragem infinita. Na prática, taxas de amostragem altas, mas viáveis, podem levar a erros desprezíveis, como será detalhado na Seção 0.

$$P_{média} = \frac{1}{T} \cdot \int_{T_{inicio}}^{T_{fim}} P_{inst} \cdot dt \tag{6}$$

O tempo T pode ser definido e controlado por software no computador hospedeiro. As amostras armazenadas no osciloscópio são transferidas para o hospedeiro, que possui um software para o cálculo da potência média dissipada. Além disso, o hospedeiro também é responsável por configurar o FPGA da placa de prototipação.

O consumo de energia, por outro lado é definido matematicamente, através da Equação 7 [RAB96] como a integral da dissipação de potência deste no intervalo de tempo T em que estava em operação. No caso de se trabalhar com amostragem, $E_{consumida}$ pode ser aproximada pelo produto de $P_{média}$ por T.

$$\mathrm{E_{consumida}} = \int_0^{T} \mathrm{P} \cdot \mathrm{dt} \tag{7}$$

Recursos de Medida

Neste trabalho, descarta-se o uso de estimativas da dissipação de potência para FPGAs, uma vez que ferramentas comerciais disponíveis atualmente não possuem uma precisão satisfatória ou necessitam de uma quantidade muito grande de dados de simulação para obter tal precisão [HYU05]. Sendo assim, optou-se pela medição real da dissipação de potência. Para alcançar este intento, adquiriu-se a plataforma ML550 [XIL09c] da Xilinx, que contém um FPGA da família Virtex-5, modelo XC5VLX50T-1, 4 osciladores (250, 200, 133 e 33 MHz) e 2 circuitos sintetizadores de relógio, totalizando 8 fontes do sinal de relógio, além de outros dispositivos e periféricos. Este plataforma foi concebida especificamente para a medição da dissipação de potência. Por esta razão, o fabricante incluiu em seu projeto circuitos integrados que possuem em seu interior resistores *shunt* de alta precisão que possibilitam o cálculo da dissipação de potência do FPGA. Estes resistores estão conectados em série com os reguladores de tensão que alimentam o FPGA como mostra a Figura 13.

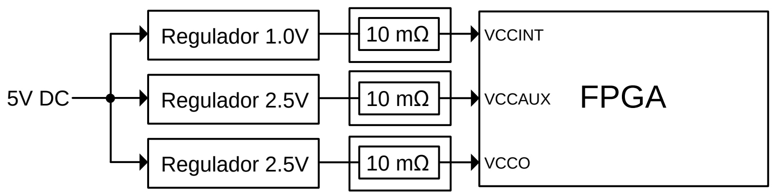

Figura 13 – Tensões de alimentação do FPGA e seus respectivos resistores *shunt*.

Como se observa nesta Figura, o FPGA usa três linhas de alimentação [XIL09f]: VCCINT, a tensão de alimentação do núcleo do FPGA que implementa a lógica dos módulos de hardware; VCCAUX, a tensão de alimentação de módulos auxiliares tais como DCMs, controlador JTAG, *drivers* LVDS, etc.; e VCCO, a tensão de alimentação do anel de *pads* dos pinos de entrada e saída do FPGA. Portanto, através dos três resistores de precisão é possível calcular separadamente a potência dissipada em cada uma das três fontes de alimentação do FPGA. Neste trabalho optou-se somente pela medição da tensão sobre o resistor de precisão conectado ao VCCINT, uma vez que o mesmo é responsável pela maior parte da potência (estática e dinâmica) dissipada pelo FPGA [XIL09e]. Além do mais, a infraestrutura definida para a prototipação das organizações possui uma interface de dados reduzida e utiliza apenas um DCM e dois *buffers* para o sinal de relógio. Por estes motivos, as tensões sobre os resistores de precisão conectados ao VCCAUX e VCCO não precisam ser medidas.

A partir da medida das tensões sobre os resistores de precisão, é possível calcular a dissipação instantânea da potência do FPGA. Para o cálculo desta dissipação, a Xilinx sugere a Equação 8 [XIL09c]. Nesta Equação, ***P*** é a potência em Watts; ***V*** é a tensão do regulador; ***Vs*** é a tensão medida no *shunt resistor* e ***Rs*** é a sua resistência. Ressalta-se ainda, que a potência dissipada pelo resistor *shunt* é desprezível e, portanto, não é considerada na Equação 8.

$$P = V \cdot (Vs/Rs) \tag{8}$$

A medida das tensões sobre os resistores *shunt* é efetuada através de pinos que dá acesso aos circuitos integrados que os contêm. Conforme a Figura 13, VCCINT demanda uma tensão de 1 volt. VCCAUX e VCCO demandam ambos 2,5 volts. Os resistores de precisão possuem todos 10 mΩ e 0,1% de tolerância. Substituindo estes valores na Equação 8 deriva-se a Equação 9.

$$P_{VCCINT} = 1 \cdot (Vs/0{,}010) \tag{9}$$

Para efetuar a medição da tensão no resistor de precisão utilizou-se o osciloscópio Agilent Infiniium DSA80000B [AGI09]. Este equipamento permite medições de sinais de alta frequência e dá suporte a medidas de sinais com frequências de até 4 GHz. Sua taxa de amostragem é de no máximo 40 Gsa/s (ou 40 bilhões de amostras por segundo) e sua memória de armazenamento é capaz de armazenar um total de 64 milhões de amostras. A configuração da taxa de amostragem e o número de amostras armazenadas em memória são efetuados automaticamente pelo osciloscópio, e embora ambos estejam intrinsecamente relacionadas, é possível configurá-los manualmente. Neste trabalho, configurou-se manualmente a taxa de amostragem, sendo a mesma configurada para a maior taxa possível. Desta forma é possível aproximar ao máxima as Equações 6 e 7. A definição da maior taxa de amostragem possível é função do número de amostras necessárias para uma determinada amostragem, sendo que este número não pode exceder o tamanho da memória disponível no osciloscópio: 64 milhões de amostras. Entende-se por amostragem o conjunto de amostras armazenadas na memória do osciloscópio para um determinado circuito prototipado em FPGA durante o tempo de execução de um algoritmo em hardware. Grandes intervalos de tempo exigem um grande número de amostras e se a memória disponível for excedida torna-se necessário utilizar taxas de amostragem menores. O osciloscópio alerta quando se escolhe de um par taxa/tempo de amostragem que extrapola o tamanho da sua memória.

A amostragem da tensão sobre o resistor de precisão durante o tempo de execução de um algoritmo em FPGA deve ser efetuada no osciloscópio através de uma janela de amostragem. Tal janela é definida como o intervalo de tempo em que o módulo, ou circuito, prototipado executa sua computação. Para implementar esta janela, deve-se criar junto ao circuito um sinal de gatilho, que indica ao osciloscópio o início e o fim da computação. Neste trabalho, implementou-se um módulo de hardware adicional responsável por detectar, no barramento de dados da organização Plasma, valores pré-determinados para identificar o início e o fim da computação, “1BABABAB” e “1CACACAC” na base hexadecimal, respectivamente. Tais valores são escritos na memória RAM da organização Plasma através de instruções *sw* (*store word*) inseridas no início e no fim do código fonte de todas as aplicações desenvolvidas. Quando se detecta no barramento de dados o valor “1BABABAB”, atribui-se o nível lógico ‘1’ ao sinal de gatilho. O mesmo permanece neste nível até que se detecte o valor “1CACACAC”, quando então imediatamente vai para o nível lógico ‘0’. Ressalta-se que somente as amostras presentes no visor do osciloscópio podem ser exportadas para arquivo e, por esta razão, a “janela de amostragem” deve ser ajustada para o tamanho exato do visor do osciloscópio. Este ajuste é realizado configurando-se a escala horizontal do osciloscópio. Na Figura 15 observa-se um exemplo deste ajuste.

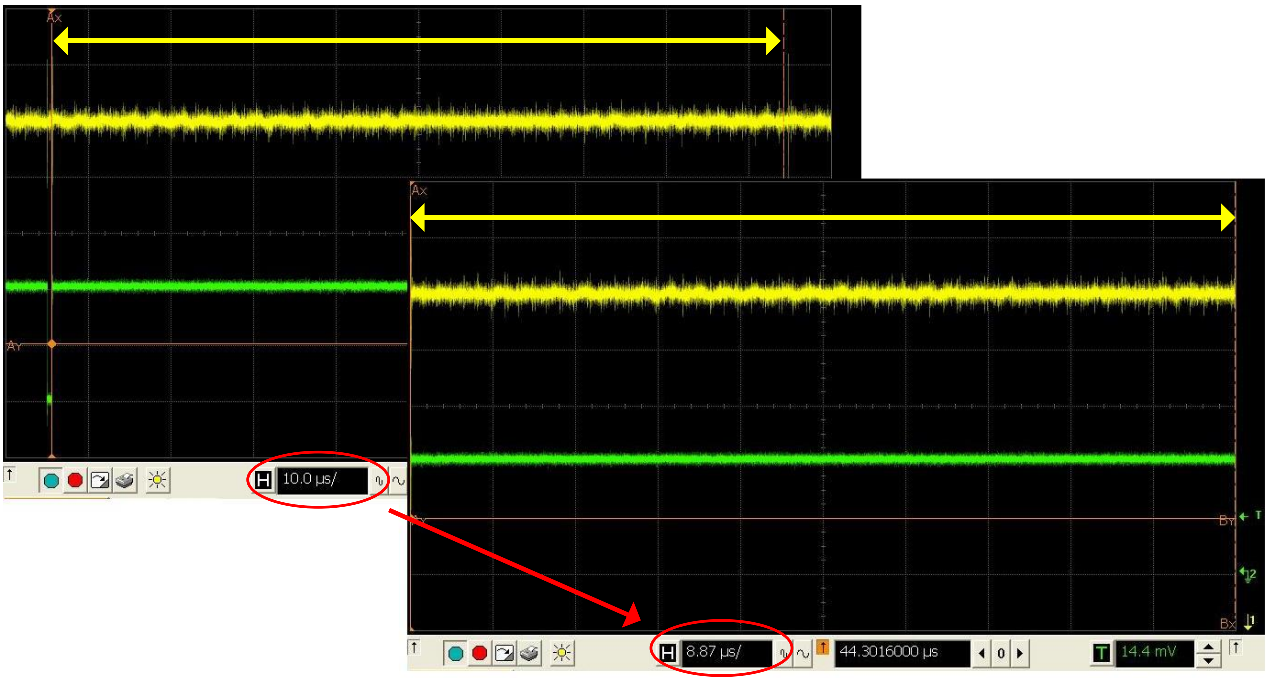

Figura 14 – Ajuste da “janela de tempo” ao visor do osciloscópio, através da configuração da escala horizontal.

Se a computação for executada em laço infinito, ou por um determinado número de vezes, é possível calcular a média das amostragens efetuadas através da janela de amostragem. Desta forma, é possível obter resultados mais precisos. O osciloscópio disponibiliza uma série de funções matemáticas que podem ser aplicadas nos sinais medidos em seus canais. Uma delas é a função média. O osciloscópio permite que se calcule uma média de até 65536 janelas de amostragem. Neste trabalho definiu-se 16384 como o número total de janelas a ser calculado pela função média. Esta definição teve origem no tempo total necessário para tal cálculo. Com este número, leva-se aproximadamente noventa minutos para obter o resultado do cálculo da média. Com números maiores é possível obter uma média com uma precisão ainda maior. Entretanto, o tempo necessário para o seu cálculo torna-se demasiadamente extenso. Na Figura 15 visualiza-se o sistema específico montado neste trabalho para a amostragem da tensão sobre o resistor de precisão conectado ao VCCINT do FPGA da plataforma ML550.

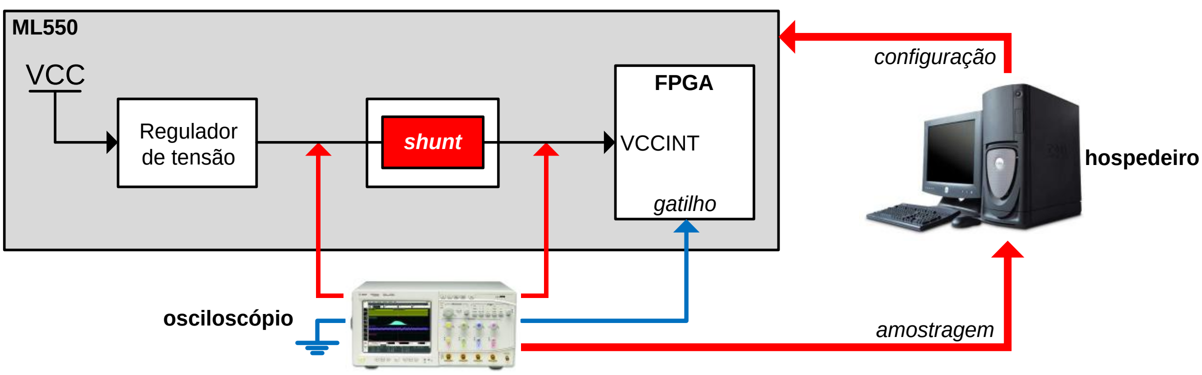

Figura 15 – Sistema para a medição da tensão no resistor de precisão.

Após o calculo da média das amostragens, é possível exportar as amostras armazenadas na memória do osciloscópio para um arquivo no computador hospedeiro. O arquivo exportado possui a extensão “tsv” (*tab separated values*) e apresenta como conteúdo linhas que são divididas em duas colunas: a primeira coluna contém o instante, em segundos, de todas as amostras; a segunda coluna contém os valores, em Volts, das tensões amostradas sobre o resistor de precisão conectado ao VCCINT. Este é o valor que deve substituir a variável ***Vs*** na Equação 9.

As amostras da segunda coluna do arquivo são lidas por um programa denominado *calcPOT* (código fonte no Apêndice D), desenvolvido no escopo deste trabalho com o objetivo de calcular:

- a potência instantânea dissipada para cada amostra (Equação 10);
- a energia consumida (Equação 11);
- a potência média dissipada (Equação 12);
- a máxima potência instantânea dissipada (Equação 13).

$$P_{inst} = 1 \cdot (Vs/0{,}010) \tag{10}$$

$$E_{consumida} = \Delta t \cdot \sum_{i=1}^{n} |P(i)_{inst}| \tag{11}$$

$$P_{média} = \frac{E_{consumida}}{T_{execução}} \tag{12}$$

$$P_{pico} = \max\,(para\ todo\ P_{inst}) \tag{13}$$

A Equação 10 é aquela sugerida pela Xilinx para o cálculo da potência instantânea dissipada. A Equação 11, utilizada para o cálculo da energia consumida, foi baseada na Equação 7. A integração foi implementada através do somatório dos valores absolutos da primeira amostra (*i=1*) até a última (*i=n*). O Δt, ou período de amostragem (inverso da frequência de amostragem), é colocado em evidência e multiplica o somatório. Isto ocorre porque o mesmo é base da área de todas as amostras do sinal, tal como se observa na Figura 16.

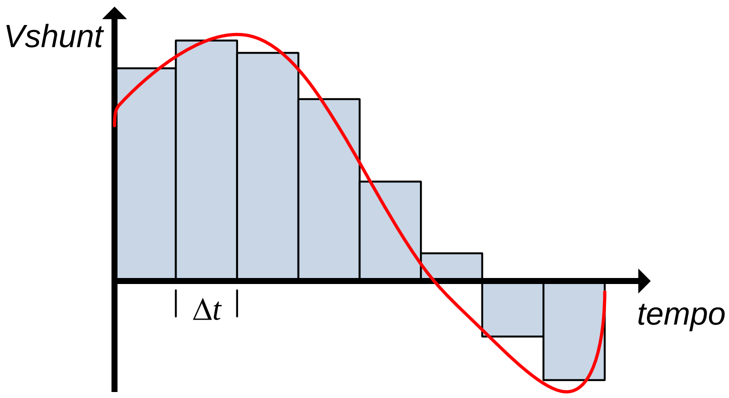

Figura 16 – Relação entre área sob a curva de uma função contínua e a área resultante de um somatório de amostras obtidas a uma taxa de amostragem fixa.

A potência média dissipada, calculada através da Equação 12, é a divisão da energia consumida (Equação 11) pelo tempo de execução de um determinado circuito prototipado no FPGA (no caso deste trabalho, de uma aplicação sendo executada na organização Plasma). A potência instantânea máxima dissipada (Equação 13) é a maior potência entre todas as potências instantâneas dissipadas, calculadas para uma determinada amostragem.

5. IMPLEMENTAÇÃO DE UM CP1 DA ORGANIZAÇÃO MIPS-I

Neste Capítulo são detalhados os três coprocessadores implementados durante a pesquisa propostos pelo Autor. A Seção 5.1 apresenta os detalhes de uma “implementação-base” que foi utilizada na implementação de todos os coprocessadores. Obviamente cada coprocessador demanda alterações nesta base, sendo as mesmas devidamente relatadas em suas respectivas Seções. A Seção 5.2 apresenta a implementação do coprocessador que emprega a unidade FPU100, denominado de HFP100. A Seção 5.3 apresenta os coprocessadores que empregam as unidades de ponto flutuante de latência mínima e máxima gerados com o auxílio da ferramenta, denominados respectivamente de HFPmin e HFPmax. Finalizando o Capítulo, a Seção 5.4 apresenta as estimativas de ocupação de área e da frequência de operação oriundas das ferramentas de síntese lógica e física.

5.1. Implementação Base dos Coprocessadores

Não se encontra na literatura informações detalhadas sobre a implementação de um coprocessador de ponto flutuante para um processador compatível com a organização MIPS-I. Por este motivo, a implementação de um CP1 para o processador Plasma partiu do zero. Entre as informações não encontradas, a de maior relevância diz respeito à transferência de dados entre o processador e coprocessador e entre o coprocessador e a memória. Para determinar esta informação, estudou-se a transferência de dados entre o Plasma e sua memória RAM. Este estudo compreendeu a análise das instruções de *sw* (*store word*) e *lw* (*load word*), bem como dos seus respectivos comportamentos nas interfaces de dados do Plasma e da memória RAM. Desta forma foi possível definir um procedimento para a transferência de dados entre o CP1 e Plasma e entre o CP1 e memória RAM. Sabe-se ainda que o CP1 deve contemplar os formatos de precisão simples, dupla e possíveis formatos de precisão estendida, conforme previsto pelo padrão IEEE-754, e que deve possuir um banco de registradores de 32 posições que armazene palavras de 32 bits. Portanto, a implementação minimalista de um CP1 para o Plasma deve empregar três módulos: um banco de registradores, uma unidade de ponto flutuante e um módulo de controle.

O banco de registradores empregado é o mesmo utilizado pelo processador MR2 [ARQ08], alvo de estudos nas disciplinas de graduação da PUCRS. Como unidade de ponto flutuante empregou-se as três unidades de ponto flutuante: a unidade FPU100 e as duas unidades implementadas com o auxílio da ferramenta CoreGen. O módulo de controle, responsável pela decodificação das instruções de ponto flutuante, pelo controle da unidade de ponto flutuante, do banco de registradores e de outras estruturas adicionais, tais como registradores e multiplexadores, foi implementado partindo-se do zero.

O módulo de controle denominado *control_unit*, é basicamente uma máquina de estados, que além de ser responsável pelo controle de todos os módulos, é responsável também pela execução das instruções de ponto flutuante do conjunto do MIPS-I que não são providas pelas unidades de ponto flutuante. Estas instruções compreendem: as operações de valor absoluto e de valor negado de um número; as operações de conversão e comparação entre números; a operação de movimento de conteúdo entre registradores do banco de registradores do CP1; e as operações de transferência de dados entre a CP1 e Plasma, e entre a CP1 e a memória RAM.

O funcionamento da máquina de estados do módulo *control_unit* baseia-se nos campos do formato das instruções, observado na Figura 17, e é relativamente simples. A máquina de estados permanece em estado de espera até que receba uma instrução de ponto flutuante. Uma instrução de ponto flutuante é identificada pelo valor "11", na base hexadecimal, no campo *opcode* do formato da instrução. Após a detecção de uma instrução de ponto flutuante ocorre a decodificação da mesma, através dos campos *format* e *function*. Também após esta detecção é efetuada a sinalização de coprocessador ocupado. O banco de registradores é indexado, através dos campos *ft*, *fs* e *fd*, e os operandos são determinados. A operação desejada é sinalizada e executada pela unidade de ponto flutuante, ou pelos módulos adicionais quando a operação não é provida pela unidade de ponto flutuante. A máquina de estados aguarda o término da operação para então armazenar o resultado no banco de registradores. Por fim, ocorre a sinalização de coprocessador não ocupado.

31					0
opcode	**format**	**ft**	**fs**	**fd**	**function**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Figura 17 – Formato das instruções de ponto flutuante.

5.2. Coprocessador HFP100

A proposta de implementação do primeiro CP1, visualizado na Figura 18, emprega a unidade de ponto flutuante FPU100 e foi denominado pelo Autor como HFP100 (HFP é acrônimo para *Hardware Floating Point*). Este coprocessador utiliza a "implementação-base" descrita na Seção anterior e a única modificação necessária se restringe à exclusão do hardware da operação de raiz quadrada na unidade FPU100, já que a mesma não faz parte do conjunto de instruções MIPS-I.

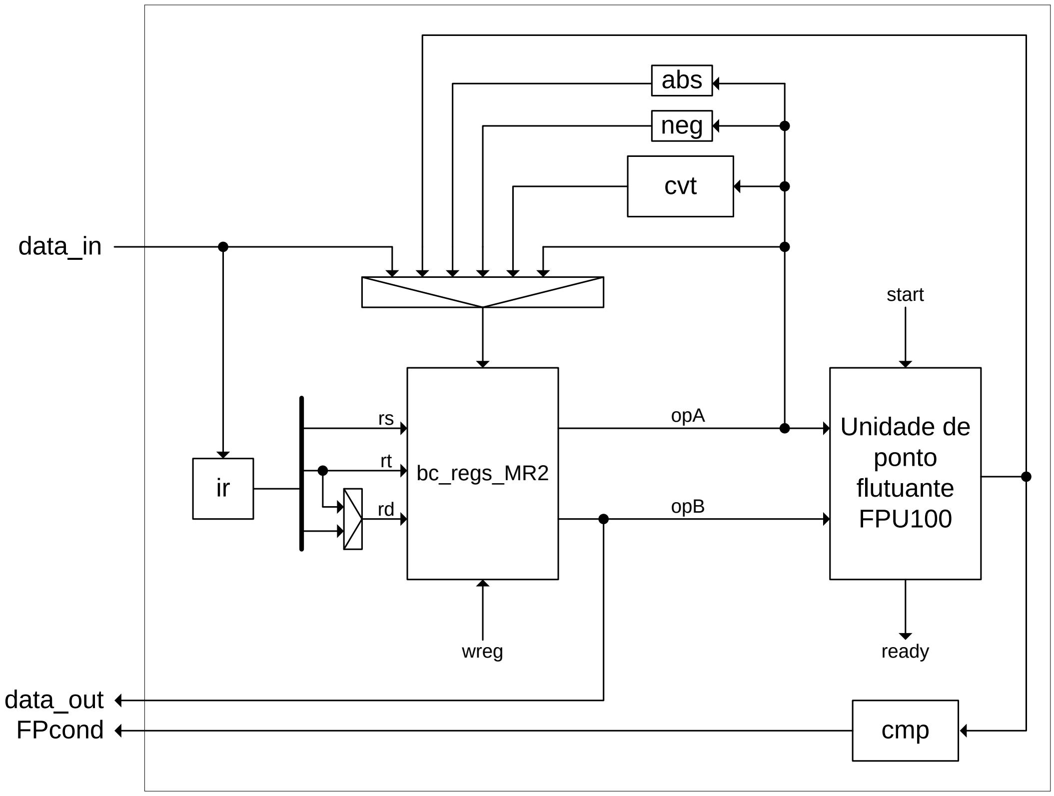

Figura 18 – Diagrama de blocos do CP1 proposto implementado com a unidade FPU100.

Na Figura 18 alguns sinais da interface de dados do CP1 foram abstraídos, uma vez que as versões, síncrona e assíncrona, da organização denominada Plasma-HFP demandam diferentes interfaces. Suas respectivas interfaces serão abordadas e detalhadas no Capítulo 6 a seguir.

5.3. Coprocessadores HFPmim e HFPmax

Os coprocessadores denominados pelo Autor de HFPmin e HFPmax empregam, respectivamente, as unidades de ponto flutuante de latência mínima e de latência máxima, implementadas com o auxílio da ferramenta CoreGen. Com o intuito de minimizar possíveis alterações na "implementação-base", as interfaces destas unidades de ponto flutuante foram mantidas idênticas à interface da unidade FPU100, excetuando-se os sinais **start_o** e **ready_i** na unidade de latência mínima. A exclusão destes sinais no HFPmin exigiu pequenas modificações na máquina de estados do módulo de controle *control_unit* da implementação base. Além disso, outra modificação necessária na "implementação-base" de ambos HFPmin e HFPmax, residiu na exclusão dos módulos adicionais responsáveis pelas operações de conversão e comparação entre números, uma vez que tais operações são realizadas pelas unidades de ponto flutuante implementadas com o auxílio do CoreGen. Ressalta-se que apesar das modificações efetuadas, os funcionamentos dos coprocessadores HFPmin e HFPmax permanecem iguais ao funcionamento do coprocessador HFP100. Na Figura 19 observa-se o diagrama de blocos do coprocessador HFPmin proposto. O diagrama de blocos do HFPmax é similar, diferindo apenas nos sinais **start_o** e **ready_i** existentes na unidade de ponto flutuante.

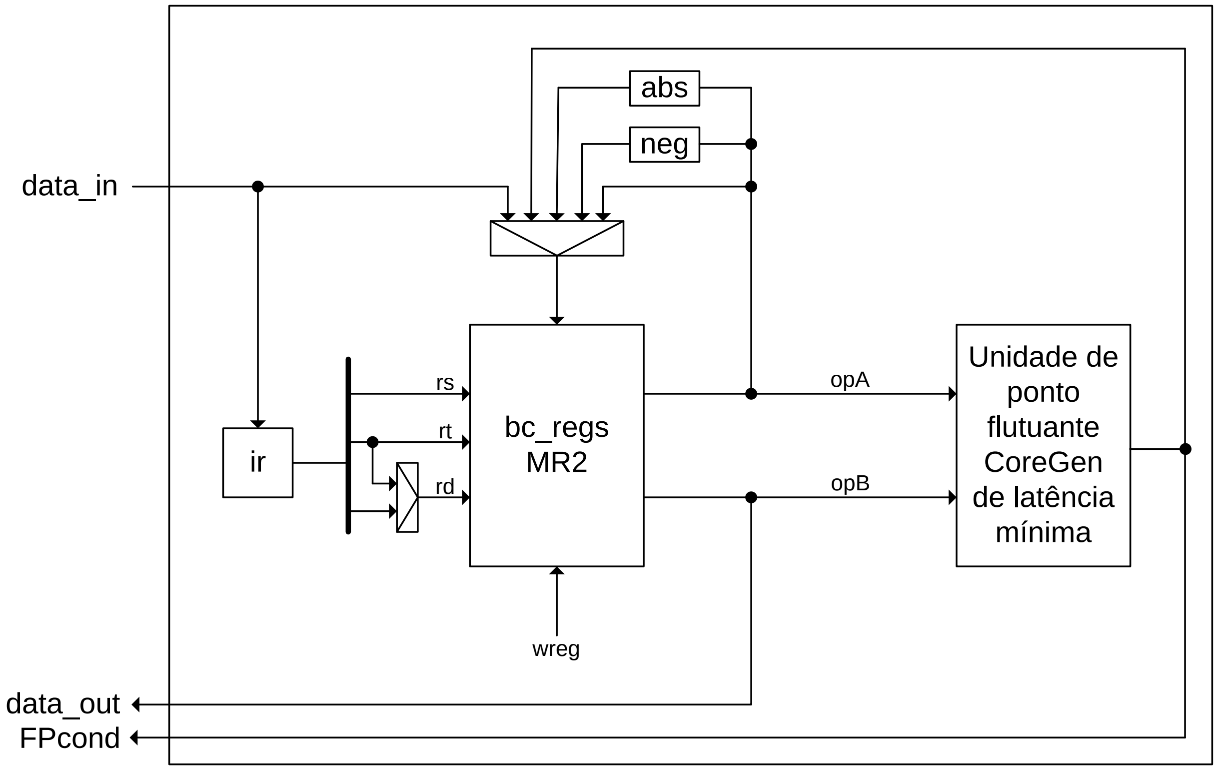

Figura 19 – Diagrama de blocos do CP1 proposto implementado com a unidade CoreGen de latência mínima.

5.4. Estimativas Iniciais de Ocupação de Área e da Frequência de Operação

Após a validação em nível de simulação das três implementações do CP1, determinaram-se suas estimativas de ocupação de área e da frequência de operação em ferramentas de síntese. Tais estimativas permitiram posteriormente definir qual das versões do CP1 seria a mais adequada para a organização denominada Plasma-HFP-GALS. Para a geração destas estimativas utilizou-se as ferramentas Xilinx ISE Design Suite 11.4 (*engine* de síntese XST) e Synplicity Synplify Premier DP C-2009.03, utilizando em ambas as ferramentas suas configurações padrões de síntese. Como dispositivo alvo empregou-se os FPGAs Virtex-4 XCV4FX100-10 e Virtex-5 XC5VLX50T-1.

A Tabela 3 lista as estimativas de ocupação de área e da frequência de operação provenientes das duas ferramentas de síntese para os três coprocessadores, para o processador *mlite* e para o banco de registradores Reg_bank_MR.

Tabela 3- Estimativas iniciais da frequência de operação e de ocupação de área para as três implementações do CP1 e seus sub-módulos. As estimativas também foram geradas para o processador mlite e para o banco de registradores Reg_bank_MR. O total de LUTs é de 84352 e 28800 para os dispositivos XC4VFX100-100 e XC5VLX50-1 respectivamente.

Coprocessador - Sub-módulo	**XST**				**Synplify**	
	XC4VFX100-10		XC5VLX50T-1		XC4VFX100-10	
	Freq. (MHz)	Area (LUTs)	Freq. (MHz)	Area (LUTs)	Freq. (MHz)	Area (LUTs)
HFP100	29,43	10348 (12%)	39,63	7184 (24%)	33,6	5399 (6%)
- FPU100	27,99	8465 (10%)	40,48	6252 (21%)	78,00	3283 (3%)
- Controle	349,53	865 (1%)	428,82	696 (1%)	179,00	1085 (1%)
HFPmin	5,85	4144 (4%)	8,64	3273 (11%)	5,70	4596 (5%)
- CoregenMin	83,99	2848 (3%)	487,80	2438 (8%)	190,00	3489 (4%)
- Controle	730,35	131 (1%)	853,84	101 (1%)	234,10	98 (1%)
HFPmax	90,92	4163 (4%)	129,62	3143 (10%)	46,6	4810 (5%)
- CoregenMax	249,83	2926 (3%)	342,35	2290 (7%)	208,90	3582 (4%)
- Controle	338.01	126 (1%)	356,51	105 (1%)	234,10	98 (1%)
Mlite	72,07	2860 (3%)	78,61	2058 (7%)	38,50	2382 (2%)
Reg_bank_MR	-	1064 (1%)	-	672 (2%)	NA	1062 (1%)

Observando-se esta Tabela conclui-se que as três unidades de ponto flutuante adotadas representam a maior parte da ocupação de área dos coprocessadores implementados, sendo pouco maiores que a área ocupada pelo processador Plasma. Além disso, percebe-se que os coprocessadores HFPmin e HFPmax tiveram suas estimativas de operação drasticamente reduzidas, o que não ocorre ao coprocessador HFP100. Não foi possível determinar a causa para esta ocorrência, porém, todas as desconfianças recaem sobre um possível caminho crítico demasiadamente longo no coprocessador.

Um fato importante a ser observado é a grande disparidade entre os valores obtidos pelas ferramentas ISE e Synplify para as sínteses dos coprocessadores utilizando o Virtex4 como FPGA alvo. Para os coprocessadores HFPmin e HFPmax observam-se valores sensivelmente melhores na ocupação de área quando se utiliza o ISE. Isto pode ser explicado pelo fato do software e das unidades CoregenMin e CoregenMax serem disponibilizados pela Xilinx, fabricante do FPGA Virtex4. Acredita-se que a ferramenta ISE pode posicionar as unidades e realizar um roteamento de uma forma mais otimizada que a ferramenta Synplify, quando são utilizados componentes disponibilizados pelo seu fabricante. Por outro lado, o Synplify apresenta valores significativamente melhores na questão da área quando o projeto utiliza somente código HDL genérico. Isto pode ser devido ao fato do Synplify ser projetado por uma empresa especializada tão somente em ferramentas de CAD, provavelmente possuindo algoritmos de posicionamento e roteamento melhores do que o ISE. Obviamente, pode-se afirmar que esta não é uma explicação plausível para o problema, porém, é importante ressaltar que se tentou de todas as formas possíveis, através de configurações e parametrizações de síntese, buscar uma redução na ocupação de área dos coprocessadores na ferramenta ISE. No entanto, não houve sucesso nesta tentativa, uma vez que as reduções obtidas foram insignificantes perto dos resultados obtidos pelo Synplify.

Outro fato que deve ser observado é a grande disparidade nas estimativas da frequência de operação, para a qual também não se tem uma explicação plausível para a causa. Uma das possíveis causas pode ser devido ao uso, pelo ISE, de multiplicadores *hard*, disponíveis no sílicio do FPGA, algo que o Synplify poderia não estar utilizando.

Ressalta-se que todas as análises realizadas sobre a Tabela 3 foram hipotéticas, sendo que até o término da escrita deste documento não se teve uma comprovação técnica para cada uma das hipóteses aventadas nesta Seção. Investigações que comprovem ou descartem estas hipóteses são relevantes e devem ser realizadas no futuro, uma vez que delas podem se ajuizar possíveis pontos de melhoria nos coprocessadores.

6. PROPOSTA DE ORGANIZAÇÕES PLASMA-HFP

Este Capítulo detalha as organizações propostas no escopo deste trabalho. Tais organizações resultam da integração dos coprocessadores do Capítulo 5 à organização Plasma original. A Seção 6.1 detalha a implementação e três versões da organização denominada Plasma-HFP. A Seção 6.2 apresenta detalhes de implementação, tais como a aplicação da técnica GALS, otimizações e modificações que resultaram na organização não síncrona denominada Plasma-HFP-GALS. A Seção 6.3 detalha a implementação da organização denominada Plasma-HFP-GALS-LP, resultado do emprego de uma técnica que visa a redução da dissipação de potência e do consumo de energia.

6.1. Organização Plasma-HFP

A proposta geral da organização denominada Plasma-HFP, aparece na Figura 20. Geraram-se três organizações pela integração de cada um dos três coprocessadores de ponto flutuante do Capítulo 5 à organização Plasma. A integração exige modificações na organização Plasma original, bem como no processador *mlite*.

Na organização Plasma original as modificações compreenderam a adição de quatro multiplexadores. Estes são controlados diretamente pelo CP1 e utilizados nas instruções de transferência de dados entre o Plasma e a memória (usado pelas instruções *sw* e *lw*, *store word* e *load word*), entre o Plasma e o CP1 (usado pelas instruções *mtc1* e *mfc1*, *move to co-processor1* e *move from co-processor1*) e entre o CP1 e a memória (usado pelas instruções *swc1* e *lwc1*, *store word co-processor1* e *load word co-processor1*). As modificações no *mlite* compreenderam a implementação das instruções *swc1*, *lwc1*, *mtc1* e *mfc1*, e também das instruções de salto condicional *bc1t* e *bc1f* (*branch if condition 1 true* e *branch if condition 1 false*, respectivamente). As instruções *swc1* e *lwc1*, baseadas nas instruções *sw* e *lw* já existentes no processador, foram implementadas no *mlite* somente para o cálculo do endereço de acesso à memória RAM. Desta forma, evita-se que o conteúdo armazenado no banco de registradores do *mlite*, exigido para este cálculo, tenha que ser transferido para o CP1, economizando ciclos de relógio e área em hardware. Não se implementou a decodificação do restante das instruções de ponto flutuante no processador *mlite*, pois isto já é efetuado pelo CP1. Adicionalmente, criaram-se no *mlite* os sinais *fetch* e *FPcond*, que serão explicados a seguir com o detalhamento da interface do CP1.

Conforme a Figura 20 ilustra, o CP1 possui em sua interface de entrada os seguintes sinais (abstraem-se aqui os sinais *clk* e *rst*): o sinal *fetch*, utilizado para diferenciar instruções de dados, uma vez que a organização Plasma utiliza a organização de memória *Von Neumann*; o sinal *data_in*, de 32 bits, para as instruções provenientes da memória RAM ou para os dados provenientes do processador *mlite* (instruções *mtc1* e *lwc1*). Sua interface de saída possui: o sinal *busy*, para indicar que o CP1 está em operação (este sinal está conectado ao sinal *mem_pause* do *mlite* e, portanto, o processador é pausado durante toda a operação do CP1); o sinal *FPcond* para indicar o resultado das operações de comparação (este sinal é utilizado pelo processador mlite nas instruções de salto condicional *bc1t* e *bc1f*); o vetor de sinais *bus_mux* (4 bits), utilizado para o controle de todos os multiplexadores (multiplexador do sinal *data_in* do CP1, multiplexadores dos sinais *data_write* e *mem_byte_en* da memória RAM e multiplexador do sinal *mem_data_r* do processador *mlite*); e por fim, o sinal de saída *data_out*, de 32 bits, utilizado para o retorno da leitura de registrador do banco de registradores do CP1 (instruções *mfc1* e *swc1*).

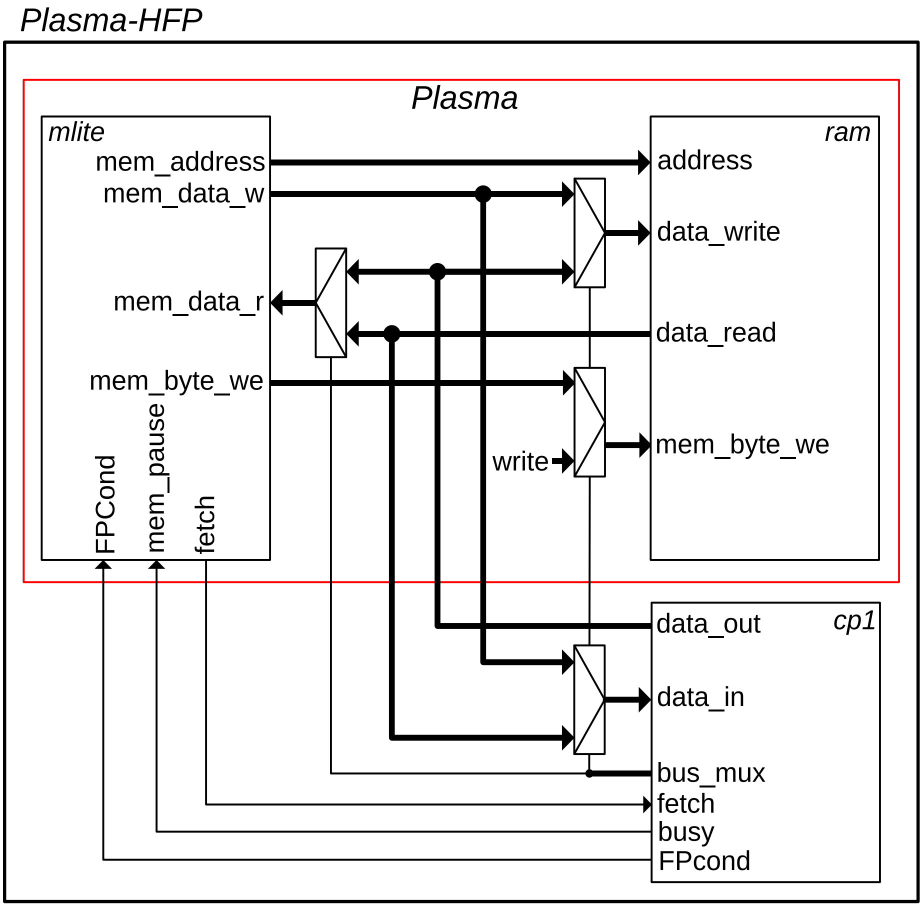

Figura 20 – Diagrama de blocos da organização Plasma-HFP proposta.

Com o objetivo de explorar o espaço de projeto do processador embarcado com coprocessador de ponto flutuante, integrou-se separadamente cada um dos três coprocessadores

Capítulo 5 à organização Plasma-HFP, resultando nas organizações Plasma-HFP100, Plasma-HFPmin e Plasma-HFPmax. As três unidades de ponto flutuante empregadas possuem a mesma interface de comunicação e a mesma funcionalidade, porém com características distintas em termos de ocupação de área, frequência máxima de operação e latência.

6.2. Organização Plasma-HFP-GALS

Sabe-se que um sistema digital totalmente síncrono é limitado pelo módulo de hardware que apresenta a menor frequência de operação. Além do mais, o acréscimo de módulos a um projeto tende a reduzir a frequência máxima do novo sistema. Desta maneira, o desempenho dos módulos de hardware com maior frequência máxima de operação acaba sendo diminuído, o que degrada o desempenho global do sistema. Para superar esta limitação, podem-se empregar técnicas de projeto para alcançar uma maior frequência de operação em alguns ou todos os módulos do sistema digital.

A implementação da organização denominada Plasma-HFP-GALS incluiu, além da aplicação da técnica GALS, a seleção do CP1 mais adequado para esta nova organização e modificações, tanto neste CP1 selecionado como também na organização Plasma-HFP, que foi utilizada como base para a implementação da organização Plasma-HFP-GALS.

A definição de qual coprocessador empregar entre os três desenvolvidos baseou-se em estimativas da frequência de operação. Dada a frequência de operação do *mlite* (78,61 MHz), optou-se por evitar coprocessadores com frequência de operação inferior a este processador. Conforme a Tabela 3 da Seção 5.4, o coprocessador HFPmax alcançou a maior estimativa da frequência de operação (129,62 MHz) e, por este motivo, foi selecionado para integrar a organização Plasma-HFP-GALS. Apesar de apresentar uma alta latência, este coprocessador recebeu otimizações que aumentaram significativamente sua frequência de operação. Operando em frequências maiores, suas instruções de ponto flutuante têm suas latências diminuídas e, assim sendo, maiores desempenhos podem ser alcançados pela organização Plasma-HFP-GALS.

Otimização do CP1 Implementado

Apesar de o coprocessador HFPmax apresentar uma frequência de operação maior que a do processador *mlite* (aproximadamente 130 MHz), foi possível ainda obter uma estimativa de frequência de operação maior, uma vez que, conforme a Tabela 3 da Seção 5.4 do Capítulo anterior, sua unidade de ponto flutuante possui uma estimativa de frequência de operação de aproximadamente 343 MHz.

Através dos relatórios de síntese, se pode analisar a organização do coprocessador HFPmax. Descobre-se então que seu caminho crítico (inicio na interface de entrada do banco de registradores e fim na interface de saída da unidade de ponto flutuante) é muito longo. Isto pode ser explicado em parte pelos grandes multiplexadores em cada saída do banco de registradores, associado ao caminho combinacional da unidade de ponto flutuante.

Para resolver este problema, inseriram-se registradores na interface de entrada da unidade de ponto flutuante. Esta solução teve um grande impacto na estimativa da frequência de operação do coprocessador, praticamente dobrando-a para aproximadamente 240 MHz. Analisando o banco de registradores, descrito em VHDL de forma genérica, percebeu-se que sua ocupação de área era em torno de dez vezes maior que a do banco de registradores utilizado pelo processador *mlite*. Verificando o banco deste processador, constatou-se que o mesmo emprega módulos de memória embarcada disponibilizados nos FPGAs da Xilinx e Altera. O projeto destes módulos de memória embarcada foram concebidos de forma otimizada, conferindo ao banco do *mlite* um desempenho maior que o banco utilizado no HFPmax. Por esta razão, adotou-se o banco de registradores do processador *mlite* no coprocessador HFPmax.

Estas duas soluções, o registro das entradas da unidade de ponto flutuante e a troca do banco de registradores, associadas às alterações realizadas na máquina de estados do módulo de controle do coprocessador, permitiram alcançar uma estimativa de 342,35 MHz, frequência de operação quatro vezes maior que a estimativa da frequência de operação do processador *mlite* (78,61 MHz). Este processo permite maximizar o desempenho da arquitetura Plasma-HFP-GALS.

Aplicação da Técnica GALS de Projeto

Conforme a Seção 0, a aplicação da técnica GALS de projeto compreende duas fases: o particionamento do sistema em sub-módulos que possuem seus próprios domínios de relógio, e a escolha da interface de comunicação assíncrona. Avaliando o processador *mlite*, verificou-se que o particionamento do mesmo resulta em complexo reprojeto que não justifica a aplicação da técnica GALS de projeto. Por esta razão, a organização Plasma-HFP-GALS, baseada na Plasma-HFP, foi particionada em dois domínios de relógio: um domínio para o processador e sua memória RAM, e outro para o coprocessador HFPmax. A interface de sincronização denominada *2FF* foi selecionada para a comunicação entre os dois domínios, uma vez que seu uso é muito simples e a que produz o menor número de alterações no hardware entre as alternativas possíveis. Na Figura 21 é possível observar o diagrama de blocos da organização Plasma-HFP-GALS proposta.

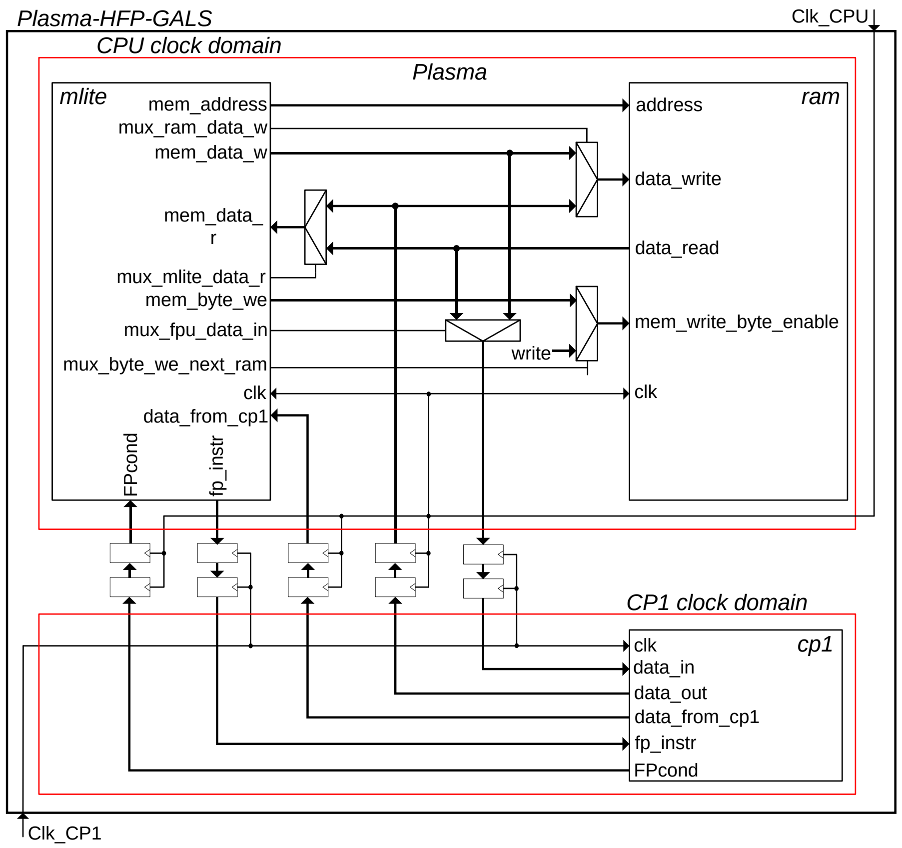

Figura 21 – Diagrama de blocos da organização Plasma-HFP-GALS proposta.

Entretanto, o emprego de técnicas GALS de projeto exige considerações de temporização específicas. Dois problemas surgem com o emprego destas. Primeiro, há o acréscimo de dois ciclos de relógio na latência dos sinais onde se inserem os registradores adicionais. Isto é problemático somente na transferência de dados de um domínio de maior frequência para um de menor frequência de operação. Segundo, existe a diferença de frequências de operação entre os diferentes domínios de relógios, o que poderia redundar em múltiplas execuções da mesma instrução pelo CP1 (de maior frequência), uma vez que o processador *mlite* (de menor frequência) gerencia a busca das instruções na memória RAM.

O primeiro problema acrescenta dois ciclos do relógio do coprocessador à latência de todos os sinais deste que interagem com o domínio do processador *mlite*: *busy*, *data_out* e sinais de controle dos multiplexadores. Assim sendo, a organização pode funcionar de modo incorreto, uma vez que os dados contidos nos referidos sinais atingem seus destinos com o atraso de dois ciclos de relógio. A Figura 22 ilustra um problema que pode ser causado nesta situação: nela o processador *mlite* é pausado pelo CP1 através do sinal *mem_pause*, que por sua vez está conectado ao sinal *busy* por meio de *2FF*; assim, o processador é pausado com o atraso de dois ciclos de relógio, ocasionando a perda de duas instruções pelo CP1, uma vez que o mlite se encontra ainda realizando buscas durante estes dois ciclos de relógio.

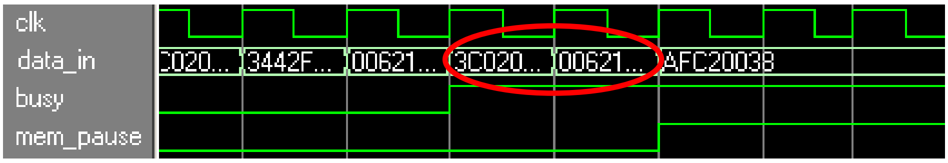

Figura 22 – Exemplo de falha causada por uma interface GALS *2FF* mal projetada. Pode ocorrer a perda de duas instruções pelo CP1.

O segundo problema pode fazer com que uma determinada instrução de ponto flutuante seja executada múltiplas vezes pelo CP1. Sendo o processador *mlite* o gerenciador da memória RAM e possuindo uma frequência de operação menor que o CP1, a instrução lida da memória permaneceria no barramento de dados por um tempo maior que o tempo de execução da instrução no CP1. Por exemplo, assuma-se que o CP1 possui uma frequência de operação de 250 MHz e o processador mlite uma frequência de 25 MHz. Instruções de baixa latência, tais como as instruções *mtc1*, *mfc1*, *lwc1* e *swc1*, que duram dois ciclos de relógio, seriam executadas indevidamente por três vezes no CP1, exatamente como se observa na Figura 23.

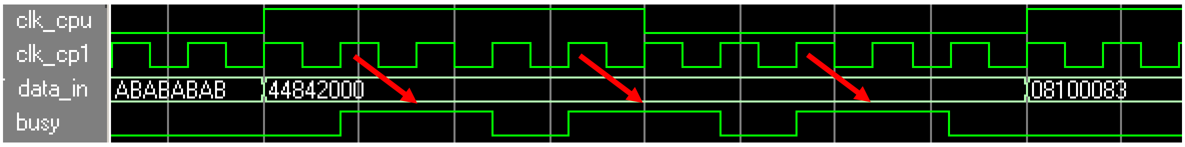

Figura 23 – Exemplo de possível falha devido ao uso de técnicas GALS de projeto para módulos com frequências de operação muito distintas: múltiplas execuções da mesma instrução de ponto flutuante pelo CP1.

Modificações no Plasma-HFP e no HFPmax

A solução para ambos os problemas está em pequenas modificações no Plasma-HFP e também no coprocessador HFPmax. A solução adotada para o primeiro problema consiste na transferência, do HFPmax para o processador *mlite*, da lógica do sinal *busy* (ver Figura 20) e dos sinais de controle dos multiplexadores (ver Figura 20 e Figura 21). Desta forma, evita-se o acréscimo de dois ciclos de relógio nestes sinais. A solução do segundo problema consiste na implementação de um novo protocolo de comunicação entre o processador *mlite* e o CP1. A implementação deste protocolo exigiu: (1) a já citada exclusão do sinal *busy*; (2) a adição dos sinais *fp_instr* e *data_from_cp1*, tanto no *mlite* como no CP1, conforme a Figura 21; (3) a implementação de uma máquina de estados no *mlite* para o controle do sinal *fp_instr*; (4) e a modificação da máquina de estados do módulo de controle do CP1 para o controle do sinal *data_from_cp1*. O sinal *fp_instr* é utilizado pelo protocolo com o seguinte propósito: em sua borda de subida, para a indicação de instrução de ponto flutuante e para notificar ao CP1 que o *mlite* foi pausado corretamente; durante o seu nível lógico '1', para permitir ao CP1 o tempo necessário para que a instrução seja executada; em sua borda de descida, para que o CP1 não execute uma determinada instrução por diversas vezes. A Figura 24 ilustra esta situação e é esclarecida a seguir.

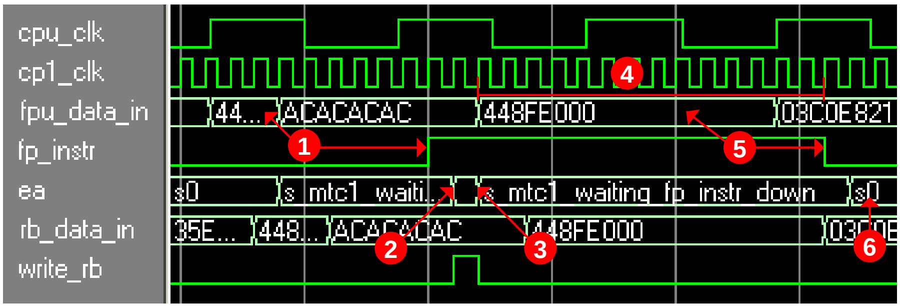

Figura 24 – Forma de onda da execução de uma instrução no CP1.

A execução de uma instrução de ponto flutuante opera da seguinte maneira: (1) o CP1 detecta a instrução *mtc1* e aguarda que o sinal *fp_instr* confirme a presença de instrução de ponto flutuante e a notificação de processador pausado; (2) o CP1 inicia a execução da instrução de ponto flutuante (o dado “ACACACAC” proveniente do mlite é armazenado no banco de registradores do CP1); (3) o CP1 termina a execução da instrução de ponto flutuante; (4) restam ainda alguns ciclos de relógio para o CP1; (5) o CP1 aguarda a borda de descida do sinal *fp_instr*, evitando que a instrução seja executada novamente; (6) o *mlite* retoma sua execução.

O número de ciclos de relógio do processador em que o sinal *fp_instr* deve permanecer em nível lógico '1', para que a instrução seja executada adequadamente, é função da latência das instruções de ponto flutuante do CP1 e da relação de frequências da organização Plasma-HFP-GALS. Entende-se “relação de frequência” como a razão entre as frequências de operação do coprocessador e do processador. No exemplo da Figura 25, para uma instrução que possua uma latência de 12 ciclos, e a organização Plasma-HFP-GALS possuindo uma relação de frequência 8 (ou seja, a frequência do CP1 oito vezes maior que a do *mlite*), são necessários no mínimo 3 ciclos do processador para que esta instrução seja devidamente executada.

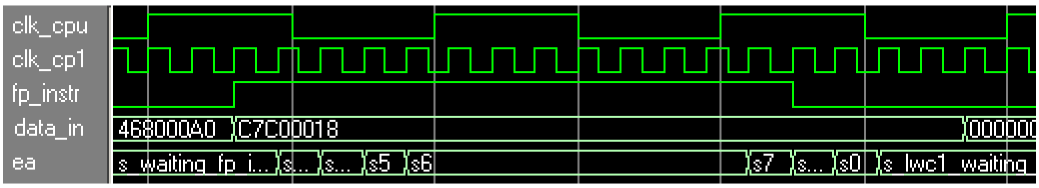

Figura 25 - Instrução de ponto flutuante de latência de 12 ciclos de relógios sendo executada em uma relação de frequência CP1/CPU = 8.

Como não foi implementada uma solução para a definição automática do número total de ciclos de relógio necessário para cada instrução de ponto flutuante, conforme as frequências empregadas, é necessário definir manualmente para todas as relações de frequência empregadas, o número total de ciclos de relógio necessário para cada instrução de ponto flutuante. Esta definição é realizada através de constantes (ver Figura 26) no código fonte do processador *mlite*, em tempo de projeto. Nesta Figura é possível verificar que instruções apresentam uma latência resultante cada vez menor conforme se aumenta a relação de frequência.

```
    --razao freq. cop1 / freq cpu = 8
    constant cvt_cycles         : std_logic_vector(5 downto 0)  := "000001";  -- 2 ciclos
    constant cmp_cycles         : std_logic_vector(5 downto 0)  := "000001";  -- 2 ciclos
    constant neg_abs_mov_cycles  : std_logic_vector(5 downto 0)  := "000000"; -- 1 ciclo
    constant add_sub_cycles   : std_logic_vector(5 downto 0)  := "000010";      -- 3 ciclos
    constant mul_cycles   : std_logic_vector(5 downto 0)  := "000010";          -- 3 ciclos
    constant div_cycles   : std_logic_vector(5 downto 0)  := "000100";          -- 5 ciclos

    --razao freq. cop1 / freq cpu = 4
--  constant cvt_cycles         : std_logic_vector(5 downto 0)  := "000100";  -- 5 ciclos
--  constant cmp_cycles         : std_logic_vector(5 downto 0)  := "000011";  -- 4 ciclos
--  constant neg_abs_mov_cycles  : std_logic_vector(5 downto 0)  := "000001"; -- 2 ciclos
--  constant add_sub_cycles   : std_logic_vector(5 downto 0)  := "000110";      -- 7 ciclos
--  constant mul_cycles   : std_logic_vector(5 downto 0)  := "000100";          -- 5 ciclos
--  constant div_cycles   : std_logic_vector(5 downto 0)  := "001010";          -- 11 ciclos

    --razao freq. cop1 / freq cpu = 2
--  constant cvt_cycles         : std_logic_vector(5 downto 0)  := "001000";  -- 9 ciclos
--  constant cmp_cycles         : std_logic_vector(5 downto 0)  := "000110";  -- 7 ciclos
--  constant neg_abs_mov_cycles  : std_logic_vector(5 downto 0)  := "000011"; -- 4 ciclos
--  constant add_sub_cycles   : std_logic_vector(5 downto 0)  := "001100";      -- 13 ciclos
--  constant mul_cycles   : std_logic_vector(5 downto 0)  := "001001";          -- 10 ciclos
--  constant div_cycles   : std_logic_vector(5 downto 0)  := "010101";          -- 22 ciclos
```

Figura 26 – Constantes que definem os números de ciclos de relógio do processador em que o sinal *fp_instr* deve permanecer em nível lógico ‘1’ para cada uma das instruções de ponto flutuante.

A definição dos números, embora seja manual, possibilita o desempenho máximo da organização Plasma-HFP-GALS. Parte de uma solução que possibilitaria uma definição automática, seria a implementação de um protocolo de comunicação do tipo *Req*(*mlite*)->*Ack*(CP1). Porém, esta alternativa acresceria à latência de todas as instruções de ponto flutuante dois ciclos de relógios, devido aos registradores de sincronização no sinal *Ack*, o que acarreta a perda de desempenho da organização Plasma-HFP-GALS.

O sinal *data_from_cp1* é utilizado pelo CP1 somente para indicar ao processador *mlite*, o instante em que o retorno de dados oriundos das instruções *mfc1* e *swc1* devem ser amostrados. Obviamente, devido ao método *2FF*, estas instruções passam a ter uma latência de quatro ciclos de relógio e não mais os dois ciclos da organização Plasma-HFP.

6.3. Organização Plasma-HFP-GALS-LP

Com o intuito de reduzir o possível aumento na dissipação de potência e no consumo de energia da organização Plasma-HFP-GALS gerado pelo coprocessador de ponto flutuante, decidiu-se pela avaliação do emprego de técnicas que visem esta redução. A nova organização foi denominada de Plasma-HFP-GALS-LP, onde LP é acrônimo para *Low Power* (baixa potência).

O requisito adotado neste trabalho dita que as técnicas empregadas devem ser prototipáveis em FPGAs da Xilinx, uma vez que este tipo de dispositivo foi utilizado para a validação das organizações e também, posteriormente, para a medição da dissipação de potência e do consumo de energia. Empregou-se a técnica de chaveamento de relógio (em inglês, *clock gating*) e a técnica de controle dinâmico de frequência (em inglês, *dynamic frequency scaling*). O diagrama de blocos da organização Plasma-HFP-GALS proposta encontra-se na Figura 27.

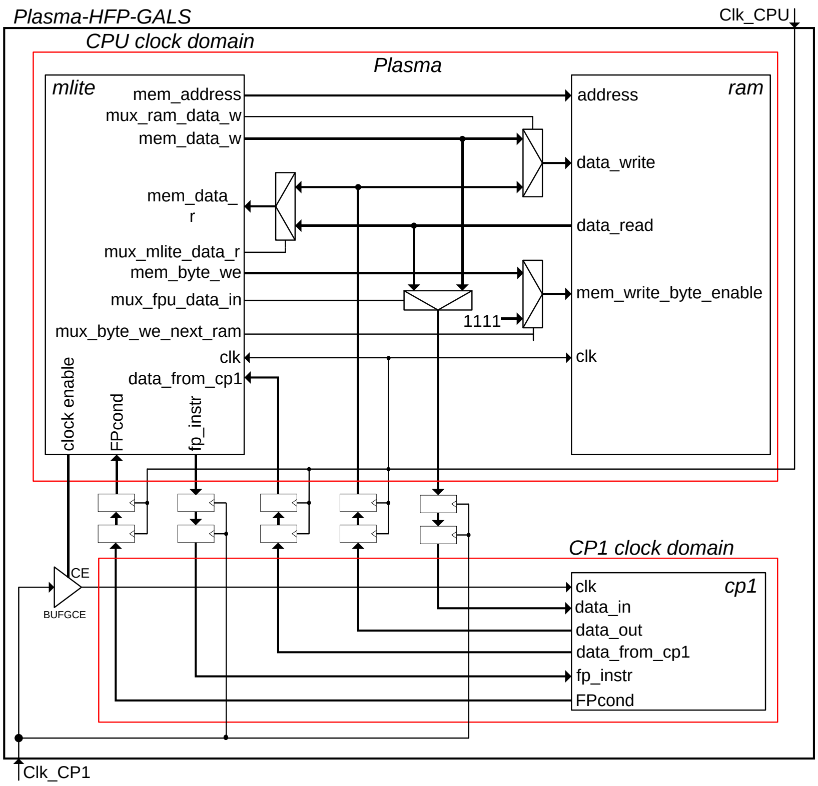

Figura 27 – Diagrama de blocos da organização Plasma-HFP-GALS-LP proposta.

Aplicação de Técnicas de Redução da Dissipação de Potência/Consumo de Energia

É importante ressaltar que inicialmente instituiu-se que as técnicas de redução da dissipação de potência e/ou consumo de energia a empregar neste trabalho seriam as técnicas de chaveamento de relógio e de controle dinâmico de frequência (DFS). Entretanto, devido a limitações explicadas a seguir, a utilização da técnica DFS foi descartada deste trabalho.

Para a implementação da técnica de chaveamento de relógio em FPGA utilizou-se componentes disponibilizados no FPGA. Os FPGAs da família Virtex-5 da Xilinx disponibilizam um componente denominado *BUFGCE* [XIL09d] dedicado à implementação da técnica de chaveamento de relógio. Com o emprego desta técnica é possível alcançar uma redução de até 30% do consumo da potência dinâmica de um circuito digital em FPGA [WAN09]. Na Figura 28 observa-se um exemplo da utilização deste componente.

O emprego da técnica DFS em FPGAs é bastante restrito, dado que os componentes destinados ao gerenciamento dos sinais de relógio em FPGAs Xilinx, tais como o *digital clock manager* (DCM) [XIL09d], ou gerenciador digital de relógio, não permitem o controle dinâmico da frequência em tempo de execução. Por este motivo e também devido ao fato da definição manual do número total de ciclos de relógio do processador em que o sinal *fp_instr* deve permanecer em nível lógico '1' para cada uma das instruções de ponto flutuante ser efetuada em tempo de projeto, não foi possível implementar a técnica de controle dinâmico de frequência em FPGA.

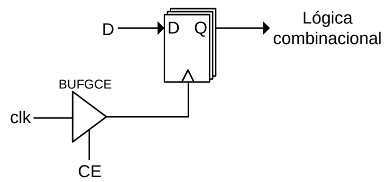

Figura 28 – Emprego do componente BUFGCE para a implementação da técnica de chaveamento de relógio em FPGAs Xilinx da família Virtex-5.

Modificações na Organização Plasma-HFP-GALS

A implementação da técnica de chaveamento de relógio exigiu a criação do sinal *clock_enable* no processador da organização Plasma-HFP-GALS, conforme mostra a Figura 27. Este sinal controla o componente *BUFGCE*, que por sua vez controla o chaveamento do sinal de relógio do coprocessador de ponto flutuante.

Para a lógica de controle do sinal *clock_enable*, foi utilizada, e modificada, a máquina de estados de controle do sinal *fp_instr*, já existente no processador. A habilitação deste sinal é função da existência, ou não, de uma instrução de ponto flutuante para a execução no CP1. A lógica de controle do sinal *clock_enable* é bastante simples. Inicialmente o sinal *clock_enable* se encontra desativado, desligando o sinal de relógio do CP1. No instante em que a máquina de estados decodifica uma instrução de ponto flutuante, o processador é pausado e o sinal *clock_enable* é ativado. O componente *BUFGCE* possui uma latência de 1 ciclo de relógio para ligar o sinal de relógio, e por este motivo, decidiu-se não utilizar um sincronizador no sinal *clock_enable*. Desta forma, o *BUFGCE* é ligado dois ciclos de relógio do processador antes que o coprocessador receba a indicação da existência de uma instrução de ponto flutuante, tempo suficiente para que este componente libere o sinal de relógio (1 ciclo de relógio do coprocessador). Com o sinal de relógio ligado, o CP1 recebe através do sinal *fp_instr* a indicação de que o *mlite* está devidamente pausado e que existe uma instrução para ser executada. Ao término da execução e após a borda de descida do sinal *fp_instr*, o sinal *clock_enable* é desativado e o sinal de relógio do CP1 torna a ser desligado.

Adicionalmente, ressalta-se que não foi analisado o caso da utilização de uma relação de frequência inversa. De fato, presume-se que a organização funciona adequadamente se o número total de ciclos de relógio do processador em que o sinal *fp_instr* deve permanecer em nível lógico '1' para cada uma das instruções de ponto flutuante for definido adequadamente conforme as novas frequências.

7. VALIDAÇÃO: SIMULAÇÃO E PROTOTIPAÇÃO

Este Capítulo reportará os meios utilizados para a validação dos diversos módulos de hardware implementados neste trabalho, desde as simulações dos coprocessadores e organizações implementadas, detalhadas na Seção 7.1, até as prototipações em FPGAs dos mesmos, detalhadas na Seção 7.2.

7.1. Simulação

Primeiramente validaram-se as unidades de ponto flutuante que foram posteriormente empregadas na implementação dos coprocessadores compatíveis com a organização MIPS-I. Esta validação ocorreu por meio da estrutura de simulação que acompanha a distribuição da unidade FPU100 [OPE08b], que além de validar esta unidade também foi utilizada para a validação das unidades HFPmin e HFPmax, geradas com o auxílio do CoreGen.

Após a implementação dos coprocessadores, validaram-se os mesmos em nível de simulação. Esta simulação teve por objetivo a verificação da funcionalidade das instruções de ponto flutuante lógico-aritmético, bem como das instruções de transferência de dados entre processador, coprocessador e memória RAM. Para realizar esta verificação criou-se um *testbench*, que consiste de uma máquina de estados capaz de interagir com interface de dados do coprocessador. Esta máquina de estados efetua a leitura de instruções de ponto flutuante, em código de máquina, existentes em um arquivo de teste, de extensão “hex”, e as envia ao coprocessador sob teste. Este arquivo, criado com o auxílio do simulador do conjunto de instruções da MIPS-I denominado MARS [MIS08], contém todas as instruções de ponto flutuante previstas pela organização MIPS-I. A validação ocorreu assegurando-se o correto funcionamento das instruções de ponto flutuante nos sinais internos dos coprocessadores, que são visualizados nas formas de ondas geradas pela ferramenta de simulação Modelsim SE 6.1f.

Procurando obter uma simulação mais realista, optou-se pelo modelo de simulação *timed*, que permite agregar às simulações os tempos de atraso dos componentes que compõem o substrato tecnológico do FPGA. Este modelo é gerado por meio da ferramenta de síntese lógica e física, após a etapa de “posicionamento e roteamento”, e consiste basicamente de dois arquivos: um arquivo VHDL *netlist*, contendo a implementação equivalente do coprocessador em componentes arquiteturais do FPGA, e um arquivo, de extensão “sdf” (*Standard Delay Format*), contendo os tempos de atraso destes componentes. O *testbench* criado, adicionado destes dois arquivos, forma o plano de fundo que permite uma simulação precisa dos coprocessadores. Na Figura 29 observa-se a estrutura criada para a validação dos coprocessadores implementados pelo Autor.

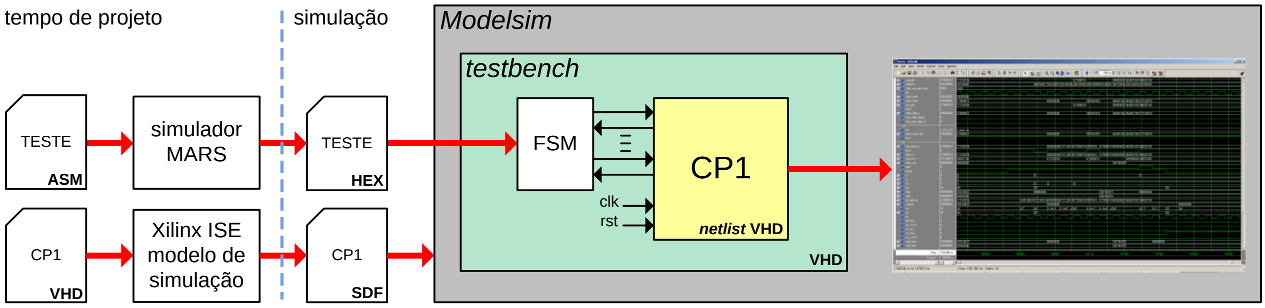

Figura 29 – Estrutura criada para a simulação dos coprocessadores.

Em seguida, ocorreu a validação em nível de simulação das organizações resultantes das integrações dos coprocessadores validados à organização Plasma. Para esta validação criou-se um programa de teste, desenvolvido em linguagem C, em substituição ao arquivo criado com o auxilio do MARS, utilizado anteriormente na validação dos coprocessadores. Este programa foi criado com o intuito de verificar a interação entre processador e coprocessador, bem como de testar todas as instruções de ponto flutuante previstas pela MIPS-I, incluindo todas as possíveis variações de cada uma destas instruções. O arquivo contendo o código fonte deste programa pode ser visto no Apêndice A.

O programa de teste é executado pelo processador *mlite* e, para tanto, deve ser carregado na memória RAM da organização Plasma. Isto é feito utilizando-se o software *ram_image* [OPE08b], disponibilizado juntamente com a distribuição Plasma, que permite a cópia de uma imagem do código de máquina de um determinado programa para a memória RAM da organização Plasma. Este software recebe como entrada dois arquivos: o arquivo responsável por implementar a memória RAM da organização Plasma (ram_xilinx.vhd) e o arquivo contendo o código de máquina de um determinado programa (prog_teste.hex). Este código de máquina é obtido compilando-se este programa no compilador GNU GCCelfmips [OPE08b], que acompanha a distribuição Plasma. O *ram_image* gera como saída um novo arquivo denominado “ram_image.vhd” contendo todas as instruções existentes no arquivo “prog_teste.hex”. A Figura 30 ilustra o fluxo utilizado para carregar programas na memória de instruções da organização Plasma.

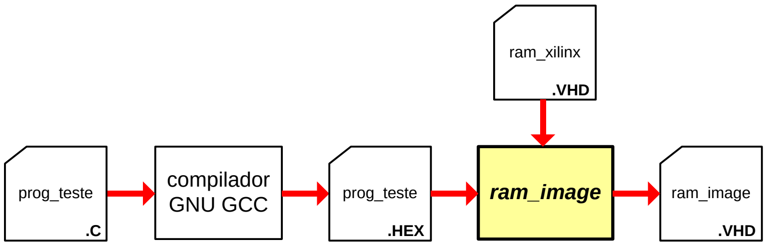

Figura 30 – Fluxo utilizado para carregar programas na memória de instruções da organização Plasma.

O arquivo "ram_image.vhd" gerado a partir do programa de teste deve substituir o arquivo "ram_xilinx.vhd" na síntese das organizações a serem testadas, para a geração do modelo de simulação *timed*. A simulação destas organizações utilizou a mesma estrutura de simulação utilizada para a validação dos coprocessadores, porem, algumas modificações foram realizadas para se adequar a validação de tais organizações. Como o programa de teste já se encontra armazenado na memória RAM da organização, não é necessário anexar à simulação um arquivo "hex" de teste, como ocorre na validação dos coprocessadores. Portanto, uma máquina de estados para realizar a leitura deste arquivo não se faz necessária. Por esta razão, tal máquina foi excluída do *testbench*. Outra alteração diz respeito à interface de dados das organizações implementadas. Como o programa de teste, que se encontra na memória RAM, não faz uso de memória externa, a interface de dados existente para este fim não se justifica. Por este motivo, as organizações foram encapsuladas em um par entidade/arquitetura denominado *top*, que possui apenas duas portas de entrada: o sinal de relógio, ***clk***, e o sinal de reset, ***rst***. Na Figura 31 observam-se as alterações realizadas na estrutura criada para a validação dos coprocessadores, que redundaram em uma estrutura para a validação das organizações implementadas.

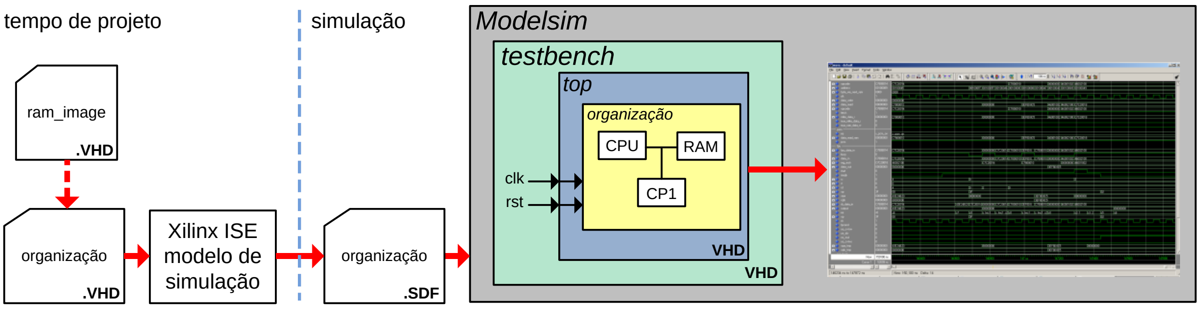

Figura 31 – Estrutura de simulação criada para a validação das organizações implementadas.

A validação também ocorreu assegurando-se o correto funcionamento das instruções (principalmente as que transferem dados entre processador, coprocessador e memória RAM) nos sinais internos das organizações, que são visualizados através das formas de ondas geradas pela ferramenta de simulação.

7.2. Prototipação

Após a validação de todos os coprocessadores em nível de simulação, validaram-se os mesmos em nível de prototipação. Inicialmente prototipou-se tais coprocessadores utilizando o FPGA Xilinx Virtex-4 XCV4FX100-10 existente na plataforma DN8000K10PCI da Dinigroup. Posteriormente, quando se adquiriu a plataforma de modelo ML550 da Xilinx, prototipou-se utilizando o FPGA Virtex-5 XC5VLX50T-1. Para a prototipação dos coprocessadores criou-se uma estrutura similar ao *testbench* utilizado em sua simulação. Nesta estrutura criou-se um vetor de instruções contendo as instruções de ponto flutuante existentes no arquivo "hex" de teste gerado com o auxílio do MARS. Uma máquina de estados efetua a leitura das instruções neste vetor e as envia, uma a uma, ao coprocessador prototipado. Para validá-lo, assegura-se o correto funcionamento das instruções de ponto flutuante em seus sinais internos, que são visualizados através das formas de ondas geradas pela ferramenta ChipScope Pro. Esta ferramenta permite que se observem os sinais internos dos módulos de hardware prototipados no FPGA em tempo de execução. A Figura 32 apresenta a estrutura de prototipação criada para a validação dos coprocessadores implementados.

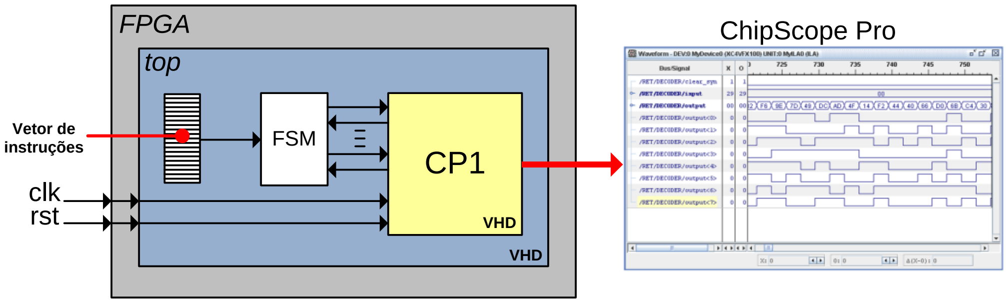

Figura 32 – Validação dos coprocessadores implementados.

Para a validação em nível de prototipação das organizações resultantes das integrações dos coprocessadores implementados à organização Plasma, utilizou-se a mesma abordagem empregada em suas simulações. Ou seja, prototipou-se no FPGA as organizações encapsuladas no *top* somente com os sinais ***clk*** e ***rst***, e com o arquivo "ram_image.vhd", gerado pelo *ram_image* a partir do programa de teste desenvolvido em linguagem C. A validação ocorreu assegurando-se o correto funcionamento das instruções (principalmente as que transferem dados entre processador, coprocessador e memória RAM) nos sinais internos das organizações, que também são visualizados através das formas de ondas geradas pela ferramenta ChipScope Pro. A Figura 33 apresenta a abordagem utilizada para a validação das organizações implementadas neste trabalho.

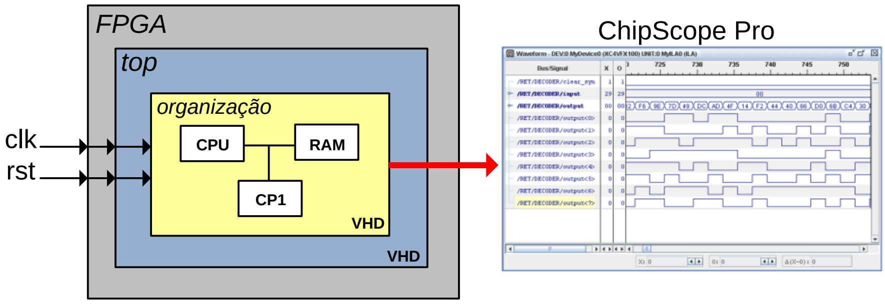

Figura 33 – Validação das organizações implementadas.

8. RESULTADOS EXPERIMENTAIS

Este Capítulo apresenta os resultados obtidos na pesquisa proposta pelo Autor. A Seção 8.1 apresenta as diferentes latências das instruções de ponto flutuante nas organizações implementadas, enquanto a Seção 8.2 expõe estimativas de ocupação de área e da frequência de operação destas organizações, dados oriundos das ferramentas de síntese lógica e física empregadas. O desempenho das organizações é apresentado na Seção 8.3 e suas respectivas densidades computacionais na Seção 8.4. Finalizando o Capítulo e completando os dados de exploração do espaço de projeto, a Seção 8.5 apresenta as medições da dissipação de potência e do consumo de energia das organizações executando diferentes aplicações intensivas em instruções de ponto flutuante.

8.1. Latência das Instruções de Ponto Flutuante

As organizações construídas possuem diferentes latências para as mesmas instruções de ponto flutuante, uma vez que os coprocessadores empregados possuem diferentes latências de computação para estas. Tal latência é resultado da soma de latências da instrução em questão do coprocessador de ponto flutuante particular e da latência das operações de controle do mesmo coprocessador para esta instrução. A Tabela 4 lista as latências de todas as instruções de ponto flutuante nas organizações Plasma-HFP100, Plasma-HFPmax, Plasma-HFPmin e Plasma-HFP-GALS. Para esta última adotam-se três relações de frequência entre o coprocessador e o Plasma (2, 4 e 8).

Tabela 4 – Latência das instruções de ponto flutuante, em ciclos de relógio, para as organizações construídas.

Instrução	Plasma-HFP100	Plasma-HFPmin	Plasma-HFPmax	Plasma-HFP-GALS 2x	Plasma-HFP-GALS 4x	Plasma-HFP-GALS 8x
Add	12	4	19	13	7	3
Sub	12	4	19	13	7	3
Mul	17	4	14	10	5	3
Div	39	4	34	22	11	5
Abs	3	3	3	4	2	1
Neg	3	3	3	4	2	1
Cvt	6	4	12	10	5	2
Cmp	12	4	9	9	4	2
Mov	3	3	3	4	2	1
mtc1	2	2	2	2	2	2
mfc1	2	2	2	4	4	4
lwc1	2	2	2	2	2	2
swc1	2	2	2	4	4	4

Na Tabela 5 encontram-se as latências das instruções de ponto flutuante emuladas em software. Somente as latências das instruções de adição, subtração, multiplicação e divisão foram consideradas, uma vez que tais instruções são as mais relevantes e as que possuem as maiores latências dentre as demais, supostamente menores.

Tabela 5 – Latência das instruções de ponto flutuante emuladas em software, em ciclos de relógio.

Instrução	Emulação
Add	227
Sub	278
Mul	165
Div	297

8.2. Estimativas de Ocupação de Área e da Frequência de Operação

Para a geração de estimativas de ocupação de área e da frequência de operação utilizou-se o ambiente Xilinx ISE Design Suite 11.3 (ferramenta de síntese XST). Em função das disponibilidades de plataformas de prototipação, selecionaram-se como dispositivos-alvo os FPGAs Virtex-4 XCV4FX100-10 e Virtex-5 XC5VLX50T-1. A Tabela 6 lista as estimativas para as cinco organizações, acrescidas das estimativas para a organização Plasma. Nota-se que a capacidade do dispositivo Virtex-5 é aproximadamente a metade da capacidade do dispositivo Virtex-4.

Tabela 6- Estimativas da frequência de operação e de ocupação de área para as cinco organizações construídas e para a organização Plasma. O total de LUTs para os dispositivos XC4VFX100-100 e XC5VLX50-1 é de respectivamente 84352 e 28800.

Organização	XST			
	XC4VFX100-10		XC5VLX50T-1	
	Freq. (MHz)	Área (LUTs)	Freq. (MHz)	Área (LUTs)
Plasma	65,97	3369 (3%)	78,05	2222 (7%)
Plasma-HFP100	15,31	11489 (13%)	24,34	8320 (28%)
Plasma-HFPmin	5,85	7169 (8%)	8,64	5453 (18%)
Plasma-HFPmax	60,46	7516 (8%)	77,33	5504 (19%)
Plasma-HFP-GALS	65,49	7869 (9%)	79,09	5647 (19%)
Plasma-HFP-GALS-LP	66,08	7675 (9%)	79.60	5534 (19%)

É importante destacar que a Tabela 3, apresentada na Seção 5.4, possui somente as estimativas de ocupação de área e da frequência de operação para os coprocessadores implementados no contexto deste trabalho, que agora constituem as organizações que compõem as estimativas apresentadas na Tabela 6.

8.3. Desempenho das Organizações

Para avaliar o desempenho das organizações, desenvolveram-se quatro aplicações em linguagem de programação C: as funções seno e cosseno e filtros FIR e IIR. Os Apêndices B e C deste documento contêm o código fonte das aplicações, escritos em linguagem C. As duas primeiras aplicações baseiam-se em expansões da série de Taylor das funções seno e cosseno, sendo ambas implementadas para o cálculo dos dez primeiros termos de cada série. As duas últimas aplicações são filtros do tipo passa-baixas, criados com quatro coeficientes e com frequência de corte definida em 4 KHz. As quatro aplicações foram executadas nas organizações:

1. Plasma sem CP1, emulando instruções de ponto flutuante em software.
2. Plasma-HFP100
3. Plasma-HFPmin
4. Plasma-HFPmax
5. Plasma-HFP-GALS operando em três relações de frequência: 2, 4 e 8.

Para avaliar o desempenho das aplicações em cada organização, primeiramente determinou-se o número total de ciclos de relógio necessários para a execução de cada uma destas aplicações. O número total de ciclos para o seno e cosseno é determinado através do instante em que é realizada a leitura do operando da memória RAM até o instante em que o resultado final é armazenado na memória RAM. Para os filtros FIR e IIR este número é definido pelo total de ciclos de relógio necessários para o cálculo de uma amostragem. Ou seja, do instante em que se realiza a leitura do dado amostrado na RAM até o instante em que o resultado do cálculo efetuado sobre esta amostragem é armazenado na RAM. Tais números foram obtidos nas simulações das organizações através da implementação de um contador de instruções. Para determinar o início e o fim desta contagem, utilizou-se o mesmo expediente empregado na implementação da janela de amostragem, detalhada na Seção 0: a leitura dos valores “1BABABAB” e “1CACACAC”, respectivamente, no barramento de dados das organizações. Na Tabela 7 se encontra o número total de ciclos de relógio para as quatro aplicações.

Tabela 7 – Número de ciclos de relógio para as aplicações empregadas.

Organização	sen	cos	FIR	IIR
Emulação	89898	81906	2662	3776
Plasma-HFP100	11648	10739	867	367
Plasma-HFPMín	8568	7911	762	186
Plasma-HFPMáx	11638	10729	884	396
Plasma-HFP-GALS 2x	11366	10472	872	303
Plasma-HFP-GALS 4x	9381	8671	805	224
Plasma-HFP-GALS 8x	8722	8065	775	172

Observando a Tabela, percebe-se expressiva discrepância nos números de ciclos de relógio necessários para a emulação dos filtros FIR e IIR. Investigando tal ocorrência, determinou-se inicialmente o número total de instruções de ponto flutuante existentes em cada uma destas aplicações. A Tabela 8 apresenta tais números, incluindo também os dados de seno e cosseno.

Tabela 8 – Número de instruções de ponto flutuante executado por aplicação.

sen	cos	FIR	IIR
1124	1043	37	26

Em seguida, analisou-se o código fonte de ambos os filtros com o objetivo de determinar o tipo de instruções de ponto flutuante empregadas em cada uma destas duas aplicações. Verificou-se então, que o filtro FIR possui 5 adições e 5 multiplicações de ponto flutuante, enquanto que o filtro IIR possui 8 adições e 9 multiplicações. Em ambos os filtros, o restante das instruções corresponde, em sua grande maioria, a instruções de acesso à RAM.

Tabela 9 – Tempo de execução, em milissegundos, para as quatro aplicações desenvolvidas.

Organização	Frequência de operação (MHz)	seno	cosseno	FIR	IIR
Emulação	50	1,79796	1,63812	0,05324	0,07552
Plasma-HFP100	25	0,46592	0,42956	0,03468	0,01468
Plasma-HFPMín	5	1,71360	1,58220	0,15240	0,03720
Plasma-HFPMáx	50	0,23276	0,21458	0,01768	0,00792
Plasma-HFP-GALS 2x	CPU 50/CP1 100	0,22732	0,20944	0,01744	0,00606
Plasma-HFP-GALS 4x	CPU 50/CP1 200	0,18762	0,17342	0,01610	0,00448
Plasma-HFP-GALS 8x	CPU 50/CP1 400	0,17444	0,16130	0,01550	0,00344

O filtro IIR apresenta um número significativamente maior de instruções aritméticas em relação às demais instruções de ponto flutuante, não obstante possuir um número menor de instruções de ponto flutuante. Por este motivo e devido à latência das instruções aritméticas empregadas (Tabela 5), a emulação do filtro IIR demanda um número de ciclos de relógio maior que a emulação do filtro FIR. Nas organizações com CP1 isto não ocorre devido à enorme redução de latência das instruções aritméticas de ponto flutuante (Tabela 4).

Tabela 10 – Acelerações alcançadas nas aplicações.

Aceleração da aplicação na organização - em relação à organização	seno	cosseno	FIR	IIR
Plasma-HFP100				
- Emulação	7,72x	7,63x	3,07x	10,29x
Plasma-HFPMín				
- Emulação	10,49x	10,35x	3,49x	20,30x
- Plasma-HFP100	1,36x	1,36x	1,14x	1,97x
Plasma-HFPMáx				
- Emulação	7,72x	7,63x	3,01x	9,53x
- Plasma-HFP100	1,00x	1,00x	0,98x	0,93x
- Plasma-HFPMín	0,74x	0,74x	0,86x	0,47x
Plasma-HFP-GALS 2x				
- Emulação	7,91x	7,82x	3,05x	12,46x
- Plasma-HFP100	1,02x	1,02x	0,99x	1,21x
- Plasma-HFPMín	0,75x	0,75x	0,87x	0,61x
- Plasma-HFPMáx	1,02x	1,02x	1,01x	1,31x
Plasma-HFP-GALS 4x				
- Emulação	9,58x	9,44x	3,31x	16,86x
- Plasma-HFP100	1,24x	1,24x	1,08x	1,64x
- Plasma-HFPMín	0,91x	0,91x	0,95x	0,83x
- Plasma-HFPMáx	1,24x	1,24x	1,10x	1,77x
- Plasma-HFP-GALS 2x	1,21x	1,21x	1,08x	1,35x
Plasma-HFP-GALS 8x				
- Emulação	10,31x	10,15x	3,43x	21,95x
- Plasma-HFP100	1,33x	1,33x	1,12x	2,13x
- Plasma-HFPMín	0,98x	0,98x	0,98x	1,08x
- Plasma-HFPMáx	1,33x	1,33x	1,14x	2,30x
- Plasma-HFP-GALS 2x	1,30x	1,30x	1,12x	1,76x
- Plasma-HFP-GALS 4x	1,07x	1,07x	1,04x	1,30x

De posse do número total de ciclos de relógio necessário para a execução de cada aplicação (Tabela 7) e atribuindo-se uma frequência de operação para cada uma das organizações é possível derivar o tempo de execução destas aplicações. As frequências de operação das organizações são atribuídas levando-se em conta suas estimativas de frequência de operação (Tabela 6) e, principalmente, a limitação física dos osciladores disponíveis na plataforma ML550. Tais tempos de execução encontram-se na Tabela 9 abaixo.

De posse da Tabela 9, calcula-se as acelerações alcançadas pelas quatro organizações, nas quatro aplicações desenvolvidas. A Tabela 10 apresenta tais acelerações, calculadas em relação à emulação das instruções de ponto flutuante e em relação às organizações entre si.

8.4. Densidade Computacional

Brunelli et al. [BRU05] propuseram o conceito de densidade computacional de uma implementação de hardware em FPGA. Este cálculo pode ser feito usando a Equação 14.

$$densidade = \frac{f_{relógio}}{ciclos \cdot comp} \quad (14)$$

Nesta Equação, $f_{relógio}$ é a frequência de operação do sistema computacional, $ciclos$ é o número de ciclos de relógio necessário para a execução de uma determinada aplicação neste sistema, e $comp$ é o número total de LUTs necessário para a implementação do sistema computacional em FPGA. Os resultados da Tabela 11 foram gerados para o FPGA Xilinx Virtex-5 XC5VLX50T-1, empregando-se a ferramenta XST para síntese lógica. A Tabela 11 apresenta o cálculo da densidade computacional para as quatro aplicações em todas as organizações, sendo que um maior valor de densidade denota um melhor compromisso entre área em hardware e desempenho.

Tabela 11 – Comparação das densidades computacionais para as quatro aplicações em todas as organizações.

Organização	Frequência de operação (MHz)	seno	cosseno	FIR	IIR
Emulação	50	0,2503	0,2747	8,4531	5,9593
Plasma-HFP100	25	0,2579	0,2798	3,4657	8,1875
Plasma-HFPMín	5	0,1070	0,1159	1,2033	4,9297
Plasma-HFPMáx	50	0,7806	0,8467	10,2763	22,9401
Plasma-HFP-GALS 2x	CPU 50/CP1 100	0,7790	0,8455	10,1540	29,2220
Plasma-HFP-GALS 4x	CPU 50/CP1 200	0,9438	1,0211	10,9991	39,5279
Plasma-HFP-GALS 8x	CPU 50/CP1 400	1,0152	1,0979	11,4248	51,4782

8.5. Medição da Dissipação de Potência e do Consumo de Energia

Seguindo a técnica descrita na Seção 4.4 foi possível medir as tensões sobre o resistor de precisão existente na plataforma ML550 durante a execução das quatro aplicações nas organizações prototipadas no FPGA XC5VLX50T-1. As aplicações são carregadas na memória RAM das organizações em tempo de projeto. Assim, quatro sínteses de uma mesma organização seriam necessárias. Como se sabe, as etapas de posicionamento e roteamento da síntese física não são determinísticas e, por este motivo, o hardware gerado pode diferir de uma síntese para outra. Portanto, é necessário que se utilize somente uma síntese para cada organização, a fim de se evitar discrepâncias nos cálculos posteriores. Deve-se então gerar uma solução que permita a alteração das aplicações nas organizações em tempo de execução, sem realizar novamente a síntese. A Figura 34 ilustra a solução empregada neste trabalho.

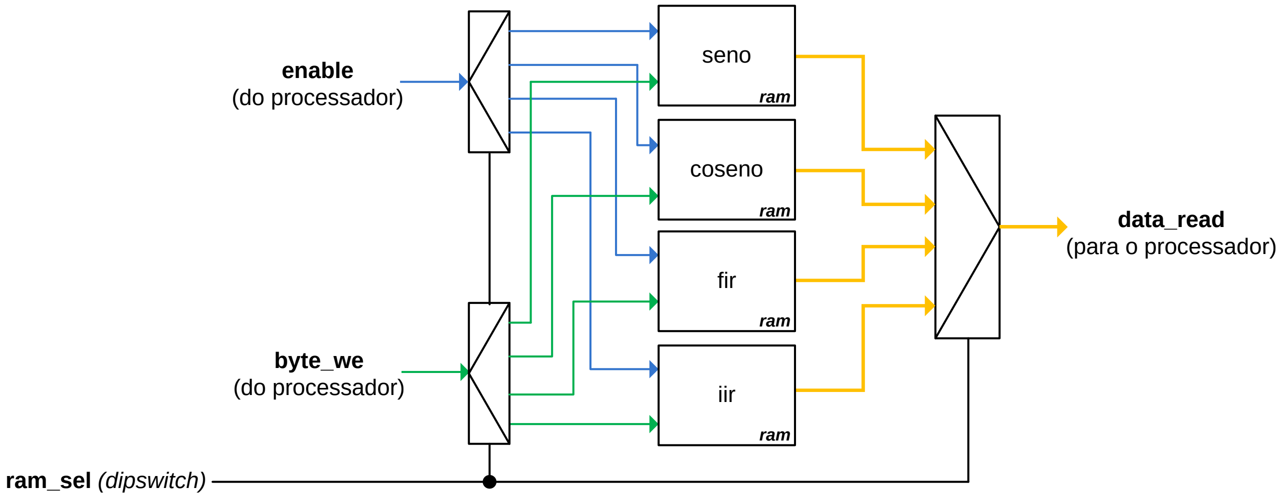

Figura 34 – Estrutura criada para a seleção das aplicações em tempo de execução nas organizações. As demais portas das interfaces de dados foram abstraídas da Figura, a fim de facilitar a visualização.

A solução consiste na implementação de uma estrutura que compreende: quatro memórias RAM, cada uma contendo uma das quatro aplicações; a inclusão de um multiplexador no ***data_read***, sinal de retorno das instruções de leitura da memória RAM; e a inclusão de dois demultiplexadores nos sinais ***enable*** e ***byte_we***, ambos controlados pelo processador e responsáveis pelo controle da memória RAM. O restante dos sinais da interface de dados, ***data_write*** e ***address*** são sinais de entrada em comum para todas as memórias RAMs e, por esta razão, não necessitam de demultiplexadores. A seleção das aplicações é realizada, em tempo de execução, através de uma micro-chave externa existente na plataforma ML550.

Para permitir uma análise do que seria a utilização da técnica DFS, não implementada neste trabalho, igualmente se faz necessário alterar a frequência de operação das organizações em tempo de execução. De forma análoga à solução anterior, a seleção da frequência de operação é realizada também através de uma micro-chave existente na plataforma ML550. Através deste dispositivo é possível selecionar as frequências existentes dentro do espectro definido pelo Autor: ***1***, ***2,5***, ***5***, ***10***, ***25*** e ***50*** MHz. Estas frequências são geradas a partir da divisão de um sinal de relógio de 200 MHz, proveniente de oscilador externo e devidamente tratado por DCM. Definidos os métodos para a seleção das aplicações e das frequências de operação, mediram-se as tensões sobre o resistor de precisão em todas as organizações, executando as quatro aplicações separadamente nas frequências de operação definidas. Esta medição permitiu, conforme a Seção 4.4, o cálculo da dissipação da potência instantânea do FPGA, e a partir desta derivaram-se outros cálculos: potência média dissipada e energia consumida pelo FPGA durante a execução das aplicações, bem como o pico de potência instantânea. As Tabelas 12 a 15 apresentam os resultados obtidos, respectivamente para as aplicações seno, cosseno, FIR e IIR. Ressalta-se ainda que nestas Tabelas o tempo de execução das aplicações foi medido diretamente no osciloscópio, através da janela de amostragem.

Tabela 12 – Resultados obtidos para o seno.

Organização	Tempo de execução (ms)	Potência média dissipada (W)	Energia consumida (mJ)	Pico de potência instantâneo (W)
Emulação	18,0107	15,9861	287,9212	16,5729
Plasma-HFP100	2,3378	15,8258	36,9973	18,4170
Plasma-HFPMín	1,7218	15,8014	27,2067	18,4300
Plasma-HFPMáx	2,3918	15,9674	38,1908	18,6540
Plasma-HFP-GALS 2x	2,2738	15,9490	36,2647	18,6010
Plasma-HFP-GALS 4x	1,9028	15,5644	29,6158	18,3120
Plasma-HFP-GALS 8x	1,7438	15,1838	26,4773	17,9430
Plasma-HFP-GALS-LP	1,7438	15,4122	26,8758	18,1009

Tabela 13 – Resultados obtidos para o cosseno.

Organização	Tempo de execução (ms)	Potência média dissipada (mW)	Energia consumida (mJ)	Pico de potência instantâneo (W)
Emulação	16,3965	16,0204	262,6784	16,6250
Plasma-HFP100	2,1560	15,6239	33,6849	18,1140
Plasma-HFPMín	1,5904	15,9567	25,3774	18,5620
Plasma-HFPMáx	2,2046	15,9863	35,2432	18,6410
Plasma-HFP-GALS 2x	2,1010	16,0366	33,6926	18,5490
Plasma-HFP-GALS 4x	1,7588	15,6293	27,4887	18,4830
Plasma-HFP-GALS 8x	1,6124	15,5502	25,0730	18,3509
Plasma-HFP-GALS-LP	1,6124	15,6297	25,2013	18,4399

Tabela 14 – Resultados obtidos para o filtro FIR.

Organização	Tempo de execução (ms)	Potência média dissipada (W)	Energia consumida (mJ)	Pico de potência instantâneo (W)
Emulação	0,5360	15,4550	8,2838	18,3379
Plasma-HFP100	0,1732	16,1164	2,7913	19,3390
Plasma-HFPMín	0,1522	16,1010	2,4505	19,1810
Plasma-HFPMáx	0,1772	16,1686	2,8650	19,3920
Plasma-HFP-GALS 2x	0,1744	16,1457	2,8158	19,4180
Plasma-HFP-GALS 4x	0,1610	16,1259	2,5962	19,2339
Plasma-HFP-GALS 8x	0,1541	15,9997	2,4799	18,9440
Plasma-HFP-GALS-LP	0,1541	16,0883	2,4792	19,1019

Tabela 15 – Resultados obtidos para o filtro IIR.

Organização	Tempo de execução (ms)	Potência média dissipada (W)	Energia consumida (mJ)	Pico de potência instantâneo (W)
Emulação	0,6631	15,3974	10,2115	18,2850
Plasma-HFP100	0,0696	15,8057	1,1001	18,8379
Plasma-HFPMín	0,0356	15,5811	0,5609	18,7329
Plasma-HFPMáx	0,0756	15,7382	1,1961	18,7590
Plasma-HFP-GALS 2x	0,0618	15,4517	0,9549	18,5489
Plasma-HFP-GALS 4x	0,0436	15,5743	0,6790	18,6800
Plasma-HFP-GALS 8x	0,0340	15,3547	0,5220	18,4169
Plasma-HFP-GALS-LP	0,0340	15,4667	0,5259	18,5640

9. CONCLUSÕES E TRABALHOS FUTUROS

Esta Dissertação teve como objetivo maior a avaliação do impacto da adoção de um coprocessador de ponto flutuante em um processador embarcado. Para alcançar este intento empregou-se a organização Plasma e implementou-se diversos coprocessadores de ponto flutuante distintos. A integração destes coprocessadores ao Plasma resultou em diversas organizações. Além disso, implementou-se uma organização não-síncrona, operando em três relações de frequência (2, 4 e 8), e uma organização não-síncrona dotada de uma técnica de redução da dissipação de potência. Estas implementações permitiram explorar o espaço de projeto de um processador embarcado capaz de executar aplicações que fazem uso de números em ponto flutuante.

9.1. Conclusões

Através dos resultados obtidos neste trabalho, conclui-se que a adoção de um coprocessador de ponto flutuante deve depender fortemente da quantidade de cálculos em ponto flutuante da aplicação alvo do sistema embarcado que se pretende projetar. As restrições de projeto também devem ser consideradas, uma vez que estas informações irão definir qual organização, dentro do espaço de projeto, permitirá a obtenção do melhor compromisso entre os requisitos de desempenho, ocupação de área em silício e consumo de potência.

A frase "*quantidade maciça de cálculos em ponto flutuante*", muitas vezes repetida no decorrer deste trabalho, não deve ser traduzida como o número absoluto de instruções aritméticas de ponto flutuante existentes em uma aplicação. Na realidade ela representa a proporção do total de instruções aritméticas de ponto flutuante perante o total de instruções lógico-aritméticas utilizadas para o controle do fluxo de execução da aplicação. Quanto maior for esta proporção, ou seja, quanto maior for o número de instruções de ponto flutuante em relação ao número das demais instruções lógico-aritmética, maior será o ganho em desempenho. Quanto menor for esta proporção, maior será o tempo despendido no controle de fluxo de execução da aplicação e, consequentemente, um ganho menor em desempenho será obtido.

A determinação desta proporção poderia ter sido mais bem explorada, uma vez que a mesma é mandatória na definição do emprego ou não de um coprocessador de ponto flutuante em hardware. Assim sendo, a análise e implementação de uma fórmula que determine precisamente a proporção da aplicação alvo do sistema (fórmula esta que poderia ser nomeada de *taxa* ou *densidade de IPF* - instruções de ponto flutuante - ou algo similar), mostra-se um desafio importante a ser enfrentado no futuro. Isto é particularmente relevante se forem considerados outros requisitos além do desempenho, tais como ocupação de área e dissipação de energia e/ou consumo de energia.

Sob o ponto de vista do requisito de desempenho das aplicações, a adoção de um coprocessador de ponto flutuante em hardware constitui uma alternativa bastante atraente, desde que tais aplicações demandem uma quantidade maciça de cálculos em ponto flutuante (ou possuam uma alta *taxa* ou *densidade de IPF*), conforme já se sabe. O uso de um coprocessador, principalmente se aliado à técnica GALS de projeto, pode trazer um aumento de até 2195% no desempenho das aplicações, em relação à emulação em software das instruções de ponto flutuante. Ou seja, uma aceleração máxima de aproximadamente 22 vezes.

Excetuando-se a organização Plasma-HFP100, o acréscimo de área decorrente da adoção do coprocessador em todas as demais organizações é de aproximadamente 250%. É um acréscimo significativo, em relação a ocupação de área do processador Plasma, que certamente não pode ser desprezado. Porém, se houver área disponível em FPGA, o desempenho alcançado por tais organizações justifica o emprego do coprocessador. Sendo suas ocupações de área similares, pode-se afirmar que a escolha de uma das organizações existentes depende exclusivamente do seu desempenho. Neste caso, por motivos óbvios e a não ser por outras questões como a dissipação de energia e/ou consumo de energia, deve-se empregar a organização de maior desempenho e descartar as demais.

A organização Plasma-HFP100 é a maior organização em termos de ocupação de área, cerca de 370% maior que o processador Plasma e de 150% maior que as demais organizações. A latência de suas instruções de ponto flutuante é baixa, porém a sua estimativa de frequência de operação é baixa em relação às demais: 25 MHz. Sua grande ocupação de área e baixa frequência de operação praticamente o descartaria do espaço de projeto não fosse uma vantagem interessante: seu código HDL é genérico, o que permite sua prototipação em qualquer FPGA e mesmo o porte para ASICs, ao contrário das demais organizações, que empregam módulos de hardware específicos (gerados com o CoreGen) para os FPGAs utilizados no escopo deste trabalho.

Com relação ao consumo de potência das organizações, ainda parece cedo para retirar conclusões. A análise dos dados obtidos está em andamento e novas medidas ainda estão sendo coletadas. Desde já se percebe que as organizações dotadas de CP1 têm uma clara vantagem energética em relação ao uso de emulação de ponto flutuante, o que já se esperava para FPGAs.

Nota-se que a potência média dissipada varia muito pouco em todas as implementações, demonstrando que o tempo de execução nestas é o principal fator determinante de gasto total de energia pela aplicação.

Levando-se em conta todas as considerações tecidas até aqui, conclui-se que a organização Plasma-HFP-GALS 8x apresenta-se como a melhor opção para aplicações que possuem uma alta *taxa* ou *densidade de IPF*. Isto pode ser comprovado pelo fato de que tal organização possui a maior densidade computacional dentre as demais.

Observando os resultados obtidos para a organização Plasma-HFP-GALS-LP, verificou-se que o mesmo possui uma dissipação, dependendo da aplicação, entre 0,5% a 1,5% maior que a Plasma-HFP-GALS 8x, organização na qual foi baseada. Analisando o código fonte do coprocessador da Plasma-HFP-GALS-LP, percebeu-se que os módulos de hardware dos operadores que o compõe, gerados a partir do Coregen, já possuem recurso para desligar o sinal de relógio quando não se encontram em funcionamento. Desta forma, a Plasma-HFP-GALS-LP não apresentou qualquer ganho com a técnica de chaveamento de relógio implementada. O pequeno acréscimo na dissipação de potência pode ser explicado pelo hardware adicional proveniente do controlador de chaveamento de relógio.

Importante ressaltar que os resultados obtidos até meados de julho de 2009 foram publicados na conferência ReConFig'09 [ROD09].

9.2. Trabalhos Futuros

Conforme a Seção anterior, um trabalho a ser realizado futuramente é a implementação da fórmula, ou equação, que determine precisamente a proporção de instruções aritmética de ponto flutuante perante as instruções lógico-aritméticas utilizadas para o controle do fluxo de execução da aplicação alvo do sistema. Como comentado na referida subsecção, esta implementação torna-se mandatória devido a sua importância na verificação da aptidão de uma determinada aplicação alvo se beneficiar com o emprego de uma unidade de ponto flutuante em hardware. Uma possível abordagem inicial seria a determinação, em tempo de compilação, do total de ambos os tipos de instruções existentes na aplicação alvo do sistema.

Outro trabalho interessante a ser realizado no futuro seria a implementação de um mecanismo de ajuste dinâmico, em tempo de execução, do estado do sistema embarcado. Este ajuste se daria a partir de uma série de informações oriundas do contexto, ou ambiente, em que este sistema embarcado está inserido. Levando-se em conta o fato de que sistemas embarcados convergem para dar suporte a múltiplas funcionalidades, tal mecanismo poderia fazer uso da *taxa* ou *densidade de IPF*. Previamente equacionadas, estas podem servir para decidir se uma determinada aplicação deve ser executada empregando-se emulação, diminuindo assim a dissipação de potência e privilegiando o consumo de energia, ou se deve ser executada pelo coprocessador de ponto flutuante, beneficiando o desempenho. Outras informações poderiam ser levadas em consideração, tais como o nível de energia disponível em suas baterias, requisitos de tempo real, entre outros. Adicionalmente, existe a possibilidade de se empregar mais de um coprocessador (obviamente se houver área em silício disponível). Neste caso, seria possível selecionar o coprocessador mais adequado para determinados requisitos de desempenho e de dissipação de potência e/ou consumo de energia. O mecanismo trabalharia de forma autônoma ajustando o estado do sistema, objetivando a obtenção do melhor compromisso possível entre os requisitos de projeto e das aplicações alvo. Desta forma, os recursos disponibilizados pelo sistema embarcados seriam mais bem explorado e, consequentemente, suas utilizações seriam maximizadas.

No projeto em nível de transação entre registradores, citam-se o uso de sincronizadores do tipo fast-flops [DOB09] e a implementação de um mecanismo de controle que permita o emprego de técnicas DFS. O sincronizador *2FF* empregado neste trabalho dificulta a implementação de um mecanismo ou protocolo de comunicação que permita o controle dinâmico das frequências de operação das organizações não-síncronas. Além disso, o emprego da técnica DFS em FPGAs é bastante restrito, uma vez que os componentes destinados ao gerenciamento dos sinais de relógio, tais como o DCM [XIL09d], por exemplo, não permitem o controle dinâmico da frequência em tempo de execução. Uma possível alternativa é apresentada na Figura 35. Nesta alternativa utiliza-se DCMs configurados, em tempo de projeto, para a geração de diferentes frequências de relógio, sendo seus respectivos sinais de relógio multiplexados pelos componentes BUFGMUX [XIL09d], disponíveis nos FPGAS Xilinx da família Virtex-5. Desta forma, um determinado módulo de hardware pode ter a frequência do seu sinal de relógio alterada dinamicamente. Obviamente, este controle deve ser realizado dentro do universo das frequências definidas pelos DCMs empregados.

Outros futuros aperfeiçoamentos nos coprocessadores podem incluir o suporte a precisão dupla, e o tratamento de exceções previstas pela norma IEEE-754. Precisão dupla é requisito de algumas aplicações que não podem tolerar erros devido ao uso de poucos bits de precisão do formato precisão simples. Do ponto de vista de exceções de ponto flutuante, os recursos de hardware apenas sinalizam as exceções, relegando seu tratamento para o software do sistema. O sistema operacional que acompanha a distribuição Plasma, não utilizado nesta pesquisa, pode ser alterado para incluir em seu *kernel* o tratamento de exceções.

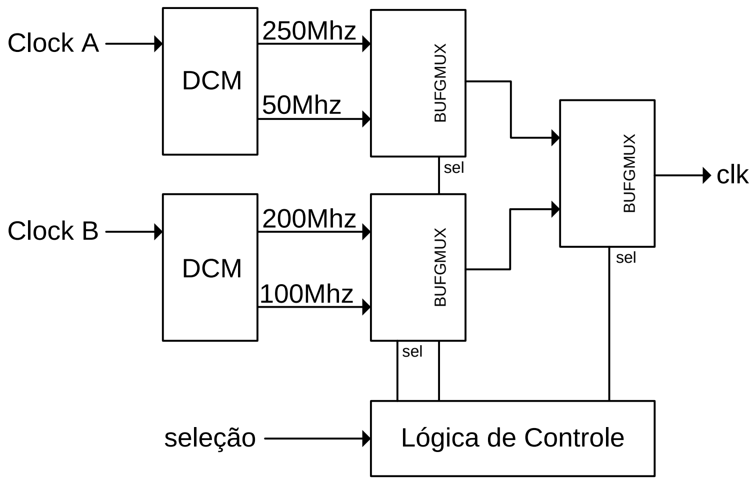

Figura 35 – Possível solução para técnica DFS para FPGAs Xilinx da família Virtex-5.

REFERÊNCIAS

[AGI09] Agilent Technologies, Inc. “DSA80000B Digital Signal Analyzer”. Capturado em: http://www.home.agilent.com/agilent/product.jspx?cc=GB&lc=eng&ckey=829638&nid=-34286.0.00&id=829638, Novembro 2009.

[ALT09] Altera Corporation. “Cyclone II FPGAs at Cost That Rivals ASICS”. Capturado em: http://www.altera.com/products/devices/cyclone2/cy2-index.jsp, Agosto 2009.

[ARQ08] Arquitetura MR2. Capturado em: http://www.inf.pucrs.br/~calazans/undergrad/orgcomp/MR2.zip, Junho 2008.

[AMD05] Amde, M.; Felicijan, T.; Efthymiou, A.; Edwards, D.; Lavagno, L. “Asynchronous on-chip networks”. Computers and Digital Techniques, 152(2), Março 2005, pp. 273-283.

[AME85] American National Standards Institute - Institute of Electrical and Electronics Engineers. “Standard for Binary Floating-Point Arithmetic Standard 754”. 1985.

[AMI07] Amini, E.; Najibi, M.; Jeddi, Z.; Pedram, H. “FPGA Implementation of Gated Clock based Globally Asynchronous Locally Synchronous Wrapper Circuits”. In: International Symposium on Signals Circuits and Systems (ISSCS’07), Julho 2007, pp. 1-4.

[BEA06] Beauchamp, M.; Hauck, S.; Underwood, K.; Hemmert, K. “Embedded floating-point units in FPGAs”. In: 14th International Symposium on Field Programmable Gate Arrays (FPGA’06), Fevereiro 2006, pp. 12-20.

[BEC03] Becker, J.; Huebner, M.; Ullmann, M. “Power Estimation and Power Measurement of Xilinx Virtex FPGAs: Trade-offs and Limitations”. In: 16th Symposium on Integrated Circuits and Systems Design (SBCCI’03), Setembro 2003, pp. 283-288.

[BEL02] Belanovic, P.; Leeser, M. “A Library of Parameterized Floating-Point Modules and Their Use”. In: 12th Int. Conf. on Field-Programmable Logic and Applications (FPL’02), Setembro 2002, pp. 657-666.

[BRU05] Brunelli, C.; Garzia, F.; Campi, F.; Mucci, C.; Nurmi, J. “A FPGA Implementation of an Open-Source Floating-Point Computation System”. In: Proceedings of the International Symposium on System-on-Chip (SoC’05), Novembro. 2005, pp. 29-32.

[CAL98] Calazans, N. L. V. “Automated Logic Design of Sequential Digital Circuits”. Imprinta, 1998. 342p. Capturado em: http://www.inf.pucrs.br/~calazans/publications/prjlog/v1.0/.

[CHA84] Chapiro, D. M. “Globally-Asynchronous Locally Synchronous Systems”. PhD Thesis, Stanford University, Outubro 1984, 134 p.

[CHO09] Chong, Y.; Parameswaran, S. “Flexible multi-mode embedded floating-point unit for field programmable gate arrays”. In: 17th International Symposium on Field Programmable Gate Arrays (FPGA’09), Fevereiro 2009, pp. 171-180.

[COM74] Computer Systems Laboratory. “Macromodular Computer Design Part I - Development of Macromodules Volume IV - The Synchronozer "GLITCH" Problem -

Technical Report No. 47". Relatório Técnico, Washington University, St. Louis, 1974, 61p.

[COR00] Cortadella, J.; Kishinevsky, M.; Kondratyev, A.; Lavagno, L. "Introduction to asynchronous circuit design: specification and synthesis". Tutorial. In: 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC'00), Abril 2000.

[CPU09a] CPU World. "Atom Z550 Specs". Capturado em: http://www.cpu-world.com/CPUs/Atom/Intel-Atom Z550 AC80566UE041DW.html, Agosto 2009.

[CPU09b] CPU World. "AMD Mobile Sempron 2800+". Capturado em: http://www.cpu-world.com/CPUs/K8/AMD-Mobile Sempron 2800 - SMS2800BOX3LB.html, Agosto 2009.

[DID02] Dido, J.; Geraudie, N.; Loiseau, L.; Payeur, O.; Savaria, Y.; Poirier, D. "A flexible floating-point format for optimizing data-paths and operators in FPGA based DSPs". In: 10th International Symposium on Field Programmable Gate Arrays (FPGA'02), Fevereiro 2002, pp. 50-55.

[DOB09] Dobkin, R.; Ginosar, R. "Two-phase synchronization with sub-cycle latency". Integration the VLSI Journal, 42(3), Junho 2009, pp. 367-375.

[DON03] Donno, M.; Ivaldi, A.; Benini, L.; Macii, E. "Clock-tree power optimization based on RTL clock-gating". In: Design Automation Conference (DAC'03), 2003, pp. 622-627.

[DSP08] DSP Design Line. "Tutorial: Floating-point arithmetic on FPGAs". Capturado em: http://www.dspdesignline.com/showArticle.jhtml?articleID=196603215, Junho 2008.

[DSP09] DSP Design Line. "Fixed vs floating point: a surprisingly hard choice". Capturado em: http://www.dspdesignline.com/howto/197002280, Agosto 2009.

[EET01] EETimes. "ARM adds floating-point unit to processor core". Capturado em: http://eetimes.com/electronics-news/4131968/ARM-adds-floating-point-unit-to-processor-cores, Junho 2001.

[EET09] EETimes Europe. "Floating point coprocessor is tailored to handle GPS data". Capturado em: http://www.eetimes.eu/products/dsp/199703869, Agosto 2009.

[EXT09] ExtremeTech. "Sony Details PSP Chip Specs". Capturado em: http://www.extremetech.com/article2/0,1558,1639250,00.asp, Agosto 2009.

[GIN03] Ginosar, R. "Fourteen ways to fool your synchronizer". In: IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'03), May 2003, pp. 89-96.

[HAR93] Harrod, P. L.; Baum, A. J.; Biggs, J. P.; Howard, D. W.; Merritt, A. J.; Oldham, H. E.; Seal, D. J.; Watters, H. L. "FPA10-A 4 MFLOP floating point coprocessor for ARM". In: Custom Integrated Circuits Conference (CICC'93), Maio 1993, pp. 1-4.

[HDR09] HDRsoft Ltd. "FAQ – HDR images for photography". Capturado em: http://www.hdrsoft.com/resources/dri.html#hdri, Agosto 2009.

[HYU05] Hyung, L.; Kyungsoo, L.; Yongseok, C.; Naehyuck, C. “Cycle-Accurate Energy Measurement and Characterization of FPGAs”. Analog Integrated Circuits and Signal Processing, 42(3), Março 2005, pp. 239-251.

[INA96] Inacio, C.; Ombres, D. “The DSP decision: fixed point or floating?”. IEEE Spectrum, 33(9), Setembro 1996, pp. 72-74.

[ITR05] International Technology Roadmap for Semiconductors. “ITRS 2005 Edition”. Capturado em: http://www.itrs.net/Links/2005ITRS/Home2005.htm, Dezembro 2005.

[IYE02] Iyer, A.; Marculescu, D. “Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors”. In: 29th International Symposium on Computer Architecture (ISCA’02), Maio 2002, pp. 158-168.

[JEO99] Jeong, C.-H.; Park, W.-C.; Han, T.-D.; Kim, S.-D. “Cost/Performance Trade-Off in Floating-Point Unit Design for 3D Geometry Processor”. In: The First IEEE Asia Pacific Conference on ASICs (AP-ASIC’99), Agosto 1999 pp. 104–107.

[JIE08] Jie, S.; Ning, Y.; Xiao-Yan, Z. “An IEEE compliant floating-point adder with the deeply pipelining paradigm on FPGAs”. In: 2008 International Conference on Computer Science and Software Engineering (CSSE’08), Setembro 2008, pp. 50-53.

[JON06] Jones, P. H.; Cho, Y. H.; Lockwood, J. W. “An Adaptative Frequency Control Method using Thermal Feedback for Reconfigurable Hardware Applications”. In: IEEE International Conference on Field Programmable Technology (FPT’06), Dezembro 2006, pp. 229-236.

[KAD07] Kadlec, J.; Bartosinski, R.; Danek, M. “Accelerating Microblaze floating point operations”. In: 17th International Conference on Field-Programmable Logic and Applications (FPL’07), Setembro. 2007, pp. 621-624.

[KIL07] Kilts, S. “Advanced FPGA Design: Architecture, Implementation, and Optimization”. Wiley-Interscience, 2007. 336p.

[LUO05] Luo, Y.; Yu, J.; Yang, J.; Bhuyan, L. “Low Power Network Processor Design Using Clock Gating”. In: Designa Automation Conference (DAC’05), 2005, pp. 712-715.

[MAN07] Manders, C.; Farbiz, F.; Mann, S. “A Compression Method for Arbitrary Precision Floating-Point Images”. In: IEEE International Conference on Image Processing (ICIP’07), Vol. 4, Outubro 2007, pp. IV.165-IV.168.

[MES90] Messerschmitt, D. G. “Synchronization in Digital System Design”. IEEE Journal of Selected Areas in Communications, 8(8), Outubro 1990, pp. 1404-1419.

[MIS08] Missouri State University. “MARS (MIPS Assembler and Runtime Simulator)”. Capturado em: http://courses.missouristate.edu/KenVollmar/MARS/, Julho 2008.

[MOY01] Moyer, B. “Low-Power Design for Embedded Processors”. Proceedings of the IEEE, 89(11), Novembro 2001, pp. 1576-1587.

[MUT00] Muttersbach, J.; Villiger, T.; Fichtner, W. “Practical Design of Globally-Asynchronous Locally-Synchronous Systems”. In: IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC’00), 2000, pp. 52-59.

[OPE08a] OpenCores.Org. “Plasma – most MIPS I (TM) opcodes: Overview”. Capturado em: http://www.opencores.org/projects.cgi/mipsoverview, Junho 2008.

[OPE08b] OpenCores.Org. “FPU: Overview”. Capturado em: http://www.opencores.org/ projects.cgi/web/fpu100/overview, Junho 2008.

[PAP04] Pappalardo, F.; Visalli, G. “An Application-Oriented Analysis of Power/Precision Trade-Off in Fixed and Floating-Point Arithmetic Units for VLSI Processors”. In: IASTED International Conference on Circuits, Signals and Systems, Dezembro 2004, pp. 416-421.

[PAP08] Papakonstantinou, A.; Kifle, Y.; Lucas, G.; Chen, D. “MP3 decoding on FPGA: a case study for floating point acceleration”. In: 4th Annual Reconfigurable Systems Summer Institute (RSSI’08), Julho 2008, 4pp.

[PON07] Pontes, J.; Soares, R.; Carvalho, E.; Moraes, F.; Calazans, N. “SCAFFI: An intrachip FPGA asynchronous interface based on hard macros”. In: 25th IEEE International Conference on Computer Design, (ICCD07), pp. 541-546, 2007.

[PS309] PS3 Ign. “E3 2005: PS3 Tech Specs”. Capturado em: http://ps3.ign.com/articles/614/614682p1.html, Agosto 2009.

[RAB96] Rabaey, J; Pedram, M. “Low Power Design Methodologies”. Kluwer Academic Publishers, 1996, 367p.

[ROD09] Rodolfo, T. A.; Calazans, N. L. V.; Moraes, F. G. “Floating Point Hardware for Embedded Processors in FPGAs: Design Space Exploration for Performance and Area”. In: International Conference on Reconfigurable Computing and FPGAs (ReConFig'09), Dezembro 2009, pp. 24 - 29.

[RTC09] RTC Magazine. “New Applications Benefit from Floating-Point DSP Accuracy”. Capturado em: http://www.rtcmagazine.com/articles/view/100255, Agosto 2009.

[SAL06] Saleh, R.; Wilton, S.; Mirabbasi, S.; Hu, A.; Greenstreet M.; Lemieux G.; Pande P.; Grecu C.; Ivanov A. “System-on-Chip: Reuse and Integration”. Proceedings of the IEEE, 94(6), Junho 2006, pp. 1050-1069.

[SEM04] Semeraro, G.; Albonesi, D.; Maglkis, G.; Scott, M.; Dropsho, S.; Dwarkadas, S. “Hiding Synchronization Delays in a GALS Processor Microarchitecture”. In: 10th International Symposium on Asynchronous Circuits and Systems (ASYNC’04), Abril 2004, pp. 159-169.

[SPA02] Sparsø, J. and Furber, S. (Eds). “Principles of asynchronous circuit design - A systems perspective”. Springer, 2002, 360p.

[STA98] Staken, P. H. “A Practioner’s Guide to RISC Microprocessor Architecture”. Wiley-Interscience, 1996. 400p.

[STE09] Steiner, G.; Jones, B.; Alfke, P. “Floating Point: Have it Your Way with FPGA Embedded Processors”. Xcell Journal. 1º Trimestre. 2009, pp. 32-35.

[TEX08] Texas Instruments, Inc. “Comparing Fixed- and Floating-point DSPs”. Capturado em: http://focus.ti.com/lit/wp/spry061/spry061.pdf, Junho 2008.

[VEN05] Venkatachalam, V.; Franz, M. “Power Reduction Techniques for Microprocessor Systems”. ACM Computing Surveys, 37(3), Setembro 2005, pp. 195-237.

[VIA09] VIA Technologies, Inc. “Via Nano Processor”. Capturado em: http://www.via.com.tw/en/products/processors/nano/, Agosto 2009.

[WAN09] Wang, Q.; Gupta, S.; Anderson, G. “Clock Power Reduction for Virtex-5 FPGAs”. In: In: 17th International Symposium on Field Programmable Gate Arrays (FPGA’09), Fevereiro 2009, pp. 13-22.

[XIL09a] Xilinx, Inc. “Spartan6 FPGA Family”. Capturado em: http://www.xilinx.com/products/spartan6/index.htm, Agosto 2009.

[XIL09b] Xilinx, Inc. “CORE Generator Guide”. Capturado em: http://www.xilinx.com/itp/xilinx6/books/data/doc/cgn/cgn0001_1.html, Agosto 2009.

[XIL09c] Xilinx, Inc. “Virtex-5 FPGA ML550 Networking Interfaces Platform - User Guide”. Capturado em: http://www.xilinx.com/support/documentation/boards_and_kits/ug202.pdf, Novembro 2009.

[XIL09d] Xilinx, Inc. “Virtex-5 Libraries Guide for HDL Designs”. Capturado em: http://www.xilinx.com/itp/xilinx82/books/docs/v5ldl/v5ldl.pdf, Novembro 2009.

[XIL09e] Xilinx, Inc. “Virtex-5 FPGA System Power Design Considerations”. Capturado em: http://www.xilinx.com/support/documentation/white_papers/wp285.pdf, Novembro 2009.

[XIL09f] Xilinx, Inc. “Xilinx Power Estimator User Guide”. Capturado em: http://china.xilinx.com/support/documentation/user_guides/ug440.pdf, Novembro 2009.

[YUH09] Yuhua, Z.; Longhua, Q.; Qiang, L.; Peide, Q.; Lei, Z. “A Dynamic Frequency Scaling Solution to DPM in Embedded Linux Systems”. In: 2009 International Conference on Information Reuse & Integration (IRI’09), Agosto 2009, pp. 256-261.

[ZHA06] Zhang, Y.; Roivainen, J.; Mämmelä, A. “Clock-Gating in FPGAs: A Novel and Comparative Evaluation”. In: 9th Conference on Digital System Design: Architectures, Methods and Tools (DSD’06), Agosto. 2006, pp. 584-590.

[ZII09] Zii Graphics and Imaging. “3DLABS Demonstrates 3D Navigation, Software GPS and Rich Media Capabilities on its DMS-02 Media Processor”. Capturado em: http://www.zii.com/resource/graphics-and-imaging/2006-dec-5-3dlabs-demonstrates.aspx, Agosto 2009.

Apêndice A – Código Fonte do Programa de Teste com todas as Instruções de PF

```
int main(void){

      while(1){
            {
                  asm("lui $9,0");
                  asm("ori $9,$9,16");
                  asm("mtc1 $9,$f2");
                  asm("cvt.s.w $f4,$f2");
                  asm("neg.s $f26, $f4");
                  asm("add.s $f6,$f26,$f0");
                  asm("add.s $f6,$f26,$f26");
                  asm("add.s $f6,$f4,$f0");
                  asm("cvt.w.s $f8,$f6");
                  asm("sub.s $f10,$f26,$f0");
                  asm("sub.s $f10,$f26,$f26");
                  asm("sub.s $f10,$f4,$f0");
                  asm("cvt.w.s $f12,$f10");
                  asm("mul.s $f14,$f26,$f0");
                  asm("mul.s $f14,$f26,$f26");
                  asm("mul.s $f14,$f4,$f0");
                  asm("cvt.w.s $f16,$f14");
                  asm("div.s $f18,$f26,$f0");
                  asm("div.s $f18,$f0,$f26");
                  asm("div.s $f18,$f26,$f26");
                  asm("div.s $f18,$f4,$f0");
                  asm("cvt.w.s $f20,$f18");
                  asm("mfc1 $10,$f20");
                  asm("add $10,$10,$10");
                  asm("mov.s $f22, $f10");
                  asm("neg.s $f22, $f22");
                  asm("abs.s $f24, $f22");
                  asm("c.lt.s $f10,$f14");
                  asm("bc1t $L1");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("$L1:c.lt.s $f14,$f10");
                  asm("bc1t $L3");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("$L3:c.lt.s $f14,$f14");
                  asm("bc1t $L4");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("$L4:c.lt.s $f14,$f10");
                  asm("bc1f $L5");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("$L5:c.lt.s $f10,$f14");
                  asm("bc1f $L6");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("nop");
                  asm("nop");
```

```
                        asm("$L6:c.eq.s $f10,$f10");
                        asm("bc1t $L7");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("$L7:c.eq.s $f14,$f10");
                        asm("bc1t $L8");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("$L8:c.eq.s $f10,$f14");
                        asm("bc1t $L9");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("$L9:c.le.s $f10,$f10");
                        asm("bc1t $L10");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("$L10:c.le.s $f14,$f10");
                        asm("bc1t $L11");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("$L11:c.le.s $f10,$f14");
                        asm("bc1t $L12");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("nop");
                        asm("$L12:nop");
                }
        }
}
```

Apêndice B – Código Fonte dos Programas de Cálculo do SENO e COSENO

```
// SENO - SERIE DE TAYLOR
//....................................(-1)^n
//    sin(X)= somatorio de 0 a 10 de ------- * X^(2n+1)
// (2n+1)!

int main(void)
{
      float X=4.5;
      float prim_potenciacao,seg_potenciacao,factorial,divisao,multiplicação;
      float somatorio=0;
      float seno;

      int a,N=10,Ni,exp;

      while(1)
      {
            a=464235435; //IDENTIFICADOR DO INICIO DE PROGRAMA - x"1BABABAB"

            for(Ni=0;Ni<N;Ni++)
            {
                  for(exp=0;exp<Ni+1;exp++)
                  {
                        if(exp==0) prim_potenciacao=1;
                        else prim_potenciacao=prim_potenciacao*(-1);
                  }

                  if(Ni==0)
                  {
                        seg_potenciacao=X;
                        fatorial=1;
                  }
                  else{
                        for(exp=0;exp<((2*Ni))+1;exp++)
                        {
                              if(exp==0)
                              {
                                    seg_potenciacao=X;
                                    fatorial=1;
                              }
                              else{
                                    seg_potenciacao=seg_potenciacao*X;
                                    fatorial=fatorial*(exp+1);
                              }
                        }
                  }
                  divisao=prim_potenciacao/fatorial;
                  multiplicacao=divisao*seg_potenciacao;
                  somatorio=somatorio+multiplicacao;
            }
            seno=somatorio;
            a=481078444; //IDENTIFICADOR DO FIM DE PROGRAMA - x"1CACACAC"
      }
}
```

```
// COSENO - SERIE DE TAYLOR
// (-1)^n
//    cos(X)= somatorio de 0 a 10 de ------- * X^(2n)
// (2n)!

int main(void){

      float X=4.5;
      float prim_potenciacao,seg_potenciacao,fatorial,divisão,multiplicacao;
      float somatorio=0;
      float coseno;

      int a,N=10,Ni,exp;

      while(1)
      {
            a=464235435; //IDENTIFICADOR DO INICIO DE PROGRAMA - x"1BABABAB"

            for(Ni=0;Ni<N;Ni++)
            {
                  for(exp=0;exp<Ni+1;exp++)
                  {
                        if(exp==0) prim_potenciacao=1;
                        else prim_potenciacao=prim_potenciacao*(-1);
                  }

                  if(Ni==0)
                  {
                        fatorial=1;
                        seg_potenciacao=1;
                  }
                  else{
                        for(exp=0;exp<((2*Ni));exp++)
                        {
                              if(exp==0)
                              {
                                    fatorial=1;
                                    seg_potenciacao=X;
                              }
                              else{
                                    fatorial=fatorial*(exp+1);
                                    seg_potenciacao=seg_potenciacao*X;
                              }
                        }
                  }
                  divisao=prim_potenciacao/fatorial;
                  multiplicacao=divisao*seg_potenciacao;
                  somatorio=somatorio+multiplicacao;
            }
            coseno=somatorio;
            a=481078444; //IDENTIFICADOR DO FIM DE PROGRAMA - x"1CACACAC"
      }
}
```

Apêndice C – Código Fonte dos Programas de Cálculo dos Filtros FIR e IIR

```
// FILTRO FIR
//
// y += h[k] * x[(oldest + k) % N];

int main(void)
{
      float x[5], h[5];
      float sample, y, mul_float, output;
      int oldest=0,a,b,k,div,mul,mod;

      //inicializando vetor x
      x[0]=0;x[1]=0;x[2]=0;x[3]=0;x[4]=0;
      //vetor de coeficientes
      h[0]=-0.005627475;
      h[1]=-0.044131055;
      h[2]=0.9229929;
      h[3]=-0.044131055;
      h[4]=-0.005627475;

      b=5;

      while(1)
      {
            a=464235435; //IDENTIFICADOR DO INICIO DE PROGRAMA - x"1BABABAB"

            sample=452.587; // read sample
            y=0;
            for (k=0;k<5;k++)
            {
                  //funcao mod
                  a=oldest+k;
                  div=a/b;
                  mul=div*b;
                  mod=a-mul;

                  mul_float = h[k] * x[mod];
                  y = y + mul_float;
            }

            //funcao mod
            a=oldest+1;
            div=a/b;
            mul=div*b;
            mod=a-mul;

            //oldest = (oldest + 1) % N;
            oldest = mod;
            output=y; // write output

            a=481078444; //IDENTIFICADOR DO FIM DE PROGRAMA - x"1CACACAC"
      }
}
```

```
// FILTRO IIR
//
// y[nT]=(b[0]*x[nT]) + (b[1]*x[nT-T]) + (b[2]*x[nT-2T]) + (b[3]*x[3]) +
// (b[4]*x[4]) + (a[1]*y[nT-T]) + (a[2]*y[nT-2T]) + (a[3]*y[nT-3T]) +
// (a[4]*y[nT-4T]);

int main(void)
{
      float output, x[5], y[5];
      int i,a;

      //inicializando vetor x
      x[0]=0;x[1]=0;x[2]=0;x[3]=0;x[4]=0;

      //inicializando vetor y
      y[0]=0;y[1]=0;y[2]=0;y[3]=0;y[4]=0;

      while(1)
      {
            a=464235435; //IDENTIFICADOR DO INICIO DE PROGRAMA - x"1BABABAB"

            for(i=4;i<0;i--) x[i]=x[i-1];

            x[0]=452.587f; // read sample

            y[0]=((1.0f)*x[0]) - ((2.7600465E-16f)*x[1]) + ((0.48602882f)*x[2])
            - ((5.7882413E-17f)*x[3]) + ((0.017664801f)*x[4])
            + ((0.37592337f)*y[1]) + ((0.56388503f)*y[2])
            + ((0.37592337f)*y[3]) + ((0.09398084f)*y[4]);

            for(i=4;i<0;i--) y[i]=y[i-1];
            output=y[0];

            a=481078444; //IDENTIFICADOR DO FIM DE PROGRAMA - x"1CACACAC"
      }
}
```

Apêndice D – Código Fonte do Programa calcPOT

```
int main (int argc, char * argv[])
{
	char filename[80];
	float f,total=0,texec,pico=0,temp,energia,pot_media;
	FILE * pFile;
	int i=0;

	if (argc != 4) {printf("\n Erro de sintaxe! Uso correto:\n\n ./calcPOT
	<\"nome_do_arquivo\"> <tempo de execucao> <taxa de amostragem>\n\n");
	return 0;}

	texec = atof(argv[2]);
	pFile = fopen (argv[1],"r");
	if (pFile==NULL) {printf("\n Erro abrindo o arquivo ou arquivo
	inexistente!\n\n"); return 0;}

	while (!feof(pFile))
	{
		fscanf (pFile, "%f", &f);//LE TEMPO DA PRIMEIRA COLUNA
		fscanf (pFile, "%f", &f);//LE AMOSTRA DA SEGUNDA COLUNA
		temp=fabs(f-0.130)/0.010; //P=V*(Vs/0.010) Watts -> P=1*((Vs-
		offset)/0.010)
		total=total+temp;//INTEGRAL
		if(pico<temp) pico=temp; //FUNCAO MAX
	}

	switch(atoi(argv[3]))
	{
	 case 100:energia=(total*0.00000001);pot_media=energia/texec;break;
	 case 200:energia=(total*0.000000005);pot_media=energia/texec;break;
	 case 250:energia=(total*0.000000004);pot_media=energia/texec;break;
	 case 400:energia=(total*0.0000000025);pot_media=energia/texec;break;
	 case 500:energia=(total*0.000000002);pot_media=energia/texec;break;
	 case 1000:energia=(total*0.000000001);pot_media=energia/texec;break;
	 case 2000:energia=(total*0.0000000005);pot_media=energia/texec;break;
	 case 5000:energia=(total*0.0000000002);pot_media=energia/texec;break;
	 case 10000:energia=(total*0.0000000001);pot_media=energia/texec;break;
	 case 20000:energia=(total*0.00000000005);pot_media=energia/texec;break;
	 case 40000:energia=(total*0.000000000025);pot_media=energia/texec;break;
	}
	fclose (pFile);

	strncpy(filename,argv[1],strlen(argv[1])-4);
	strcat(filename,"_RESULTS.txt");
	pFile = fopen (filename,"w+");
	fprintf (pFile, " POT_MEDIA: %.15f Watts\nENERGIA: %.15f Joules\nPICO:
	%.15f Watts\n",total,i-1,pot_media,energia,pico);

	printf("\n Arquivo \"%s\" gerado.\n\n",filename);

	return 0;
}
```