



Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Comunicações



ARQUITETURA COMPUTACIONAL HÍBRIDA BASEADA EM DSP E FPGA PARA PROCESSAMENTO DIGITAL DE SINAIS

Autor: Éricles Rodrigues Sousa

Orientador: Prof. Dr. Luís Geraldo Pedroso Meloni

Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: **Telecomunicações e Telemática.**

Banca Examinadora

Prof. Dr. Luís Geraldo Pedroso Meloni (presidente) — DECOM/FEEC/UNICAMP

Prof. Dr. Sílvio Ernesto Barbin — PTC/EP/USP

Prof. Dr. José Raimundo de Oliveira — DCA/FEEC/UNICAMP

Campinas – SP
26 de julho de 2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

So85a Sousa, Éricles Rodrigues
Arquitetura computacional híbrida baseada em DSP e
FPGA para processamento digital de sinais / Éricles
Rodrigues Sousa. --Campinas, SP: [s.n.], 2011.

Orientador: Luís Geraldo Pedroso Meloni.
Dissertação de Mestrado - Universidade Estadual de
Campinas, Faculdade de Engenharia Elétrica e de
Computação.

1. Processamento digital de sinais. 2. FPGA. 3.
Modelos matemáticos. 4. Sinergia. I. Meloni, Luís
Geraldo Pedroso. II. Universidade Estadual de
Campinas. Faculdade de Engenharia Elétrica e de
Computação. III. Título.

Título em Inglês: Hybrid computing architecture based on DSP and FPGA for
digital signal processing

Palavras-chave em Inglês: Digital signal processing, FPGA, Mathematical
models, Synergy

Área de concentração: Telecomunicações e Telemática

Titulação: Mestre em Engenharia Elétrica

Banca examinadora: Sílvio Ernesto Barbin, José Raimundo de Oliveira

Data da defesa: 26-07-2011

Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Ericles Rodrigues Sousa

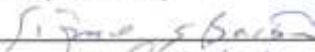
Data da Defesa: 26 de julho de 2011

Título da Tese: "Arquitetura Computacional Híbrida Baseada em DSP e FPGA para Processamento Digital de Sinais"

Prof. Dr. Luis Geraldo Pedroso Meloni (Presidente):



Prof. Dr. Sílvio Ernesto Barbin:



Prof. Dr. José Raimundo de Oliveira:



Resumo

Atualmente, aplicações multimídias exigem grande esforço computacional para manipular dados com elevadas taxas de precisão. Visando otimizar a capacidade de processamento sem elevar demasiadamente o custo do desenvolvimento em sistemas embarcados, este trabalho descreve a proposta de uma arquitetura computacional híbrida, para processamento digital de sinais, baseado-se no uso cooperativo entre DSP (*Digital Signal Processor*) e FPGA (*Field Programmable Gate Array*).

Neste estudo é realizada uma abordagem sobre o uso de um coprocessador para a acelerar rotinas que demandam grande esforço computacional em um DSP. Também é proposto um modelo matemático capaz de mensurar a eficiência do particionamento de códigos processados de forma descentralizada.

Para validação da proposta, foi construído um cenários de testes para a estimação de vetores movimento, um dos principais agentes envolvidos no processo de codificação de vídeo em alta definição.

A partir do cenário elaborado foi possível constatar a eficiência da arquitetura proposta. Sendo que, considerando um código de referência otimizado e baseado na descrição feita em [30], obteve-se mais de 97% de eficiência computacional.

Assim, este estudo permite concluir que o uso cooperativo entre DSP e FPGA se mostra muito vantajoso devido à possibilidade de unir em um único sistema as vantagens fornecidas por ambos dispositivos, caracterizando um ambiente de total sinergia e de elevada capacidade de computacional.

Palavras-chave: *DSP, FPGA, Arquitetura Computacional Híbrida, Processamento Digital de Sinais, Modelo Matemático, Sinergia e Alto Desempenho.*

Abstract

Nowadays, multimedia applications require high computational effort to manipulate data with high precision. In order to optimize the processing power without significantly increasing the cost of development in embedded systems, this work describes the proposal for a hybrid computing architecture applied to digital signal processing, based on the cooperative work between DSP (Digital Signal Processor) and FPGA (Field Programmable Gate Array).

An approach about the use of coprocessor able to accelerate a process which requires great computational effort of a DSP is provided by this study. It is also describes a mathematical model able to measure the efficiency of a partitioning code processed in a distributed system.

To validate our proposal we developed a testbed for calculate the motion estimation vector, which is one of key elements involved on high definition video coding.

From the elaborated testbed, we could found a high efficiency provided by the architecture proposed. Therefore, considering a optimized reference code based on [30], was possible achieve a computing efficiency around 97%.

This study show that cooperative work between DSP and FPGA that provides a very advantageous scenario applied to embedded systems, due to joining the features of both devices, building then, a synergy environment of high computing performance.

Key-words: *DSP, FPGA, Hybrid Computing Architecture, Digital Signal Processing, Mathematical Model, Synergy and High Performance.*

Artigos Publicados Pelo Autor

1. Sousa, E. R. and Meloni, L. G. P. "*An Analytical Model Proposed for Evaluating Efficiency of Partitioning Code in Hybrid Architectures Based on DSP and FPGA*". *IEEE International Symposium of Advances on High Performance Computing and Networking*. Alberta, Canadá, 2011 (aceito para publicação).
2. Sousa, E. R. and Meloni, L. G. P. "*High-Performance Computing Based on Heterogeneous Architectures*". *IEEE International Conference on Information Systems and Computational Intelligence*. Harbin, China, 2011.
3. Sousa, E. R. and Meloni, L. G. P. "*FPGA as Coprocessor Improving Performance for Embedded Digital Signal Processing*". *International Conference on Embedded Systems and Intelligent Technology – ICESIT 2011*. Phuket, Tailândia, 2011.
4. Sousa, E. R., LemesFh, J. M. and Meloni, L. G. P. "**Arquiteturas Híbridas e Reconfiguráveis de Múltiplos Propósitos Baseadas em DSP e FPGA**". VIII Encontro Anual de Computação, Universidade Federal de Goiás, Brasil. ISSN: 2178-6992, 2010.
5. Sousa, E. R. and Meloni, L. G. P. "**Exploração de Paralelismo Computacional em Sistemas Embarcados para Compressão de Imagens Digitais**". I Simpósio de Processamento de Sinais da UNICAMP. Universidade Estadual de Campinas-SP, outubro de 2010.
6. Sousa, E. R., Meloni, L.G.P., Monteiro, C. C., and Gondim, P. R. L. "**Plataforma Estruturada em Software para Desenvolvimento e Testes de Aplicações Segundo o Padrão SBTVD**". II Simpósio Internacional de Competências em Tecnologias

Digitais Interativas na Educação, vol. 01, pg. 63-75, ISSN: 2175-6546, Campinas, Brasil, 2009.

7. Monteiro, C., Pereira, H., Gondim, P., Rios, V., Sousa, E. R., “**Implementação de Set-top Box Virtual para Desenvolvimento de Aplicações para TV Digital Interativa**”. *8th International Information and Telecommunication Technologies Symposium - I2TS*, Florianópolis, Brasil, 2009.

À minha esposa, pais e irmãos.

Agradecimentos

Agradeço primeiramente a Deus, o qual é digno de toda honra e de toda glória.

À minha esposa, fiel companheira e grande motivadora.

Aos meus pais que me educaram e ensinaram valores éticos e morais.

Aos meus irmãos, verdadeiros exemplos de dedicação e empenho.

Ao Professor Dr. Luís G. P. Meloni pela oportunidade concedida e pela orientação desse trabalho.

Sumário

LISTA DE FIGURAS.....	XIV
LISTA DE TABELAS.....	XVI
LISTA DE ABREVIACÕES.....	XVII
1.INTRODUÇÃO.....	19
1.1 MOTIVAÇÃO.....	19
1.2 OBJETIVO.....	22
1.3 METODOLOGIA.....	23
1.4 ESTRUTURA DO TRABALHO.....	23
2.DSP - DIGITAL SIGNAL PROCESSOR	25
2.1 DESCRIÇÃO DA ARQUITETURA.....	25
2.2 NÚCLEO.....	26
2.2.1 MAC – UNIDADE DE MULTIPLICAÇÃO E ACUMULAÇÃO.....	27
2.2.2 ULA – UNIDADE LÓGICA ARITMÉTICA.....	28
2.3 MEMÓRIA.....	29
2.3.1 ACESSO DIRETO À MEMÓRIA.....	29
2.4 REQUISIÇÃO DE SERVIÇO DE INTERRUPÇÃO.....	30
2.5 PIPELINE.....	30
3.FPGA - FIELD PROGRAMMABLE GATE ARRAY.....	32
3.1 ARQUITETURA.....	32
3.2 ETAPAS DE DESENVOLVIMENTO.....	34

3.2.1 DESIGN.....	35
3.2.2 SÍNTESE.....	37
3.2.3 IMPLEMENTAÇÃO.....	37
3.2.4 SIMULAÇÃO.....	37
3.2.5 PROGRAMAÇÃO.....	38
3.3 SOFTPROCESSORS.....	38
4.ARQUITETURA COMPUTACIONAL PROPOSTA.....	42
4.1 AMBIENTE DE SINERGIA: COOPERAÇÃO ENTRE DSP E FPGA.....	43
4.1.1 PROCESSADOR DIGITAL DE SINAIS - DSP	43
4.1.2 <i>FIELD PROGRAMMABLE GATE ARRAY</i> - FPGA.....	44
4.2 DESCRIÇÃO SISTÊMICA.....	44
4.2.1 INTERCONEXÃO DOS DISPOSITIVOS.....	46
4.2.1.1 SINAIS DE CONTROLE.....	47
4.2.1.2 MECANISMO PARA EVITAR A CONTENÇÃO DE BARRAMENTO.....	52
4.2.2 PARTICIONAMENTO DE CÓDIGOS.....	53
4.2.3 MODELO MATEMÁTICO PROPOSTO.....	58
4.2.3.1 CAPACIDADE DE EXPANSÃO	60
4.2.3.2 EFICIÊNCIA GLOBAL.....	61
5.SIMULAÇÕES E RESULTADOS.....	65
5.1 RECURSOS UTILIZADOS.....	65
5.1.1 SMARTDESIGN.....	65
5.1.2 MODELSIM.....	66
5.1.3 PROCESSADOR DIGITAL DE SINAIS ADSP-BF533.....	67
5.1.4 VISUALDSP++.....	70
5.2 CENÁRIO DE TESTES.....	72
5.2.1 ESTIMAÇÃO DE MOVIMENTO.....	72
5.2.2 CONCEPÇÃO DO CÓDIGO DE REFERÊNCIA.....	75
5.2.2.1 PARTICIONAMENTO DO CÓDIGO.....	77
5.2.2.1.1 MÓDULO DSP.....	78

5.2.2.1.2 MÓDULO FPGA.....	79
5.2.2.1.3 INTEGRAÇÃO DOS MÓDULOS.....	81
5.2.2.2 RESULTADOS E ANÁLISES.....	83
6.CONCLUSÕES.....	90
REFERÊNCIAS BIBLIOGRÁFICAS.....	92

Lista de Figuras

Figura 1 – Ciclos de <i>clock</i> para o cálculo de uma FFT.....	20
Figura 2 – Tempo de processamento de uma FFT de diferentes tamanhos.....	21
Figura 3 - Visão simplificada da unidade MAC [31].....	28
Figura 4 - Arquitetura genérica de uma FPGA [40].....	33
Figura 5 - Etapas de desenvolvimento de circuitos em FPGA [40].....	35
Figura 6 - Fluxograma da descrição de circuitos em linguagem C, adaptado de [33].....	36
Figura 7 - Diagrama de blocos da arquitetura proposta.....	45
Figura 8 - Interconexão lógica entre DSP e FPGA.....	47
Figura 9 - Diagrama de tempo para escrita de dados na FPGA [20].....	50
Figura 10 - Pseudocódigo para envio dos dados para a FPGA.....	51
Figura 11 - <i>Double-buffer</i>	52
Figura 12 - Particionamento de código entre DSP e FPGA	54
Figura 13 - Processo de leitura de dados via interrupção.....	56
Figura 14 - Fluxo do processo de configuração da FPGA.....	57
Figura 15 - <i>Statistical profiler</i> apresentando o percentual de execução de diversas funções [14]....	59
Figura 16 – Visualização da aba Canvas do SmartDesign.....	66
Figura 17 – Núcleo dos processadores da família Blackfin [20].....	68
Figura 18 – Ambiente de desenvolvimento do VisualDSP++5.0.....	71
Figura 19 – Análise do tempo de execução do codificador x264 [29].....	75
Figura 20 – Modelo tradicional de computação da SAD.....	76
Figura 21 – <i>Statistical profiling</i> do código de referência que calcula a SAD.....	77

Figura 22 – Cálculo da SAD na FPGA sob controle do DSP.....	80
Figura 23 – Integração dos Módulos.....	82
Figura 24 – Ciclos de <i>clock</i> para o cálculo de um quadro de imagem.....	83
Figura 25 – Percentual de redução de ciclos de <i>clock</i>	84
Figura 26 – Capacidade de Expansão em MIPS.....	85
Figura 27 – Tempo de <i>idle</i> do DSP.....	86
Figura 28 – Eficiência da Arquitetura Proposta.....	87
Figura 29 – Eficiência média da arquitetura proposta.....	88

Lista de Tabelas

Tabela I – Área ocupada pelos SoftProcessors em diferentes FPGAs.....	39
Tabela II – Frequência máxima de operação dos SoftProcessors.....	40
Tabela III – Sinais de controle para a comunicação entre DSP e FPGA.....	48
Tabela IV – Comparação de custo entre diferentes DSPs de ponto fixo.....	67
Tabela V – Relação entre CLK e SCLK[20].....	70

Lista de Abreviações

AMC	- <i>Asynchronous Memory Controller</i>
API	- <i>Application Programming Interface</i>
CLB	- <i>Configurable Logic Blocks</i>
CLK	- <i>Core Clock</i>
CPI	- <i>Cycles per Instruction</i>
CPU	- <i>Central Processing Unit</i>
DAB	- <i>DMA Access Bus</i>
DCT	- <i>Discrete Cossine Transform</i>
DMA	- <i>Direct Memory Access</i>
DSP	- <i>Digital Signal Processor</i>
EAB	- <i>External Access Bus</i>
EBIU	- <i>External Bus Interface Unit</i>
FFT	- <i>Fast Fourier Transform</i>
FPGA	- <i>Field Programmable Gate Array</i>
FPS	- <i>Frames per Second</i>
IP	- <i>Intelectual Property</i>
JPEG	- <i>Joint Photographic Experts Group</i>
MAD	- <i>Mean Absolute Difference</i>

MAE - *Mean Absolute Errors*

MIPS - *Million Instructions per Second*

MSA - *Micro Signal Architecture*

PAB - *Peripheral Access Bus*

PLL - *Phase Locked Loop*

PPI - *Peripheral Parallel Interface*

RISC - *Reduce Instruction Set Computer*

SAD - *Sum of Absolute Difference*

SAE - *Sum of Absolute Errors*

SBTVD - *Sistema Brasileiro de Televisão Digital*

SCLK - *System Clock*

SIMD - *Single-Instruction, Multiple-Data*

SPI - *Serial Peripheral Interface*

SSEL - *System Select*

UCC - *Unidade Central de Controle*

UCp - *Unidade de Coprocessamento*

ULA - *Unidade Lógica Aritmética*

USB - *Universal Serial Bus*

VCO - *Voltage Controlled Oscillator*

VHDL - *VHSIC Hardware Description Language*

VHSIC - *Very High Speed Integrated Circuit*

1. Introdução

Com o objetivo de maximizar o desempenho computacional em sistemas embarcados, este trabalho propõe o uso cooperativo entre DSP (*Digital Signal Processor*) e FPGA (*Field Programmable Gate Array*), explorando o que esses dispositivos possuem de mais vantajoso para a resolução de algoritmos de processamento digital de sinais.

1.1 Motivação

Em muitas bibliografias podem-se encontrar diversos algoritmos propostos para a resolução dos mais diversos problemas em redes de telecomunicações. Entretanto, muitos dos quais apresentam vantagens teóricas têm a sua implementação prática dificultada devido à grande complexidade computacional envolvida, o que resulta na necessidade do uso de arquiteturas computacionais de elevado custo e isso pode inviabilizar o desenvolvimento de um determinado sistema.

A proposta de desenvolvimento apresentada neste estudo consiste em unir em uma única arquitetura computacional DSP e FPGA, explorando o que esses dispositivos possuem de mais eficaz. Constituindo-se assim, uma estrutura computacional de elevado desempenho, objetivando sempre a melhor relação custo-benefício, sendo que ao invés de se optar por processadores de elevado custo, pode-se utilizar uma FPGA para auxiliar o processamento de rotinas altamente complexas para um DSP.

A utilização de circuitos dedicados em FPGA para o processamento de algoritmos em paralelo, apresenta-se em muitos casos mais eficiente que a implementação de códigos seriais escritos em DSP, possibilitando a execução de tarefas com a utilização de menos ciclos de *clock*, o que em muitos casos, permite um processamento mais rápido de diferentes algoritmos.

Para ilustrar a viabilidade da execução de uma determinada rotina computacional, foi construída uma análise baseada em [2] onde uma FFT (*Fast Fourier Transform*) é calculada

primeiramente em DSP e depois em FPGA. Dessa forma, é possível comparar a quantidade de ciclos necessários para a execução do código e o tempo total gasto ao final da execução em ambos dispositivos.

Na análise, foi considerado o cálculo de uma FFT de diferentes tamanhos, *radix-2* e com um sinal de entrada em representação complexa. A escolha da FFT para descrever a motivação deste estudo deve-se ao fato desse algoritmo ser amplamente aplicado nos sistemas de telecomunicações, porque permite a análise do espectro, onde um sinal discreto expresso no domínio do tempo pode ser representado no domínio da frequência.

Assim, para mensurar a quantidade de ciclos de *clock* gastos pelo DSP, foi utilizado o processador ADSP-BF533 da Analog Devices [20], contendo um código composto de funções otimizadas e escritas em Assembly, as quais foram disponibilizadas pelo próprio fabricante do dispositivo e que exploram o máximo desempenho do processador.

Para uma comparação justa do desempenho com a FPGA, considerou-se o mesmo algoritmo calculado pelo DSP, reescritos em Linguagem Descritiva de *Hardware* (HDL) e disposto como um IP (*Intellectual Property*) Core da Actel [5].

Os resultados são apresentados pela Figura 1, a qual relaciona a quantidade de ciclos com o aumento da complexidade computacional do algoritmo.

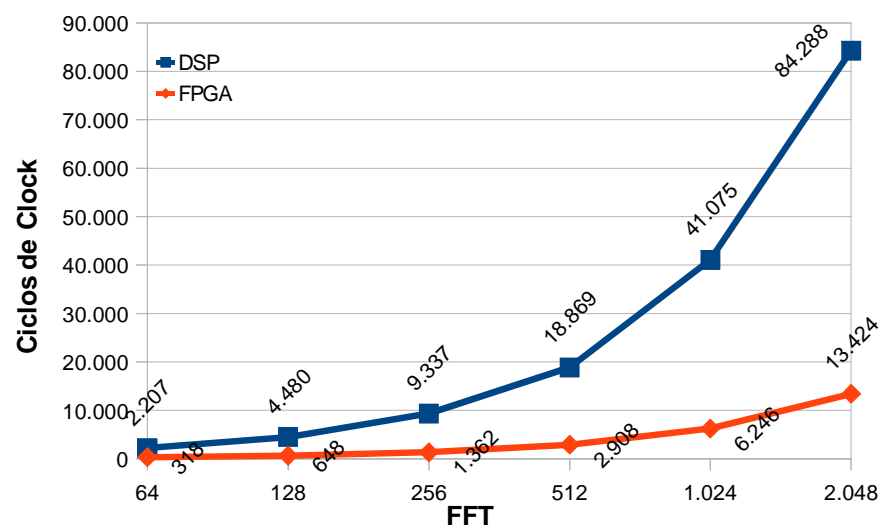


Figura 1 – Ciclos de *clock* para o cálculo de uma FFT

Conforme ilustrado pela Figura 1 é possível notar a diferença entre a quantidade de ciclos necessários para o cálculo do mesmo algoritmo nos diferentes dispositivos DSP e FPGA. A quantidade de ciclos da FPGA ficou sempre abaixo da quantidade exigida pelo DSP. Isso se deve ao fato de que a FPGA utiliza o processamento paralelo, enquanto que o DSP realiza um número limitado de operações por ciclo de *clock* e necessita gerenciar todos os seus registros internos de armazenamento temporários, resultando em um maior consumo de recursos computacionais à medida que a complexidade do código aumentou [3].

O valor médio em ciclos de *clock* para o DSP foi de 34.560 e para a FPGA foram de apenas 3.479 ciclos, ou seja, um valor cerca de 10 vezes menor.

Entretanto, deve-se considerar que os dispositivos tendem a operar em frequências diferentes. Portanto, uma análise baseada apenas no total de ciclos não permite uma ampla interpretação de qual dispositivo se apresentou como o mais eficiente, por esse motivo é necessário também avaliar o tempo necessário para a execução da rotina computacional.

Dessa forma, a Figura 2 apresenta os dados que permitem avaliar o tempo de processamento gasto pelo DSP e pela FPGA operando em diferentes frequências.

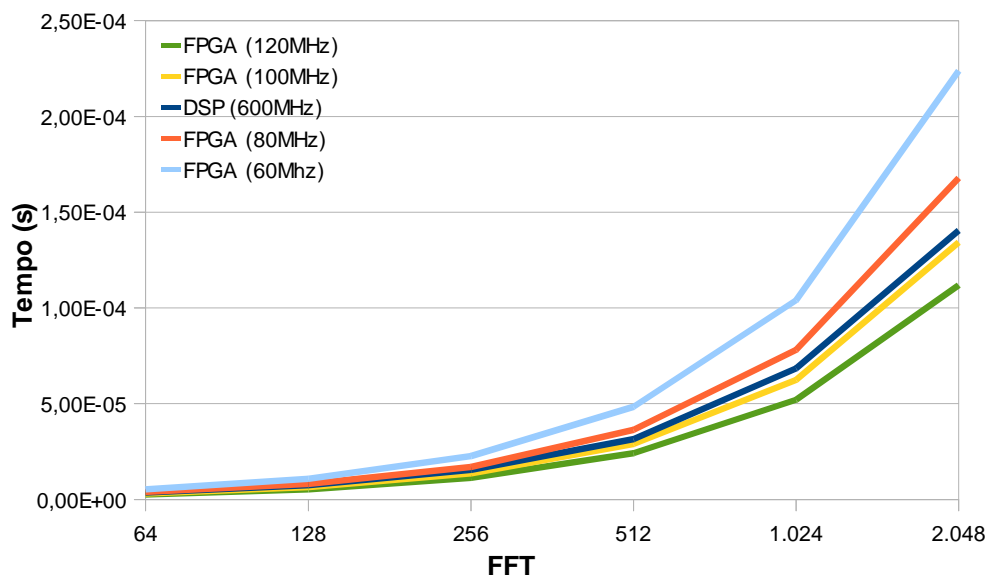


Figura 2 – Tempo de processamento de uma FFT de diferentes tamanhos

Geralmente a frequência máxima de operação de uma FPGA é determinada pela complexidade de processamento executado, por esse motivo, a Figura 2 considerou diferentes velocidades de processamento para a FPGA, ao passo que o DSP escolhido foi configurado para operar em sua capacidade máxima.

Ainda considerando a Figura 2, observa-se que mesmo o DSP consumindo mais ciclos de *clock* que a FPGA em todos os estágios do cálculo da FFT, ele conseguiu concluir o processamento mais rápido quando a FPGA estava configurada para operar nas frequências de 60 e 80MHz.

Entretanto, no instante em que a FPGA operou nas frequências de 100 e 120MHz, ela conseguiu ser mais veloz que o DSP. Ou seja, no caso ilustrado, atuando em uma velocidade 5 vezes menor que o DSP, ainda assim, a FPGA pôde realizar o cálculo da FFT em um tempo médio 23% mais rápido.

Dessa forma, baseando-se na abordagem citada é difícil afirmar de forma generalizada que um dispositivo é superior ao outro, porque dependendo da forma como as diversas variáveis envolvidas se comportam, um dispositivo pode obter melhor desempenho do que o outro e vice-versa [4]. Por esse motivo, este estudo propõe um ambiente de cooperativismo em que a FPGA processa em paralelo os dados de maior custo operacional para o DSP, permitindo o aumento da quantidade de instruções processadas por segundo, elevando o desempenho computacional do sistema, consolidando assim uma arquitetura híbrida reconfigurável e de excelente relação custo-benefício.

1.2 Objetivo

O objetivo deste estudo consiste basicamente nos seguintes:

- Desenvolver uma proposta de arquitetura computacional híbrida para processamento digital de sinais;
- Propor um modelo matemático para mensurar a eficiência global do particionamento dos códigos;

- Estruturar um ambiente de sinergia, explorando o que os dispositivos DSP e FPGA possuem de mais eficiente;
- Destacar as vantagens de se processar determinada função com a utilização de circuitos dedicados e reprogramáveis;
- Apresentar por meio de simulações o potencial da utilização dos dispositivos FPGA como coprocessadores para a execução de funções críticas.

1.3 Metodologia

Para a realização de uma proposta contextualizada ao cenário de processamento digital de sinais, foi realizada uma extensa pesquisa bibliográfica, a qual permitiu realizar o comparativo entre diversos estudos que propõe a interconexão entre DSP e FPGA.

Analisando diferentes fontes, foi possível reunir as características mais vantajosas e viáveis em um única estrutura computacional, além de propor algumas melhorias sistêmicas em relação à outros trabalhos.

Para validação deste estudo, são utilizados mecanismos de simulação e implementação prática para mensurar a eficiência do processamento descentralizado de algoritmos de processamento digital de sinais.

Primeiramente, são apresentados quais são os custos computacionais para a execução de determinado algoritmo em DSP e após identificado os pontos críticos do processo é proposto um sistema de particionamento de código que tende a reduzir o tempo total de processamento.

1.4 Estrutura do Trabalho

Para uma melhor exposição do conteúdo, este estudo está dividido em seis capítulos.

O Capítulo 1 faz uma introdução ao trabalho desenvolvido, contextualizando o leitor ao cenário abordado, enfatizando os objetivos e metodologias empregadas.

No Capítulo 2 é realizada uma abordagem generalizada sobre os dispositivos DSP, apresentando as principais características desses processadores. De forma semelhante, o Capítulo 3 apresenta os dispositivos FPGAs destacando as suas características mais relevantes.

O Capítulo 4 descreve a arquitetura proposta neste estudo que une DSP e FPGA em uma única estrutura computacional. No mesmo Capítulo também é descrito o modelo matemático proposto, capaz de mensurar a viabilidade do particionamento de códigos em sistemas distribuídos.

Para validação da arquitetura sugerida foi elaborado um cenário de testes para o particionamento de um algoritmo de codificação de vídeo adotado pelo padrão H.264/AVC (*Advanced Video Coding*), o qual normatiza o método de compressão de vídeo digital em alta definição empregado nos principais sistemas de televisão digital do mundo, dentre eles, o SBTVD (Sistema Brasileiro de Televisão Digital).

As simulações e resultados do experimento são apresentados no Capítulo 5, sendo que no Capítulo 6 são realizadas as conclusões e considerações finais sobre este trabalho.

2. DSP - *Digital Signal Processor*

O processamento digital de sinais é utilizado para transformar ou manipular um determinado sinal analógico ou digital. Esse tipo de operação pode ser aplicado nos mais diferentes tipos de cenários, tais como: telecomunicações, robótica, áudio, vídeo, processamento de sinais biomédicos, dentre outros [11].

Em meados de 1970, surgiram os Processadores Digitais de Sinais, ou DSPs (*Digital Signal Processors*) que são arquiteturas computacionais capazes de calcular em um único ciclo de *clock* uma operação de soma e multiplicação. Essa capacidade, juntamente com o modelo de arquitetura Harvard, representou um grande avanço ao se comparar com o modelo de arquitetura computacional do tipo Von-Neuman, a qual constitui a maioria dos microprocessadores atuais [11].

Com o objetivo de apresentar maiores detalhes sobre esses processadores, este Capítulo descreve de forma resumida alguns dos itens que compõem os DSPs, são eles: arquitetura, núcleo, subsistema de memória e requisição de serviços de interrupção.

2.1 Descrição da Arquitetura

Em geral, os DSPs são processadores programáveis destinados principalmente para aplicações embarcadas que possuem limitações de recursos, como, bateria, pouca memória disponível e espaço físico reduzido. Eles são facilmente encontrados em modems, telefones celulares, smartphones, *set-top boxes*, receptores, transmissores de sinais, dentre outros [38].

Além do mais, esses processadores são geralmente baseados em um arquitetura RISC (*Reduced Instruction Set Computing*) com algum acessório CISC (*Complex Instruction Set Computer*).

Em uma máquina RISC, as instruções computacionais são reduzidas e essa característica aliada a um reduzido número de circuitos internos, permitem aos processadores operar em frequências elevadas.

Já, as características CISC, quando empregadas nos DSPs são dedicadas para os cálculos de algumas operações específicas, como, convolução e divisão. Em geral, as características CISC tendem a melhorar o desempenho computacional do processador ao passo que as características RISC permitem a utilização do pipeline de forma mais eficiente.

Dessa forma, unindo características distintas, os processadores digitais de sinais são caracterizados por terem um conjunto de instruções específicas e uma arquitetura especial que permite a execução de operações matemáticas complexas de forma bastante eficiente.

Com a evolução do desenvolvimento tecnológico, esses processadores passaram a adquirir funcionalidades ainda mais sofisticadas, dentre as quais pode-se destacar a multiplicação numérica em ponto flutuante, a utilização de controladores de acesso direto à memória, os quais permitem a comunicação de dados com reduzida latência, operando de forma independente do núcleo do processador [11 e 38].

Geralmente, a arquitetura de um DSP é constituída por Núcleo, Memória e Periféricos. Para um melhor entendimento, as subseções a seguir descrevem as principais características desses componentes.

2.2 Núcleo

Devido ser possível encontrar em algumas bibliografias pequenas variações sobre a definição de Núcleo, será considerado nesta discussão, assim como em [32 e 37], que o Núcleo do DSP é constituído basicamente por um *datapath*, que por sua vez engloba, a MAC (Unidade de Multiplicação e Acumulação) e a ULA (Unidade Lógica Aritmética), além de outras unidades específicas de cada arquitetura.

As funções lógicas são processados no *datapath*. Ele é capaz de acessar dados em um subsistema de memória como um operando e retornar as informações já processadas para outras regiões de memória.

De forma semelhante às arquiteturas RISC, muitos registros de uso geral podem ser considerados como componentes de armazenamento de primeiro nível em

um arquivo de registro no *datapath*, que por sua vez é controlado pelo *controlpath*, enquanto as operações aritméticas e funções lógicas são realizadas pela MAC e pela ULA, respectivamente.

2.2.1 MAC – Unidade de Multiplicação e Acumulação

Como dito anteriormente, a unidade de multiplicação e acumulação (MAC) é um bloco de computação aritmética situada dentro do *datapath*.

A MAC realiza funções aritméticas incluindo multiplicações, convoluções, tais como: FIR, IIR e correlações. Nessa unidade é possível realizar cálculos numéricos com maior precisão que a largura do barramento de memória pode suportar. Por exemplo, nos processadores da família Blackfin da Analog Devices é possível processar dois dados numéricos de 16 bits, sendo que o barramento de memória suporta apenas 16 bits por uso do canal [22].

Geralmente, a precisão da MAC é de até duas vezes a largura do barramento de memória. Ela pode operar em dois modos distintos: interativo ou em etapa única.

Em modo interativo, a memória fornece os dados para a unidade MAC, já em modo de etapa única, os dados são fornecidos pelo registradores de arquivo e registradores de acumulação.

De acordo com sua funcionalidade, o processamento dentro da MAC pode ser dividida nas seguintes etapas:

- Preprocessamento para multiplicação, preparando os operandos para o multiplicador e para a acumulação;
- Multiplicação;
- Preprocessamento para acumulação;
- Acumulação e armazenamento dos resultados;
- Pós-processamento e sinalização, indicando que a operação foi concluída.

A Figura 3 apresenta um modelo estrutural, dos registros de multiplicação e acumulação, onde os operandos A e B, se referem às interfaces de aquisição de dados.

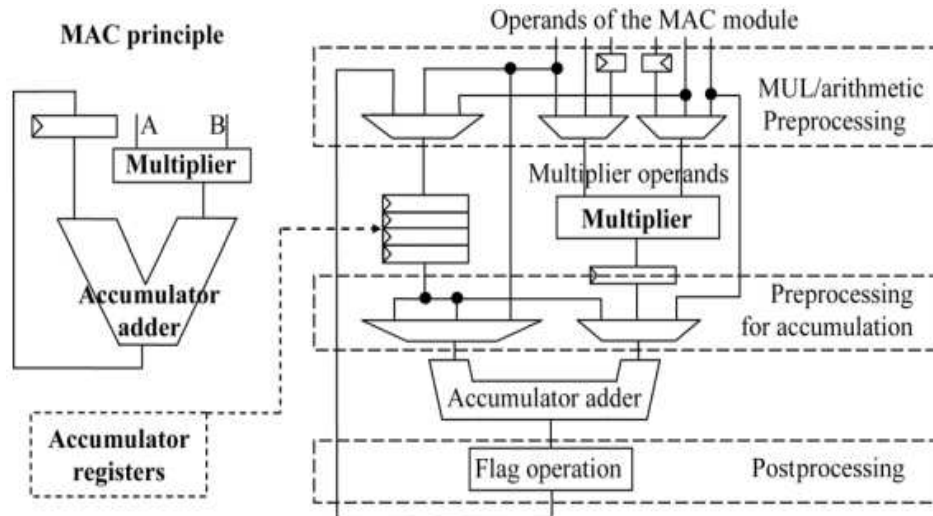


Figura 3 - Visão simplificada da unidade MAC [31]

Dentro da MAC, os operandos alimentam os circuitos de pré-processamento e os resultados do processamento são recolhidos pelos circuitos de pós-processamento e armazenados nos registros de acumulação.

2.2.2 ULA – Unidade Lógica Aritmética

A Unidade Lógica Aritmética (ULA) é uma unidade de *hardware* capaz de realizar operações aritméticas, lógicas ou de deslocamento

Geralmente os resultados produzidos pela ULA são armazenados em um acumulador. Entretanto, algumas instruções são escritas diretamente em memória. Na maioria dos DSPs cada operação da ULA pode ser realizada em apenas um ciclo de *clock*.

Ao contrário da MAC que é utilizada para computação interativa, a ULA pode ser um módulo de *hardware exclusivo*, o que é o mais comum, ou ainda ela pode ser parte de um *hardware* dentro da unidade MAC.

A ULA gera sinais capazes de controlar o fluxo de execução do programa e essas sinalizações podem ser usadas como entrada para outras operações lógicas ou aritméticas. Por exemplo, o sinal de saída de uma operação pode sinalizar para a entrada da próxima instrução, isso garante menor latência entre os processos.

2.3 Memória

Um subsistema de memória inclui os dispositivos físicos de memória, geradores de endereços, circuitos e barramentos. Entretanto, apenas o gerador de endereços é alocado dentro do núcleo do processador.

Geralmente, o subsistema de memória interna de um processador digital de sinais ocupa a maior parte da área do silício. E, devido os custos de memória interna serem elevados, existe a necessidade da utilização de banco de memória externos ao chip, os quais possuem maior latência de acesso.

Assim, para se obter um ganho na eficiência computacional é aconselhável manter os programas atualmente em execução e os dados mais frequentemente acessados na região de memória interna do chip.

Esse controle de definição de alocação de memória, também conhecido como projeto hierárquico do subsistema de memória, tem sido um dos grandes desafios em pesquisas relacionadas à sistemas embarcados. Por esse motivo, nos DSPs modernos, como o Blackfin 533 da Analog Devices [20], o usuário pode redefinir o tamanho das regiões de memórias internas para a alocação dos trechos de códigos que são acessados constantemente.

É importante ressaltar que o projeto do subsistema de memória influencia diretamente em itens como, desempenho, custo e consumo de potência. Logo, são parâmetros fundamentais considerados durante o desenvolvimento da microarquitetura, a capacidade de otimizar o desempenho do sistema utilizando o paralelismo em nível de instrução (VLIW) e o uso de técnicas eficientes de pipeline, ambas tem a finalidade de ser obter um menor consumo de área e de energia.

2.3.1 Acesso Direto à Memória

Acesso Direto à Memória, ou DMA é um método para troca de dados entre memórias sem a utilização do processador para acessar os dados [39].

A unidade de DMA é um periférico de *hardware* dentro do núcleo do DSP, sendo capaz de suportar a transferência de dados entre memórias internas ou externas. Os dados podem ser enviados diretamente de um dispositivo conectado (como uma unidade de disco ou externa memória) para outras posições de memória no espaço de endereço DSP. Dessa forma, o núcleo do processador fica totalmente livre para se dedicar ao processamento de dados [1].

Normalmente a transferência de blocos de dados leva um longo tempo, entretanto, pode-se utilizar o mecanismo de DMA para deslocar blocos de dados de uma determinada região sem impactar as rotinas que estão sendo processadas.

2.4 Requisição de Serviço de Interrupção

Interrupções são sinais oriundos de um determinado periférico que implica na troca de contexto, isso significa que o processador passa a processar imediatamente um processo de *hardware* ou *software*, com base no atendimento de uma requisição de serviço de interrupção, ou ISR (*Interrupt Service Request*) [39].

As interrupções são mecanismos que informam à CPU que um periférico específico precisa ser atendidos imediatamente. Sendo que, após atender à uma ISR a CPU retorna imediatamente ao trecho de código que estava sendo processado anteriormente.

Uma ISR é muito semelhante a uma função. Pois ela também utiliza uma pilha para armazenar o endereço de retorno, exatamente como uma função qualquer. No entanto, além de também poder ser gerada por software, uma das principais diferenças é que uma ISR pode ser acionada por um sinal elétrico enviado de um periférico diretamente para a CPU, ao invés de ser acionada por uma instrução do processador, as quais necessitam de maior tempo para serem detectadas.

2.5 Pipeline

Os atuais DSPs geralmente são constituídos de uma arquitetura composta de pelo menos um arranjo de multiplicação, como, por exemplo, 16x16, 24x24 ou 32x32 bits em ponto fixo ou flutuante. Eles utilizam uma estrutura de pipeline eficiente, devido à grande parte dos

algoritmos de processamento de sinais envolverem rotinas de multiplicação e soma de forma exaustiva.

A estrutura de pipeline permite o alcance de elevadas taxas de processamento, limitadas apenas à complexidade natural do código e as limitações do processador, sem gerar nenhuma outra latência ao processo.

E para exemplificar, lista-se a seguir o pipeline da MAC que é tipicamente composto de quatro estágios que ocorrem simultaneamente, são eles:

1. Decodificação do comando;
2. Carregamento dos operandos nos registradores;
3. Execução de uma multiplicação e armazenamento do produto e;
4. Soma/Acumulação do produto.

3. FPGA - *Field Programmable Gate Array*

Antes do surgimento da lógica programável, os circuitos lógicos eram geralmente desenvolvidos com a utilização de portas lógicas aplicadas em circuitos integrados.

Entretanto, com a evolução tecnológica, surgiram as FPGAs (*Field Programmable Gate Array*), as quais se caracterizam por um *hardware* capaz de ser reprogramado sempre que houver necessidade [7].

Os dispositivos FPGAs possuem como principal característica a capacidade de reconfiguração das suas funções internas, isso permite possíveis alterações de projeto conforme as definições feitas pelo usuário.

Neste Capítulo é apresentada uma visão geral dos conceitos fundamentais da tecnologia das FPGAs, destacando a sua arquitetura e as etapas de desenvolvimento de um sistema. Também é realizada uma abordagem sobre os *SoftProcessors*, os quais são caracterizados por módulos de processamento disponíveis nas FPGAs modernas.

3.1 **Arquitetura**

Um dispositivo FPGA, consiste em milhões de transistores conectados capazes de executar funções lógicas que vão desde uma simples adição ou soma, até complexos filtros digitais [40].

FPGAs são circuitos programáveis compostos por um conjunto de células ou blocos lógicos, alocados em forma de uma matriz dentro de um único chip. Sendo que as funcionalidades e os recursos internos do dispositivo são configuráveis por *software* [1 e 7].

Em geral, uma FPGA é composta por três elementos lógicos, denominados como: *I/O Interface*, CLBs (*Configurable Block Logic*) e PIPs (*Programmable Interconnection Points*) [40 e 7]. A Figura 4, ilustra a disposição interna desses elementos .

I/O interface é o meio pelo qual os dados são enviados de uma lógica interna da FPGA para o ambiente externo e vice-versa. Essa interface pode operar de forma unidirecional ou bidirecional.

Os CLBs são blocos lógicos configuráveis utilizados para a concepção das lógicas internas da FPGA. Eles são constituídos geralmente por lógicas combinacionais entre registradores, MUX (multiplexadores), LUT (*Look-up Tables*), SRL (*Shift Register LUT*) e RAM (*Random Access Memory*).

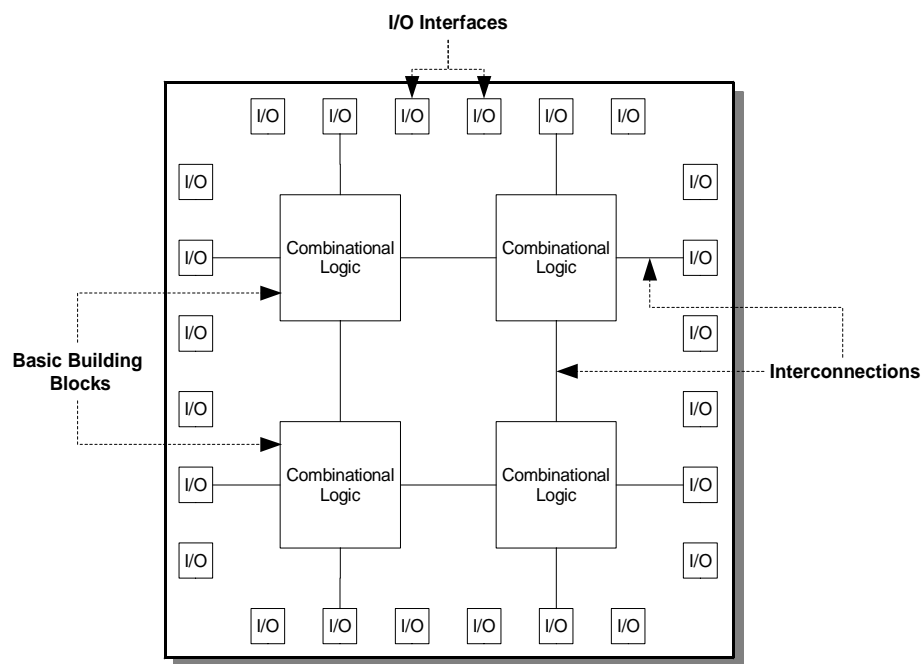


Figura 4 - Arquitetura genérica de uma FPGA [40]

Os PIPs são pontos de interconexões programáveis responsáveis pela comunicação entre os blocos internos do dispositivo. Eles ainda formam uma rede que interliga os CLBs à interface de entrada e saída de dados, com a finalidade de computar a lógica definida pelo usuário durante a etapa de desenvolvimento dos circuitos.

No caso na família Virtex da Xilinx, a menor unidade lógica configurável de uma FPGA é denominada *slice*. Os slices são encontrados dentro de cada CLB e em cada *slice*, por sua vez, existem as LUTs.

O *slice* é bastante versátil e pode ser configurado para operar como uma LUT de quatro entradas e uma saída, ou ainda, como memória RAM de 16 bits ou registradores de deslocamento de 16 bits.

Em operações como LUT, os recursos adicionais, como, *flip-flops*, multiplexadores, e portas lógicas, podem ser utilizados para implementar funções booleanas, multiplicadores e somadores com palavras de comprimento variável. Esses recursos podem ser utilizados para implementar diferentes operações .

3.2 Etapas de Desenvolvimento

As etapas de desenvolvimento de um sistema em FPGA difere do modelo de programação utilizado em DSPs, ou em processadores de propósito geral. Isso se deve ao fato de que em FPGA utiliza-se o conceito de descrição de circuitos para determinar uma lógica operacional.

Embora diversos fabricantes de FPGA possam dividir ou resumir as etapas de geração de códigos de diferentes maneiras, os conceitos relacionados à elas são basicamente os mesmos, independente do modelo de dispositivo utilizado.

De forma generalizada, as fases de concepção de circuito em FPGA, consistem basicamente em cinco etapas, são elas: *Design*, Síntese, Implementação, Simulação e Programação.

O fluxograma de desenvolvimento é ilustrado pela Figura 5, sobre a qual se pode observar que todas as etapas, exceto a de Programação, são bidirecionais. Ou seja, sempre que for necessário, o usuário pode retornar a uma etapa anterior e realizar as alterações de projeto que julgar pertinente.

Outro aspecto relevante é o fato de que a etapa de simulação pode ser acessada a partir das fases de Design, Síntese ou Implementação, isso permite simular os circuitos gerados a partir de diferentes perspectivas de desenvolvimento.

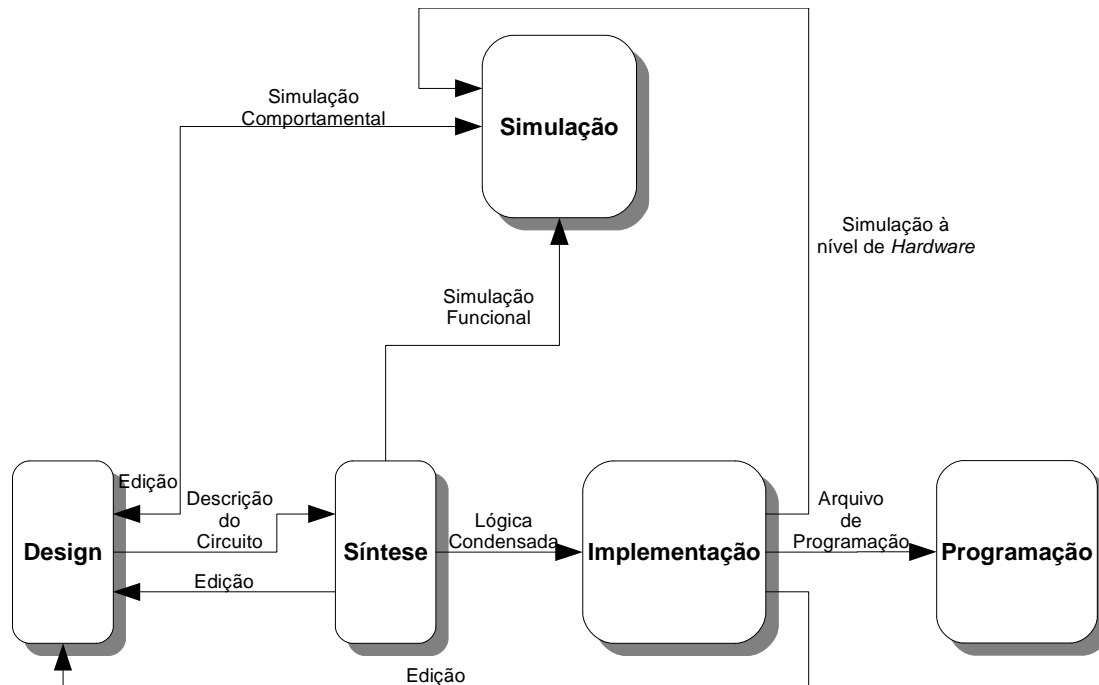


Figura 5 - Etapas de desenvolvimento de circuitos em FPGA [40]

Por exemplo, quando é realizada a simulação com os arquivos gerados pelo estágio de implementação, será possível observar os comportamentos físicos específicos do *hardware* utilizado, esse ato não permitido pela etapa de Design, a qual permite verificar apenas o comportamento lógico dos módulos desenvolvidos.

Para um melhor entendimento, as subseções seguintes descrevem as principais características do processo de desenvolvimento de um sistema em FPGA.

3.2.1 Design

A primeira etapa denominada *design* é responsável pela criação do arquivo utilizado para descrever o circuito da FPGA. Para a construção do circuito é utilizado linguagens descritiva de *hardware*, também conhecidas como HDL (*Hardware Description Language*) que determinam o comportamento de todo o sistema.

Atualmente existem ferramentas de prototipagem rápida que permitem ao desenvolvedor abstrair as camadas de baixo nível sendo capaz de descrever um circuito

utilizando apenas blocos lógicos, os quais podem encapsular funcionalidade mais complexas ou mesmo outros blocos.

Essas novas ferramentas permitem uma maior agilidade de programação, garantido um menor tempo de desenvolvimento e facilitando a manutenção dos códigos gerados.

Hoje, já é possível programar FPGAs utilizando a linguagem C, o que facilita a migração de programadores habituados com o modelo de desenvolvimento estruturado, além de permitir uma melhor legibilidade e interpretação dos códigos [33].

Entretanto, quando se utiliza a linguagem C para a descrição dos circuitos, é necessário uma etapa intermediária do processo de compilação que converte o código gerado para o seu correspondente em HDL, e só então é gerado um arquivo binário capaz de programar o dispositivo, conforme apresenta a Figura 6.

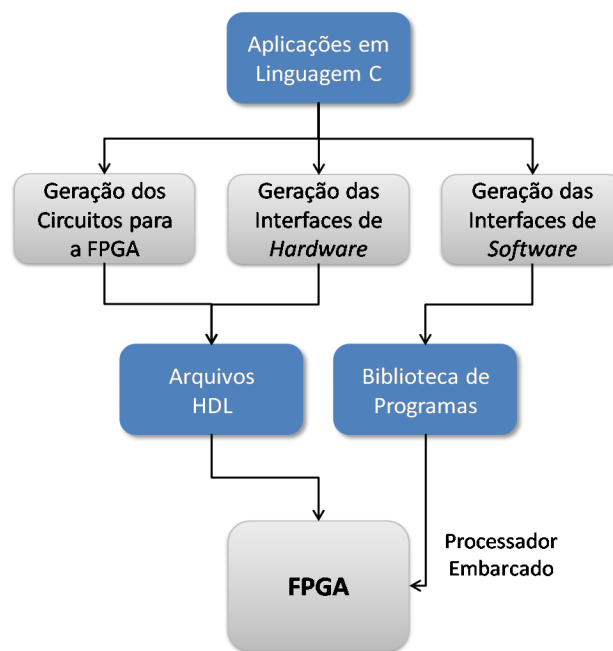


Figura 6 - Fluxograma da descrição de circuitos em linguagem C, adaptado de [33]

Assim como os modelos de prototipagem rápida, esse processo de programação estruturada também tende a reduzir o tempo de desenvolvimento de um sistema em comparação a utilização dos métodos convencionais de programação.

No entanto, é importante destacar que quanto mais camadas de abstração existirem entre o desenvolvedor e o dispositivo, maiores são as chances de serem inseridas redundâncias de compilação o que podem acarretar em uma maior ocupação de área ou no uso excedente de elementos lógicos.

3.2.2 Síntese

A etapa de desenvolvimento citada anteriormente é considerada como uma implementação de alto nível. Isso, porque não são associadas nenhuma conexão dos elementos físicos da FPGA.

Na etapa de síntese são associadas ao desenvolvimento de alto nível os recursos disponíveis pelo dispositivo. Durante esse processo, caso o usuário tenha utilizado mais recursos que o disponibilizado, será gerado um erro que impossibilita de prosseguir na construção dos circuitos, até que se corrija a lógica realizada durante a etapa de *design*.

Após ter concluído a etapa de sintetização da lógica elaborada é possível prosseguir para a fase de implementação.

3.2.3 Implementação

A implementação se refere ao processo de mapeamento da síntese gerada para um dispositivo específico. Nessa fase os recursos internos da FPGA são interconectados fisicamente e associados aos pinos da interface de entrada e saída.

Durante esse processo o *layout* físico é concebido. Esta é a fase final de desenvolvimento capaz de manipular a disposição interna dos recursos da FPGA antes dos circuitos serem escritos no dispositivo.

3.2.4 Simulação

A simulação é o processo em que o desenvolvedor pode aplicar estímulos nos sinais de entrada com a finalidade de observar o comportamento dos sinais de saída.

Os sinais resultantes da simulação podem ser analisados de diversas formas, desde o formato binário, hexadecimal ou até mesmo de forma gráfica onde é possível visualizar as curvas resultantes do experimento.

A etapa de simulação é uma das fases em que se requer maior atenção por parte do desenvolvedor. Nesse estágio é importante que sejam realizados diversos testes que possam validar o comportamento do sistema.

Uma vez constatado que os circuitos gerados se comportam da forma esperada, pode-se prosseguir para a programação dos dispositivos.

3.2.5 Programação

A última etapa no desenvolvimento de código em FPGA, consiste na programação ou configuração do dispositivo por meio de um *bit-stream* que é gerado a partir da lógica implementada.

O *bit-stream* pode ser carregado em uma memória externa volátil ou não-volátil. Entretanto, algumas FPGAs possuem memórias internas e podem armazenar a sua própria configuração sem a necessidade de um dispositivo auxiliar.

Também é possível que um dispositivo não-volátil, como, as memórias PROMs (*Programmable Read-Only Memories*), estejam localizados em uma placa secundária, e a partir de um endereçamento de memória interno pode-se configurar uma FPGA de forma serial ou paralela.

3.3 SoftProcessors

Além dos avanços em capacidade, desempenho e custo, os fabricantes de FPGAs têm introduzido, no decorrer dos anos, cada vez mais recursos inovadores.

Um dos recursos que tem despertado a atenção dos desenvolvedores nos últimos anos são os processadores definidos por *software*, ou simplesmente, *SoftProcessors*, os quais se caracterizam como propriedades intelectuais, também conhecidas como IP (*Intellectual Property*)

Cores. Esses dispositivos são implementados a partir de lógicas primitivas disponíveis nas FPGAs [26, 37 e 7].

Comparando com os DSPs, esses processadores geralmente possuem menor capacidade de processamento. Além do mais, eles ocupam determinado espaço físico dentro das FPGAs e em muitos casos, a cada novo periférico inserido no processador, como, por exemplo, um controlador SDRAM, uma interface UART, dentre outros, ocorre um aumento no total de área necessária para o módulo.

A Tabela I, com base em [8 e 25] e apresenta alguns dos mais conhecidos *SoftProcessors* da fabricante Altera e a sua relação com alguns modelos de FPGAs que os suportam.

Para estimar a quantidade de elementos lógicos necessários, considerou-se o total da área ocupada pelo *SoftProcessor* somado com dois outros dispositivos amplamente utilizados em projetos de sistemas embarcados, como, o controlador SDR/SDRAM e o periférico Timer.

Tabela I – Área ocupada pelos *SoftProcessors* em diferentes FPGAs

FPGA				<i>SoftProcessor</i>		
Nome	Part-Number	Valor (\$)	Elementos Lógicos Disponíveis	Modelo	Elementos Lógicos Consumidos	Área Ocupada (%)
Cyclone	EP1C3T100C8N	10,7	2910	Nios II/f	2595	89,18%
				Nios II/s	2055	70,62%
				Nios II/e	1435	49,31%
Cyclone II	EP2C5T144C8N	12,8	4608	Nios II/f	2500	54,25%
				Nios II/s	1930	41,88%
				Nios II/e	1440	31,25%
Cyclone III	EP3C5E144C8N	12,8	5136	Nios II/f	2370	46,14%
				Nios II/s	1870	36,41%
				Nios II/e	1220	23,75%
Cyclone IV	EP4CE6F17C8N	14,94	6272	Nios II/f	2260	36,03%
				Nios II/s	1755	27,98%
				Nios II/e	1215	19,37%

A Tabela I, apresentou dados sobre os quais é possível constatar que para se utilizar o processador embarcado em FPGA, necessita-se de um número muito expressivo de área, isso inviabiliza o uso desses dispositivos em um *hardware* programável com reduzido número de elementos lógicos.

Mas, em muitas bibliografias, como em [26], afirmar-se que os *SoftProcessors* ocupam pequenas áreas dentro das FPGAs. Entretanto, o que se verifica na prática, para que esse fato realmente seja verdadeiro, será necessário um *hardware* programável de elevado custo o que muitas vezes inviabiliza o desenvolvimento do projeto.

Por exemplo, para que o Nios II/f, que ocupa cerca de 89% da área de uma FPGA Cyclone, passe a ocupar uma área de aproximadamente 12%, será necessário um dispositivo com 20.060 elementos lógicos, isso na prática representa uma FPGA no valor em torno de \$170,00 (cento e setenta reais), mais taxas de importação [25], ou seja, um valor muito acima ao apresentado na Tabela I, onde o valor médio não ultrapassa os \$12 (doze dólares americanos).

Para se ter uma melhor noção da velocidade de processamento desses dispositivos, uma outra abordagem se faz necessário. É preciso avaliar qual é a máxima frequência de operação desses processadores. Para tanto, a Tabela II relaciona esse dado com as diferentes famílias de FPGAs.

Tabela II – Frequência máxima de operação dos *SoftProcessors*

Processador	FPGA		Frequência Máxima (MHz)
	Família	Fabricante	
Arm CortexM1	Cyclone III	Altera	100
Nios II/f	Cyclone	Altera	135
Nios II/s	Cyclone	Altera	120
NiosII/e	Cyclone	Altera	175
MicroBlaze	Virtex II Pro	Xilinx	150
MicroBlaze	Spartan 3	Xilinx	85
Picoblaze – 8 Bit	Spartan 3	Xilinx	125
Picoblaze – 8 Bit	Spartan 6	Xilinx	128
Picoblaze – 8 Bit	Virtex 6 CXT	Xilinx	165

As máximas frequências consideradas na Tabela II, referem-se ao caso ótimo, ou seja, em implementações prática esse valor tende a ser muito menor, porque a velocidade de processamento varia conforme as configurações de projeto utilizada.

Por exemplo, caso seja desenvolvido um código otimizado para a redução de consumo de área, dependendo do projeto, podem ser construídos caminhos de pipelines muito grandes e isso irá reduzir a frequência máxima de operação do processador embarcado.

4. Arquitetura Computacional Proposta

As atuais aplicações de processamento de sinais em sistemas embarcados, além de alto desempenho, necessitam de grande flexibilidade e capacidade de reprogramação para atender os mais variados tipos aplicações. Visando atender à esse requisito, este estudo apresenta uma arquitetura híbrida baseada no uso cooperativo entre DSP e FPGA.

Conforme descrito no Capítulo 3, diversas FPGAs possibilitam integrar ao chip os processadores de propósito geral baseados em *software*, também denominados por *SoftProcessors*. Entretanto, esses dispositivos geralmente não possuem o potencial necessário para substituir completamente os processadores de maior capacidade, como os DSPs.

O Capítulo 3, ainda apresentou a Tabela I, que permite identificar o grande percentual de área e elementos lógicos consumidos pelos *SoftProcessors*, o que na maioria das vezes inviabiliza o uso desses processadores em um dispositivo FPGA de baixo custo.

Logo, a abordagem realizada neste estudo, consiste no uso cooperativo entre DSPs e FPGAs de baixo custo e que operando em conjunto permitem atingir um elevado desempenho computacional, resultando em uma arquitetura flexível dotada de alta capacidade de manutenção dos códigos gerados e com total de reprogramação dos circuitos de *hardware*.

Para uma melhor exposição do conteúdo, este Capítulo aborda o ambiente de sinergia promovido pela cooperação entre DSP e FPGA enfatizando o que esses dispositivos possuem de mais vantajoso [1]. Em seguida é descrita a arquitetura computacional proposta apresentando a sua descrição sistêmica. Por fim, também é sugerido um modelo matemático analítico capaz de mensurar o ganho efetivo total de um determinado trecho de código processado de forma descentralizada dentro da arquitetura elaborada.

4.1 Ambiente de Sinergia: Cooperação entre DSP e FPGA

A arquitetura proposta parte do princípio da utilização do DSP como um agente de elevado desempenho computacional capaz de executar cálculos numéricos com grande precisão, dotado de mecanismos eficientes de manipulação de dados e detecção de sinais por meio de interrupções geradas por *software* ou *hardware*.

Também é utilizada uma FPGA, que tem por finalidade auxiliar o processamento de rotinas demasiadamente complexas de forma paralela, sendo capaz de executar várias instruções em um único ciclo de *clock*.

Dessa forma, reunindo os dispositivos DSP e FPGA em uma única estrutura é possível constituir um ambiente de total sinergia e cooperação. As duas subseções seguintes sintetizam o potencial e as principais características desses dispositivos.

4.1.1 Processador Digital de Sinais - DSP

Em geral, Processadores Digitais de Sinais (DSPs) podem ser programados em linguagem Assembly, C ou C++. Eles permitem o desenvolvimento de códigos complexos com grande eficiência computacional, tendo em vista o fato de possuírem o seu próprio código de instruções para a execução de cálculos numéricos.

Esses processadores foram desenvolvidos considerando as principais rotinas computacionais em processamento digital, as quais consistem principalmente em adição, multiplicação e transferência de blocos de memória de forma bidirecional. Eles são compostos por um conjunto de instruções otimizadas para o processamento numérico de alta velocidade, sendo capazes de executar uma operação de multiplicação e soma simultaneamente em um único ciclo de *clock*.

Os Processadores Digitais de Sinais, geralmente possuem o salvamento automático de contexto, o que significa que o dispositivo pode atender à diversas chamadas assíncronas de interrupções com mínima latência.

Os DSPs também permitem o desenvolvimento de códigos complexos de forma muito mais rápida, acompanhados de uma maior facilidade de manutenção e atualização dos algoritmos, quando comparado às linguagens descritivas de *hardware*, tais como, VHDL e Verilog.

4.1.2 *Field Programmable Gate Array* - FPGA

A *Field Programmable Gate Array* (FPGA) tem como grande vantagem a capacidade de paralelismo e reconfiguração dos seus circuitos. A reconfiguração dinâmica, existente nos mais modernos dispositivos, permite que as alterações de roteamento dos circuitos da FPGA possam ocorrer durante o tempo de execução do sistema com base em algumas características ou fatores determinantes do sistema, como, por exemplo, a oscilação de um *bit stream* de entrada.

FPGAs são extremamente flexíveis, elas são constituídas de blocos programáveis chamados elementos lógicos e uma hierarquia de interconexões reconfiguráveis permitindo que todos esses componentes estejam fisicamente conectados. Pode-se configurar esses elementos lógicos para executar funções combinacionais complexas ou simplesmente portas lógicas, como OR, AND, XOR, dentre outras.

Atualmente, para o desenvolvimento de circuitos lógicos existem ferramentas de prototipagem rápida que permitem a programação da FPGA, abstraindo diversas camadas de baixo nível e de maior complexidade de desenvolvimento.

Esse mecanismo de programação viabiliza a construção de lógicas em tempo extremamente mais curto, quando comparado à programação em linguagem descritivas de *hardware*.

4.2 Descrição Sistêmica

Face às vantagens apresentadas nas Subseções 4.1.1 e 4.1.2 e se opondo à tentativa de afirmar que uma tecnologia é superior a outra, propõe-se a construção de uma arquitetura heterogênea e flexível, utilizando DSP e FPGA para a computação de algoritmos complexos atendendo aos requisitos de execução em tempo real.

Nesta proposta, foi possível estabelecer uma estrutura de alta capacidade computacional, dotada de reconfigurabilidade, a qual pode ser vista como um *hardware* dinâmico, em que as configurações dos seus circuitos implementados em *software*, são carregados ou retirados conforme a necessidade de uso, garantindo assim uma grande flexibilidade a todo o sistema [1].

Também foi utilizada a extensão do conceito de Unidade Lógica Aritmética (ULA) para paralelizar ao máximo as instruções que exigem maior esforço computacional. Além do mais, esta proposta de arquitetura utiliza duas unidades principais, a Unidade Central de Controle (UCC) e a Unidade de Coprocessamento (UCp), constituídas respectivamente, por um DSP e por uma FPGA.

A Figura 7, apresenta o diagrama de blocos da arquitetura proposta.

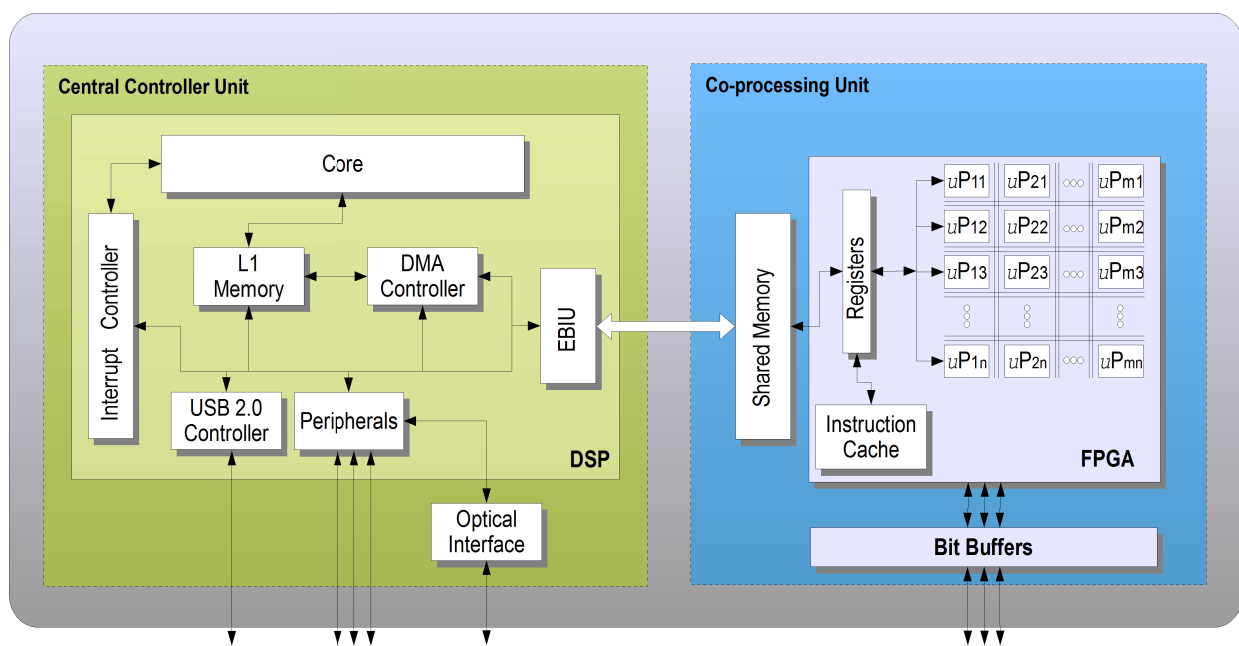


Figura 7 - Diagrama de blocos da arquitetura proposta

A Unidade Central de Controle é o núcleo do sistema. Ela determina o fluxo de execução dos dados processados pela UCp, controlando a taxa de processamento de forma adaptativa em relação à variação dos dados de entrada. Sendo ainda, rápida o suficiente para o atendimento das chamadas de interrupções geradas por *software* ou *hardware*.

A Unidade de Coprocessamento é um módulo constituído basicamente de uma FPGA. Nesse módulo coexistem micro blocos de processamento paralelo capazes de executar as operações sob controle do DSP. Para a FPGA, vários registradores de controle e *status* devem ser configurados para armazenarem as diferentes informações do processo de comunicação gerenciado pelo DSP.

De forma generalizada, na arquitetura proposta existem regiões específicas para aquisição de dados, por meio de interfaces, como: PPI (*Peripheral Parallel Interface*), SPI (*Serial Peripheral Interface*) e USB 2.0 (*Universal Serial Bus*).

Também foi analisada a possibilidade de expansão do sistema para processamento massivamente paralelo e distribuído, para isso, aconselha-se a utilização de uma interface óptica, a qual caracteriza o meio de comunicação mais eficiente para a conexão de diversos dispositivos que necessitam de menor latência e *jitter* durante a comunicação dos dados.

4.2.1 Interconexão dos Dispositivos

Um dos pontos críticos do sistema é a comunicação entre os módulos de processamento e coprocessamento. Visando diminuir os impactos negativos causados durante a transferência dos dados, como, reduzida taxa, complexidade de leitura e escrita, dentre outros. Foram analisadas diversas possibilidades de interconexão e chegou-se a conclusão que alternativa que representa a melhor relação de custo-benefício, consiste na adoção de um banco de memória compartilhado entre a Unidade Central de Controle e a Unidade de Coprocessamento.

O modelo de comunicação adotado, permite total independência dos processos. Sendo que a partir de um AMC (*Asynchronous Memory Controller*), geralmente já embutido nos DSPs modernos, pode-se atingir um *throughput* elevado, com mínima latência e *jitter*.

Valores teóricos revelam que operando em uma frequência de 133 MHz e utilizando um barramento de 16 bits, pode-se atingir até 2,128 Gbps. Essa taxa se mostra muito superior aos mais comuns tipo de periféricos disponíveis para comunicação, como, por exemplo, SPI e USB 2.0.

4.2.1.1 Sinais de Controle

Baseando-se nos processadores da família Blackfin da Analog Devices [20] e considerando a flexibilidade inerente aos *hardwares* programáveis, propõe-se um modelo de interconexão em que o DSP considera a FPGA como um banco de memória externa.

Esse mecanismo de interconexão faz com que a estrutura computacional proposta apresente uma relação custo-benefício ainda melhor, porque além de diminuir a complexidade de conexão, facilita o modo de desenvolvimento de códigos, porque todo o processo de comunicação se torna totalmente transparente ao usuário.

Considerando o controlador de memória assíncrona, a Figura 8 mostra os principais sinais compartilhados entre os dispositivos.

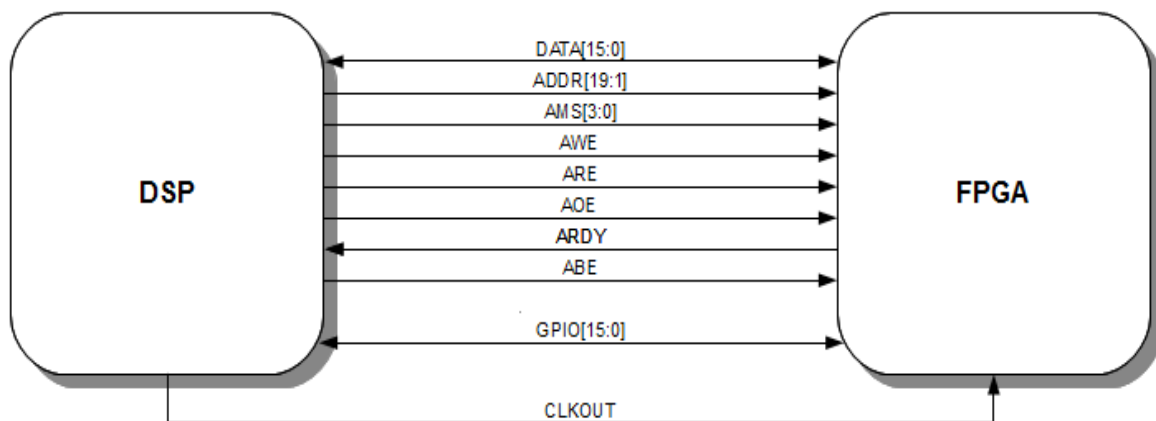


Figura 8 - Interconexão lógica entre DSP e FPGA

No projeto são basicamente 10 sinais necessários para compor a comunicação entre o DSP e a FPGA, para alguns desses sinais é preferível que eles sejam enviados com o auxílio de um *bit buffer* que tem por finalidade atuar como uma área de acesso intermediário aos sinais transmitidos ou recebidos de forma paralela. Esse dispositivo é capaz de operar em diferentes faixas de frequências, garantindo maior flexibilidade e estabilidade de operação ao barramento de acesso às interfaces.

O *bit buffer* permite ao sistema transmissor enviar os dados sem a preocupação se eles foram imediatamente lidos pelo receptor. Quando são transmitidos os bits de informações, o *bit buffer* os armazena em um deslocador (*shifter*) e permite que esses dados sejam acessados mesmo que em um período diferente do qual foram escritos.

Em outras palavras, a utilização do *bit buffer* se torna necessária devido os processos do DSP e da FPGA serem assíncronos, o que significa que em um determinado período, o agente emissor, DSP ou FPGA, podem enviar os dados com uma maior segurança de que eles serão capturados pelo outro processo, independente do ciclo que eles forem lidos.

A Tabela III, descreve os sinais utilizados para o acesso entre as unidades constituídas pelo DSP e FPGA.

Tabela III – Sinais de controle para a comunicação entre DSP e FPGA

Sinal	Diretividade (DSP como referência)	Descrição
DATA[15:0]	Entrada/Saída	Dados
ADDR[3:0]	Saída	Endereço do barramento de memória externa
$\overline{\text{AMS}}$	Saída	Seletor de memória assíncrona
$\overline{\text{AWE}}$	Saída	Ativa a escrita
$\overline{\text{ARE}}$	Saída	Ativa a leitura
$\overline{\text{AOE}}$	Saída	Ativa a saída dos dados
ARDY	Entrada	Indica que uma um dado está pronto para ser lido
$\overline{\text{ABE}}[1:0]$	Saída	<i>Byte Enable</i>
GPIOs	Entrada/Saída	Pinos de Propósito Geral
CLKOUT	Saída	Relógio de Controle

Devido às características do barramento de acesso à memória externa dos DSPs, em especial da família Blackfin, o sinal DATA é de 16 bits e os sinais, $\overline{\text{AMS}}[3:0]$, $\overline{\text{AWE}}$, $\overline{\text{ARE}}$, $\overline{\text{AOE}}$, $\overline{\text{ABE}}[1:0]$ são ativos quando o seu valor lógico é igual a zero.

Com o sinal $\overline{\text{AMS}}[x]$ é possível selecionar o banco de memória a ser utilizada na operação, esse sinal geralmente é definido dentro do registrador que configura o barramento de acesso à memória externa.

Os pinos de propósito geral de entrada ou saída GPIOs (*General Purpose Input/Output*), são sinais capazes de gerar um interrupção no processador digital de sinais, sendo desta forma um mecanismo propício para serem utilizados como sinal de controle entre o DSP e a FPGA.

O DSP ainda deve fornecer um sinal CLKOUT, que será o relógio de controle para escrever e ler na FPGA. Vale ressaltar que esse sinal caso seja constante pode também ser derivado para controlar outras operações intrínsecas ao *hardware* programável.

Ainda com referência à [20], a Figura 9, apresenta a relação dos principais sinais envolvidos de escrita seguida de uma leitura no mesmo endereçamento de memória da interface externa.

O procedimento de escrita assíncrona obedece ao seguinte ciclo:

1. No início da configuração inicial os sinais $\overline{\text{AMS}}[x]$ e $\overline{\text{ABE}}[1:0]$ assumem valores válidos;
2. Para habilitar a escrita, o sinal $\overline{\text{AWE}}$ assume o valor 0;
3. Antes de iniciar o período de *hold*, $\overline{\text{AWE}}$ se torna igual a 1;

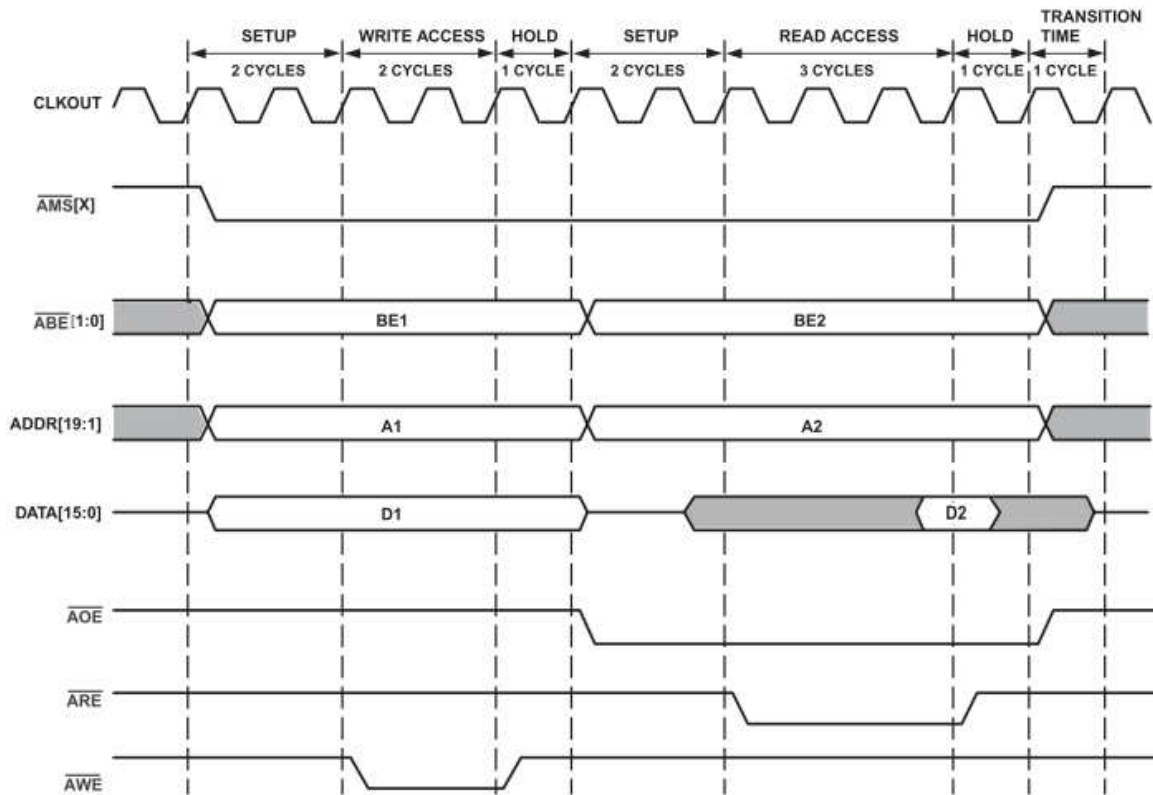


Figura 9 - Diagrama de tempo para escrita de dados na FPGA [20]

O ciclo do processo de leitura envolve outros sinais de controle, como segue:

1. No início do período de configuração, $\overline{\text{AMS}}[x]$, $\overline{\text{ABE}}[1:0]$ e $\overline{\text{AOE}}$ assumem valores válidos para o processo de leitura;
2. Para ativar a leitura o sinal $\overline{\text{ARE}}$ assume o valor lógico 0;
3. Durante o período de *hold* os dados recebidos são amostrados pela borda de subida do *clock*. O sinal $\overline{\text{ARE}}$ assumem o valor 1;
4. Caso não seja solicitado nenhuma outra leitura assíncrona no mesmo barramento de memória, no final do período de *hold*, $\overline{\text{AOE}}$ é igual a 1. E também, o sinal $\overline{\text{AMS}}[x]$ será 1 no ciclo seguinte, exceto se o mesmo banco de memória for utilizado nas operações seguintes;
5. Os sinais $\overline{\text{ABE}}[1:0]$ serão sempre 0 durante a leitura assíncrona dos 16 bits. No entanto, quando se trata de uma escrita assíncrona nos 16 bits mais significativos da memória,

$\overline{ABE}[0]$ assume o valor 1 e $\overline{ABE}[1]$ recebe 0. Entretanto, o oposto ocorre quando a escrita é realizada nos 16 bits menos significativos da memória, o sinal $\overline{ABE}[0]$ se torna igual a 0 e $\overline{ABE}[1]$ assume o valor lógico 1.

Devido esses sinais de controle serem controlados pelo AMC, surge uma camada de abstração em que o usuário não precisa necessariamente se preocupar com eles. Para ilustrar a facilidade de programação no DSP, a Figura 10 apresenta um pseudo código escrito em linguagem C, onde um banco de memória é definido e o dado é transmitido de forma totalmente transparente ao usuário sem que este necessite configurar algum periférico. Exceto os registros de controle de acesso à memória que devem ser configurados na inicialização do sistema e que não estão listados na ilustração.

```
#define MEM_BANK_0    0x20000000
#define    pTxDATA ((volatile unsigned short *) (MEM_BANK_0))

void SendData( short dado )
{
    *pTxDATA = dado;
    return;
}
```

Figura 10 - Pseudocódigo para envio dos dados para a FPGA

A partir da Figura 10, observa-se que basta definir um ponteiro para uma região de memória no caso ilustrado denominada MEM_BANK_0 que qualquer dado escrito neste o endereçamento será automaticamente enviado para a FPGA.

É importante destacar que além do AMC, o qual gerencia todo o acesso ao barramento de comunicação externa, muitos DSPs também possuem integrado ao chip um controlador de acesso direto à memória, também denominado de *DMA Controller*.

O controlador DMA é o responsável por toda a movimentação de dados na arquitetura, podendo movimentar dados tanto entre regiões de memória, como também, enviar e receber dados de diferentes interfaces sem a intervenção ou interrupção do processador.

4.2.1.2 Mecanismo para Evitar a Contenção de Barramento

Um fato interessante a ser observado é que mesmo utilizando controladores dedicados para acesso direto à memória é necessário atentar para um uso/acesso eficiente dos barramentos de comunicação, porque a utilização correta dos recursos é um fator crucial para o desenvolvimento de aplicações que demandam alto desempenho em sistemas embarcados [11].

Na arquitetura são utilizados diversos mecanismos de otimização para um uso eficiente dos recursos de comunicação, minimizando as ocorrências de contenções de barramento, as quais afetam diretamente o desempenho do sistema.

Para evitar a disputa de acesso ao mesmo endereço de memória é adotada a divisão de sub-bancos de memória, configurados no modo *double-buffer*, também conhecido como *buffer Ping-Pong*.

Com a utilização do *buffer Ping-Pong* é possível que um determinado processo W escreva em uma região de memória, enquanto um processo R realiza a leitura em outra região, anteriormente escrita por W. A Figura 11, ilustra a operação desse mecanismo.

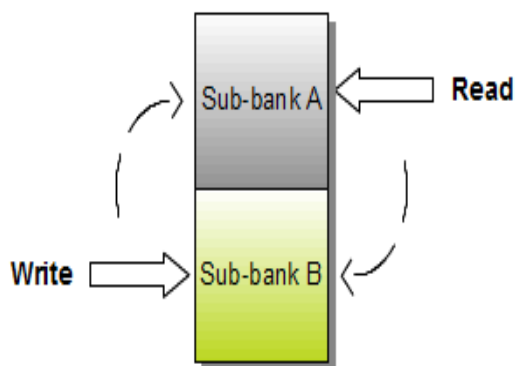


Figura 11 - *Double-buffer*

A divisão de bancos de memória tende a eliminar a concorrência de acesso, o que por sua vez, resulta na diminuição do *jitter* de acesso à memória, pois os processos de escrita e leitura não competem pelo mesmo endereçamento.

Entretanto, para se atingir o máximo desempenho em sistemas híbridos, também é de extrema importância o particionamento correto dos códigos. Assim, a Subseção 4.2.2 expõe algumas considerações sobre esse aspecto.

4.2.2 Particionamento de Códigos

O particionamento de códigos tem por objetivo descentralizar o processamento entre DSP e FPGA com a finalidade de elevar o desempenho computacional do sistema.

Um fator crítico nesse processo de divisão é determinar quais trechos do código deverão ser processados de forma paralela. Para tanto, algumas características devem ser observadas no processo, como, por exemplo, a distribuição espacial para uso eficiente dos barramentos e a complexidade computacional de determinada etapa do algoritmo [9 e 10].

A distribuição espacial dos dados é uma preocupação em particionar de maneira específica uma determinada tarefa. A descrição do acesso em um determinado sentido permite o uso eficiente do canal do barramento, evitando a contenção de dados e reduzindo conseqüentemente a latência do processo de comunicação entre as unidades [10].

Quanto à uma determinada complexidade computacional é possível detectar a partir do uso de ferramentas modernas de desenvolvimento, as estatísticas gerais de consumo de processamento por determinado trecho de código. Esse tipo de ferramenta permite identificar os gargalos do sistema, facilitando a tomada de decisão sobre quais trechos podem ser coprocessados [6 e 36]. Em geral, essa técnica produz um efeito mais eficiente do que a simples distribuição espacial dos dados.

Logo, para se obter um elevado desempenho em sistemas híbridos, a distribuição dos trechos de código, devem ocorrer de tal forma que o processador principal e tantos quantos outros módulos de coprocessamento existirem, possam operar de forma cooperativa,

descentralizada e de forma independente, objetivando explorar o ambiente de sinergia promovido pelos dispositivos DSP e FPGA.

A Figura 12, apresenta o modelo ideal de paralelismo, o qual garante que o processo principal que consome um tempo de processamento representado por $T1$, não precisa aguardar a conclusão das rotinas coprocessadas, que consomem um tempo $T2$, para dar seguimento ao fluxo natural do programa.

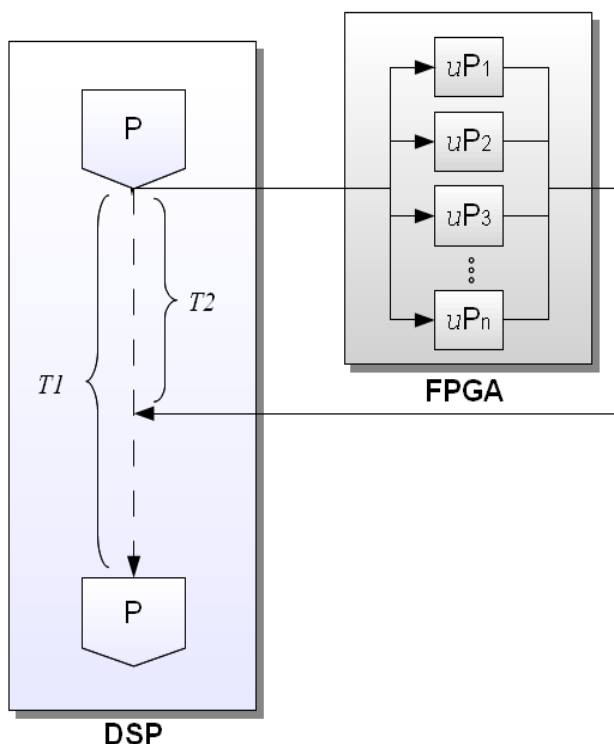


Figura 12 - Particionamento de código entre DSP e FPGA

Enquanto são coprocessadas algumas rotinas na FPGA, o DSP continua executando as seções independentes do código, acelerando a execução de regiões específicas de códigos e permitindo maior flexibilidade às rotinas computacionais, as quais podem operar de forma adaptativa e sensível ao contexto.

Entretanto, deve-se considerar o fato de que o processador principal, o qual opera em maior velocidade, pode entrar em estado de espera até que o coprocessador conclua a execução de um determinado trecho de código.

Assim, a determinação do tempo de *idle* do DSP é dada por 1.

$$T_{idle} = T_2 - T_1, \quad (4.1)$$

onde:

- T_{idle} , refere-se ao tempo total de ociosidade em que o processo de maior velocidade aguarda pelo retorno dos dados coprocessados. Se o resultado for negativo, isso significa que o processador não entrou em estado de espera, ao contrário, o valor negativo pode ser interpretado como o tempo que os dados estiveram disponíveis antes de serem solicitados;
- T_1 é o tempo considerado a partir do início do coprocessamento, no qual o processador opera sem a necessidade dos dados executados de forma descentralizada;
- T_2 , tempo gasto pelo coprocessador para executar as rotinas solicitadas pelo processo principal;

Com relação ao sincronismo dos processos em especial para o recebimento dos dados vindos da FPGA, considera-se no projeto o fato de que os DSPs modernos permitem que um determinado sinal externo gere uma requisição de interrupção. Por esse motivo, assim como representado na seção 4.3.1.1, pode-se fazer uso dessa característica para informar ao processador que os dados estão disponíveis para leitura.

A Figura 13 apresenta o processo de recepção dos dados escrito para o processador ADSP-BF533 da Analog Devices [20]. Na ilustração considera-se que um canal DMA está previamente configurado e que basta ativá-lo para que o controlador embutido no chip do DSP possa realizar a transferência dos dados sem a necessidade de intervenção direta do núcleo do processador.

```

EX_INTERRUPT_HANDLER(ISR_READ_DATA)
{
    //Acusa a geração da interrupção
    *pPORTGIO_CLEAR = PG12 ;

    //Habilita o canal DMA do destino dos dados
    *pMDMA_D0_CONFIG |= DMAEN;

    //Habilita o canal DMA de origem dos dados
    *pMDMA_S0_CONFIG |= DMAEN;

    //Sinaliza que o dado foi lido e destrava a aplicação,
    //caso ela esteja presa no laço da função principal do programa.
    ReadData = true;
}

int main()
{
    //Processamento independente do coprocessador
    :
    :

    //Verifica se o dado foi lido e se o canal DMA concluiu
    //a transferência, senão aguarda até que a rotina destrave o código.
    while(!ReadData || (*pMDMA_D0_IRQ_STATUS&DMA_RUN));

    :
    :

    //Prossegue no processamento utilizando os dados coprocessados
}

```

Figura 13 - Processo de leitura de dados via interrupção

Para a geração da requisição de interrupção do núcleo do DSP, são utilizados para os pinos de uso geral (GPIOs), previstos na interconexão lógica entre o DSP e a FPGA, conforme foi ilustrado na Figura 8.

A função `EX_INTERRUPT_HANDLER` descrita na Figura 13 é a responsável por processar a requisição de interrupção gerada pela FPGA, sendo que dentro dessa rotina são realizadas operação de baixo consumo de ciclo de *clock*, e isso pode ser verificado com o auxílio de ferramentas de depuração disponíveis no ambiente de programação dos próprios DSPs [14].

Primeiramente, ao ser gerada a ISR, o pino que PG15 o qual sinaliza a interrupção tem o seu sinal forçado para zero. Isso informa que a solicitação foi identificada e atendida. Após esse passo, os canais de transferência DMA são acionados para deslocarem os dados de uma determinada posição de memória para outra.

O segundo processo dentro da ISR é sinalizar a *flag* `ReadData` com o valor `TRUE` (verdadeiro) indicando que uma requisição de leitura foi processada e caso todos os dados já tenham sido escritos em memória, através do controlador DMA, o processador não permanecerá

preso na rotina de verificação localizada dentro da função principal do programa, mas caso isso ocorra, o tempo de *idle* passa a ser considerado até que o programa seja desbloqueado.

Para a arquitetura proposta neste estudo, dentro da FPGA coexistem micro blocos de processamento paralelo capazes de executar as operações sob controle do DSP. Esses micro blocos possuem um endereçamento único mapeado em regiões de memória compartilhada com o DSP.

Os registradores de controle e status do coprocessador armazenam diferentes informações sobre as etapas de processamento e de comunicação gerenciado pelo DSP. Para tanto, existe a necessidade de uma máquina de estado que visa garantir a correta funcionalidade do processo.

A etapa de operação para a configuração de cada micro bloco de processamento da FPGA, obedece às seguintes etapas, conforme ilustrado na Figura 14.

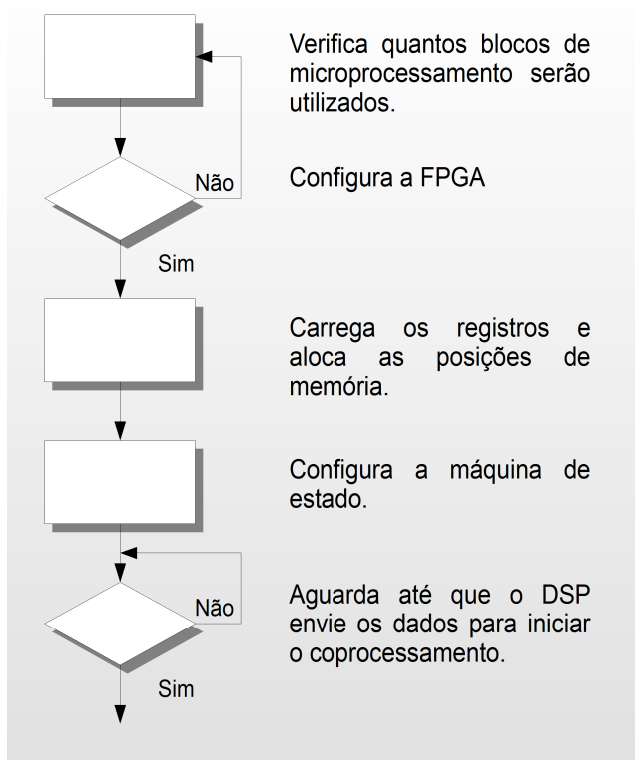


Figura 14 - Fluxo do processo de configuração da FPGA

Após o fluxo de configuração do sistema, a FPGA executa as seguintes etapas:

1. Inicia a execução do processo de forma assíncrona aos processos correntes na unidade de controle;
2. Sinaliza para a unidade de controle quais micro blocos estão envolvidos na operação e quais estão disponíveis para outras operações;
3. Após concluído o processamento, cada micro bloco retorna os valores para o DSP, utilizando os endereçamentos de memória compartilhada e sinaliza que está disponível para processar novas atividades.

4.2.3 Modelo Matemático Proposto

Para realizar um particionamento correto que conduza à resultados expressivos de otimização é necessário primeiramente identificar quais são os pontos críticos da aplicação e quais são os pontos que realmente irão fornecer um real aumento no desempenho computacional, quando forem processados de forma paralela.

Um recurso bastante útil para identificar as funções que ocupam maior parte do processamento são as ferramentas de estatísticas ou *statistical profilers*, as quais são disponibilizadas pelos fabricantes de aplicações de desenvolvimento de *software*.

Geralmente, todos os ambientes de desenvolvimento dos DSPs incluem esse recurso. Por exemplo, a Analog Devices fornece o VisualDSP++ como interface de programação para os seus processadores e a opção dessa API (*Application Programming Interface*) para a visualização do consumo de recursos de processamento é apresentada na Figura 15.

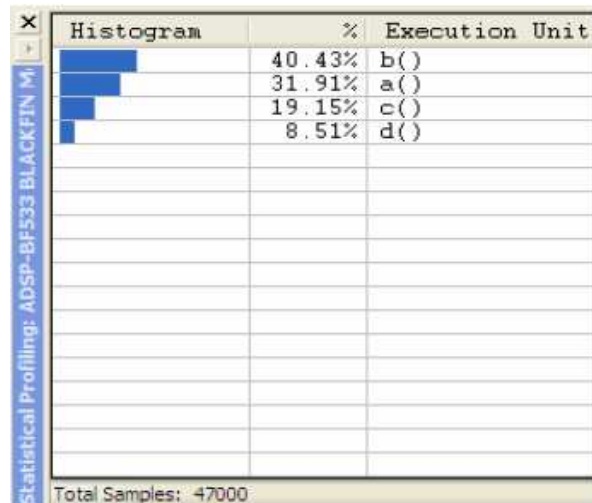


Figura 15 - *Statistical profiler* apresentando o percentual de execução de diversas funções [14]

Entretanto, ao contrário do que é citado em [36], onde os autores afirmam que para definir qual porção de código deve ser processada de forma descentralizada, deve-se observar apenas as ferramentas de estatísticas de processamento, considera-se neste estudo uma avaliação mais criteriosa e precisa.

Devido ao *statistical profiler* analisar de forma generalizada os processos que estão ocupando o processador, deve-se atentar para os tipos de rotinas que estão sendo executadas. Ou seja, como os DSP modernos possuem mecanismos de DMA e uma arquitetura otimizadas para operação com números fracionários ou flutuantes, não é viável enviar para a FPGA, uma rotina que esteja acessando memória, ou que esteja convertendo algum tipo de dado, como, por exemplo, a transformação de tipos numéricos de *int* para *float*.

Um outro ponto a ser considerado é o fato de que FPGAs, geralmente trabalham em uma frequência menor que os DSPs, por isso é extremamente importante ter alguma métrica capaz de identificar sob que determinadas condições realmente haverá uma redução no tempo de processamento final ao particionar um algoritmo e de quanto será essa melhoria.

4.2.3.1 Capacidade de Expansão

Antes de apresentar a proposta do modelo matemático para a arquitetura abordada neste estudo, julga-se necessário descrever a quantidade de Milhões de Instruções por Segundo, ou MIPS (*Million Instruction per Second*) que estarão disponíveis para o DSP, a partir do momento em que é utilizado o processamento paralelo.

A Equação 2, descrita em [24], permite calcular o desempenho de uma Unidade Central de Processamento, ou CPU (*Central Processing Unit*), expresso em MIPS.

$$MIPS = \frac{ClockRate}{CPI * 10^6}, \quad (4.2)$$

tal que:

- *ClockRate* é expresso em ciclos por segundo;
- *CPI* é a média de ciclos de *clock* por instrução.

Baseando-se na Equação 4.2, foi possível estender o mesmo conceito utilizado, aplicando-o à estrutura computacional que une DSP e FPGA.

Assim, a variável $MIPS_{AvailableOnTheDSP}$, corresponde à quantidade de MIPS disponíveis para o DSP e é representada pela Equação 4.3:

$$MIPS_{AvailableOnTheDSP} = MIPS - \frac{ClockRate - (DSP_{Cycles} - Hybrid_{Cycles})}{CPI * 10^6} \quad (4.3)$$

Logo, a versão simplificada de (4.3) pode ser escrita conforme a formulação (4.4), como segue:

$$MIPS_{AvailableOnTheDSP} = \frac{(DSP_{Cycles} - Hybrid_{Cycles})}{CPI * 10^6} \quad (4.4)$$

A Equação 4.4 considera a diferença do total de ciclos realizadas em um processamento centralizado subtraída pelo total de ciclos do sistema híbrido. Quanto maior for esse valor, maior será a capacidade de expansão em termos de capacidade de processamento do DSP.

4.2.3.2 Eficiência Global

Após realizar diversas pesquisas em acervos bibliográficos, como, IEEE *Xplore* e Portal ACM (*Association for Computing Machinery*), os quais reúnem as principais e mais recentes pesquisas na área de engenharia e computação, não foi encontrada nenhuma referência que faça menção a algum modelo matemático capaz de mensurar a viabilidade do particionamento de códigos em processamento descentralizado.

A avaliação da eficiência global que um determinado sistema descentralizado pode oferecer em relação à uma computação centralizada, ainda é considerado um problema em aberto. O que na verdade existem são alguns teoremas que definem o ganho máximo em termos de velocidade final ao utilizar vários processadores operando em paralelo. Para isso, as principais referências são as Leis de Amdahl [34] ou Gustafson [35] e elas não permitem retratar a viabilidade nem mesmo o ganho efetivo total obtido para a arquitetura proposta neste trabalho.

Assim, este trabalho propõe um modelo matemático capaz de mensurar a eficiência do particionamento de código com base em diversos fatores que influenciam diretamente no desempenho do sistema, como, por exemplo, o mecanismo de comunicação utilizado para conectar os dispositivos, a capacidade de transmissão e recepção dos dados, tempo de ociosidade ou *idle*, dentre outros.

Portanto, a partir de então, será deduzida a formulação proposta capaz de mensurar a eficiência do sistema proposto.

Primeiramente, julga-se necessário estabelecer qual será o valor de referência que deverá ser utilizado para avaliar quão eficiente é a proposta de estrutura computacional apresentada neste estudo. Para tanto, será determinado o custo computacional para o DSP processar um determinado código e uma vez conhecido esse tempo de processamento, pode-se utilizar esse valor para as futuras comparações.

Assim, a variável $T_{Centralized}$, representará o tempo consumido pelo DSP para o processar de forma isolada a rotina computacional a ser particionada na arquitetura proposta. Logo, $T_{Centralized}$ é dado pela Equação 4.5, como segue:

$$T_{Centralized} = \frac{DSP_{Cycles}}{DSP_{Freq.}} [s], \quad (4.5)$$

onde, DSP_{Cycles} refere-se à quantidade de ciclos gastos para o DSP executar o algoritmo a ser processado na FPGA e $DSP_{Freq.}$ representa a frequência do processador.

Após ter sido definido a referência $T_{Centralized}$, representa-se a variável $T_{Cooperative}$ como sendo o tempo do processamento distribuído entre DSP e FPGA, como segue:

$$T_{Cooperative} = \frac{Hybrid_{Cycles}}{DSP_{Freq.}} [s] \quad (4.6)$$

Sendo que, $Hybrid_{Cycles}$ é o total de ciclos de *clock* consumidos pelo sistema híbrido, composto por DSP e FPGA e que será utilizado como métrica comparativo para poder avaliar a eficiência atingida pelo particionamento.

Entretanto, a Equação 4.6 ainda não resultará em um valor muito exato, porque não está considerando o tempo de ociosidade do processador DSP. E esse fator é muito importante a ser considerado, porque mesmo que haja um paralelismo ideal do código, dependendo da taxa de operação entre os processos, pode ocorrer o fato em que a rotina executada em maior velocidade tenha que esperar a conclusão da outra. E quando isso ocorre a eficiência global do sistema deve decrescer.

Dessa forma, propõe-se que uma variável capaz de mensurar o tempo de *idle* do DSP seja considerada em (4.5). Logo, com base na Equação 4.1 apresentada na seção 4.3.2, a determinação de tempo de ociosidade do DSP, representada por DSP_{Idle} será obtida a partir das seguintes variáveis:

- $DSP_{NumOfWords}$ - Quantidade de palavras enviadas para a FPGA;
- $DSP_{WordSize}$ - Tamanho da palavra enviada para a FPGA;
- $DSP_{CyclesToGetAnswer}$ - Quantidade de ciclos que o DSP executa antes de necessitar dos dados da FPGA;
- $DSP_{Freq.}$ - Frequência de operação do DSP;
- $Bus_{WordSize}$ - Capacidade do barramento de comunicação (quantidade de bits transmitidos em paralelo);
- $Bus_{Freq.}$ - Frequência de operação da interface de comunicação entre os dispositivos;
- $FPGA_{Cycles}$ - Refere-se ao total de ciclos necessários para a FPGA coprocessar os dados solicitados pelo DSP.
- $FPGA_{Freq.}$ - Frequência de operação da FPGA (coprocessador);
- $FPGA_{NumOfWords}$ - Quantidade de palavras retornadas para o DSP;
- $FPGA_{WordSize}$ - Tamanho da palavra retornada enviada para o DSP.

Assim, DSP_{Idle} é dado por:

$$\begin{aligned}
 DSP_{Idle} = & \frac{DSP_{NumOfWords} * DSP_{WordSize}}{Bus_{freq.} * Bus_{WordSize}} + \frac{FPGA_{Cycles}}{FPGA_{Freq.}} + \\
 & \frac{FPGA_{NumOfWords} * FPGA_{Word.Size}}{Bus_{freq.} * Bus_{WordSize}} - \frac{DSP_{CyclesToGetAnswer}}{DSP_{Freq.}}
 \end{aligned}
 \tag{4.7a}$$

E, após definido DSP_{Idle} a Equação 4.6 é redefinida conforme (4.7b):

$$T_{Cooperative} = \frac{Hybrid_{Cycles}}{DSP_{Freq.}} + DSP_{Idle}[s],
 \tag{4.7b}$$

Portanto, uma vez definidos os tempos de referência, $T_{Centralized}$ conforme (4.7b), e o tempo de processamento distribuído $T_{Cooperative}$, de acordo com a formulação (4.5), pode-se mensurar o percentual de eficiência obtido. Ou seja, para representar o quão veloz foi o sistema híbrido em relação ao modelo de processamento centralizado, considera-se a relação apresentada em 6.

$$Efficiency = \left(1 - \left(\frac{T_{Cooperative}}{T_{Centralized}} \right) \right) * 100 \text{ [%]}, \quad (4.8)$$

O modelo analítico proposto tem o objetivo de mensurar o ganho efetivo total que será fornecido ao sistema, caso esse seja distribuído com a utilização da arquitetura proposta.

5. Simulações e Resultados

Este Capítulo descreve as simulações e resultados obtidos com a utilização da arquitetura proposta acompanhada das análises feitas com base no modelo analítico apresentado neste trabalho.

Primeiramente, se faz necessário descrever quais foram os recursos utilizados no desenvolvimento do experimento, tais como, a ferramenta de prototipagem, o simulador de comportamento de *hardware*, as características do processador digital de sinais e o seu respectivo ambiente de desenvolvimento.

É importante destacar que os resultados alcançados foram obtidos por meio de simulações e as execuções de trechos de códigos foram processadas e computadas isoladamente.

5.1 Recursos Utilizados

Para validação da proposta pode-se enumerar os recursos utilizados nos seguintes:

- SmartDesign, uma ferramenta de prototipação rápida [19];
- ModelSim, utilizada na simulação de *hardware* [18 e 23].
- Processador digital de sinais ADSP-BF533 [20 e 22];
- Ambiente de desenvolvimento VisualDSP5.0++ [21];

5.1.1 SmartDesign

A ferramenta SmartDesign da Actel, possibilita a descrição de núcleos de processamento e de circuitos, simplificando a construção de projetos em FPGA. A Figura 16 apresenta a aba Canvas, com a qual é possível interconectar diferentes blocos de processamento sem o uso explícito da linguagem descritiva de *hardware*, ou HDL (*Hardware Description Language*).

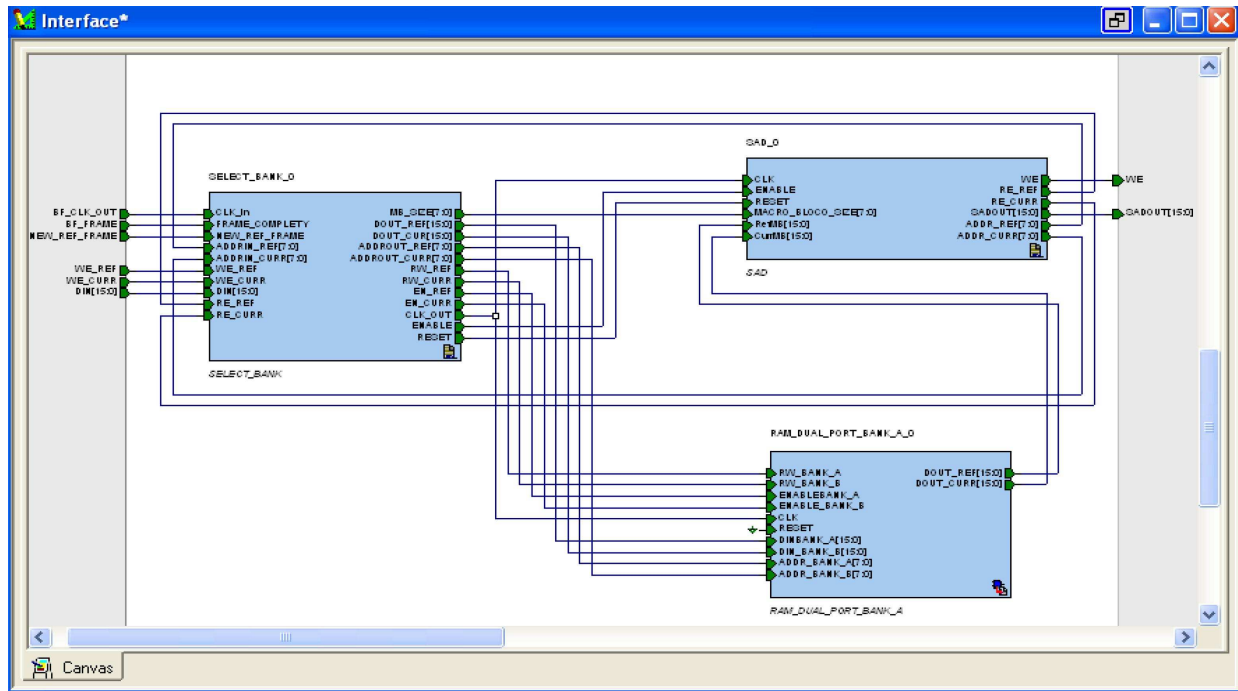


Figura 16 – Visualização da aba Canvas do SmartDesign

O SmartDesign não substitui totalmente a HDL. Entretanto, além de possuir seus próprios *cores* e *Basic Blocks*, como: Acumuladores, Multiplexadores, *Dual Port* RAM, dentre outros, ele possibilita que um arquivo escrito em linguagem descritiva de *hardware* também se transforme em um dispositivo de circuito, o qual poderá ser inserido ao escopo do projeto. Isso permite que durante a implementação dos circuitos haja um nível de abstração maior, permitindo a visualização modular dos diferentes estágios de desenvolvimento e facilitando a análise sistêmica do projeto.

5.1.2 ModelSim

ModelSim é um aplicativo desenvolvido pela MentorGraphics, capaz de verificar e simular circuitos elétricos escritos em diferentes linguagens descritivas de *hardware*, dentre as quais, as de maior destaque são: VHDL, Verilog e System Verilog [23].

Devido ser uma das ferramentas mais utilizadas pelos desenvolvedores, os fabricantes de FPGA, geralmente integram o ModelSim ao seu ambiente particular de desenvolvimento,

possibilitando geração de diversos tipos de simulações, facilitando a simulação de todo o comportamento de um sistema sem a necessidade sintetizar e descarregar o código binário para o dispositivo FPGA.

Os resultados das simulações podem ser visualizados por meio de recursos próprios da ferramenta, onde é possível analisar os atrasos e demais comportamentos dos sinais através de formatos de onda que podem ser gerados, manipulados, ou ainda, serem resultantes de circuitos secundários [18].

5.1.3 Processador Digital de Sinais ADSP-BF533

Devido a abordagem deste estudo consistir na concepção de uma arquitetura computacional de baixo custo e de alto desempenho computacional, optou-se pela utilização do ADSP-BF533, um processador que custa em torno de \$30 (trinta dólares americanos), um valor muito abaixo em comparação a outros DSPs concebidos para finalidades específicas. Por exemplo, os DSPs da família DaVinci, os quais são otimizados para o processamento de imagens, podem representar até 2,5 vezes o valor de um processador de dois núcleos, como revela a Tabela IV.

Tabela IV – Comparação de custo entre diferentes DSPs de ponto fixo

DSP	Frequência Máxima (MHz)	On-Chip RAM (kB)	Valor (\$)
ADSP-BF527	600	132	25,49
ADSP-BF533	600	148	30,29
ADSP-BF561(Dual-Core)	600	328	48,75
ADSP-BF547	600	260	30,17
TMS320DM355ZCEA21 - DaVinci	216	56	30,6
TMS320DM368ZCED - DaVinci	432	56	49,59
TMS320DM6467ZUTA - DaVinci	594	248	119,42

O ADSP-BF533 é um processador digital de sinais da família Blackfin, incorporado conjuntamente entre a Analog Devices e a Intel *Micro Signal Architecture* (MSA). Os processadores Blackfin reúnem o que é considerado o estado da arte em processamento de sinais, como, *Dual-MAC*, arquitetura RISC e um conjunto de operações contendo várias unidades operacionais de computação paralela, também conhecidos por *single-instruction, multiple-data* (SIMD) [13 e 20].

O núcleo dos processadores da família Blackfin, vide Figura 17, possuem dois multiplicadores de 16-bit, dois acumuladores de 40-bit, duas ULAs de 40-bit, e um deslocador de 40-bit. Sendo que os dados a podem ser processados em 8, 16 ou 32 bits.

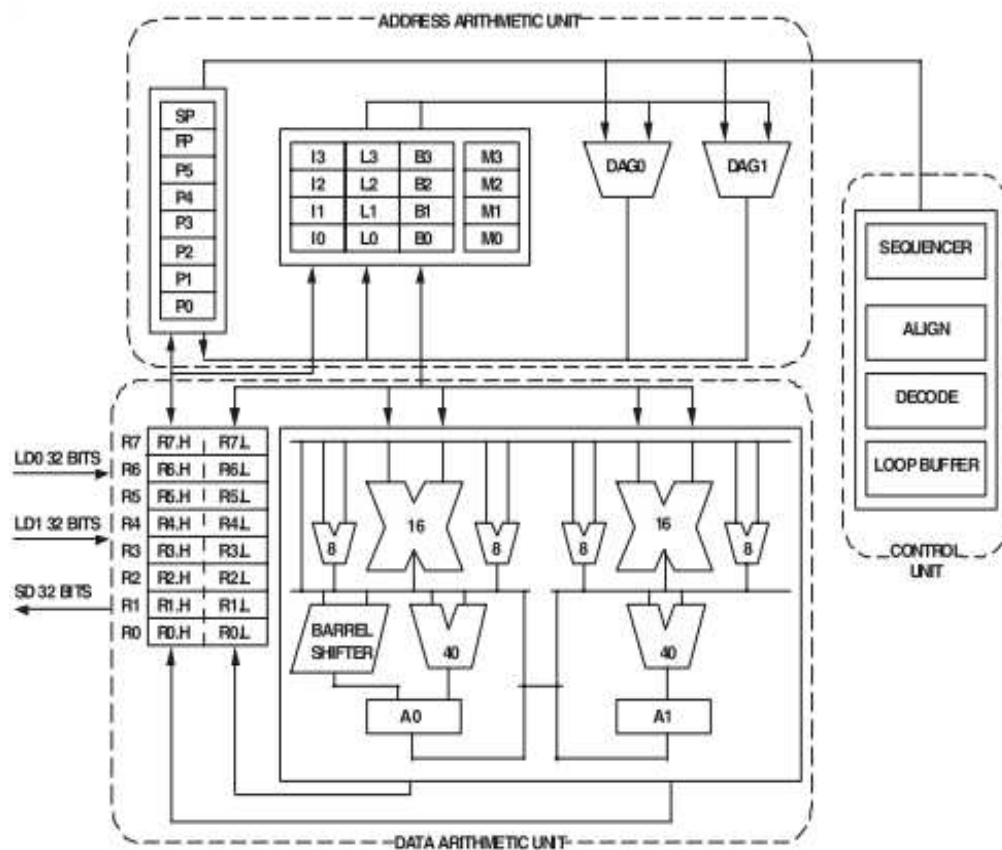


Figura 17 – Núcleo dos processadores da família Blackfin [20]

Cada MAC é capaz de realizar uma multiplicação de 16bits x 16 bits em um único ciclo de *clock*, acumulando o resultado em 40 bits. As operações envolvidas podem ser realizadas utilizando diferentes tipos numéricos, como, *signed* ou *unsigned*.

Além de um sofisticado núcleo para operações aritméticas, um dos fatos que também justificam a escolha do ADSP-BF533, é a flexibilidade encontrada nessa arquitetura, na qual existem diversos módulos de conexão para periféricos, dentre eles, pode-se citar o controlador de acesso direto à memória interna e/ou externa.

O ADSP-BF533 possui um modo particular de controle dos barramentos. Ele mantém uma relação entre a frequência de operação do processador e a frequência de controle dos periféricos de transmissão e aquisição de dados. Dado esse fato e, tendo em vista que a proposta de arquitetura descrita neste estudo utiliza o processamento descentralizado, se torna necessário enfatizar o tipo de gerenciamento utilizado por esse DSP.

A multiplicação do *clock* de entrada do processador, denominado CLKIN, fornece a frequência do *clock* do processador e o *duty cycle*, permitindo a geração do PLL (*Phase Locked Loop*). O usuário pode programar o PLL com um fator de multiplicação para CLKIN, resultando no VCO (*Voltage Controlled Oscillator*), que por sua vez é dividido e resulta na frequência de operação do processador, denominada CLK.

O usuário também pode dividir o valor do VCO para gerar o *clock* do sistema (SCLK). O SCLK, por sua vez, determina a frequência de operação dos barramentos *Peripheral Access Bus* (PAB), *DMA Access Bus* (DAB), *External Access Bus* (EAB) e o *External Bus Interface Unit* (EBIU).

O PAB controla a frequência de operação dos periféricos da arquitetura. O DAB, refere-se a um barramento exclusivo para uso do controlador DMA, o EAB é utilizado para gerenciar as requisições de acesso à memória solicitadas pelo núcleo e o EBIU é o canal que permite o acesso à regiões de memória externa ao processador.

É importante destacar que a programação dos sinais de CLK e SCLK, devem obedecer à uma relação específica determinada para a família Blackfin. Essa relação é definida pelo bit SSEL (*System Select*). A Tabela V, apresenta a correspondência entre esses sinais de controle.

Tabela V – Relação entre CLK e SCLK[20]

SSEL[3:0]	Relação CLK/SCLK	CLK (MHz)	SCLK (MHz)
1	1:1	100	100
10	2:1	200	100
11	3:1	300	133
100	4:1	400	125
101	5:1	500	120
110	6:1	600	100
N = 7 a 15	N:1	600	600/N

Para se obter um melhor desempenho, o Blackfin admite o ajuste em tempo de execução da frequência dos relógios, possibilitando um gerenciamento eficiente da dissipação de potência e consumo de energia.

5.1.4 VisualDSP++

O VisualDSP++ é o ambiente específico para a programação de sistemas embarcados para as diferentes famílias de processadores da Analog Devices. A versão utilizada nos testes é a 5.0, atualização 8 [21].

Esse ambiente reúne um conjunto de ferramentas necessárias para o desenvolvimento e gerenciamento de aplicações e projetos. Dentre as principais, pode-se destacar o ambiente gráfico de depuração e desenvolvimento integrado, também chamado de IDDE (*Integrated Development and Debugging Environment*) e um conjunto de bibliotecas otimizadas escritas em linguagem C ou Assembly para o processamento digital de sinais em tempo real. A Figura 18, apresenta o ambiente de desenvolvimento do VisualDSP++ 5.0.

Nesse ambiente de desenvolvimento é possível desenvolver aplicações em linguagem Assembly, C e C++. Sendo que ao escrever uma rotina, ao compilar o código será gerado um

arquivo correspondente em Assembly, caso não tenha sido utilizada essa linguagem, e após isso dá-se prosseguimento para a geração do arquivo binário que será descarregado no dispositivo.

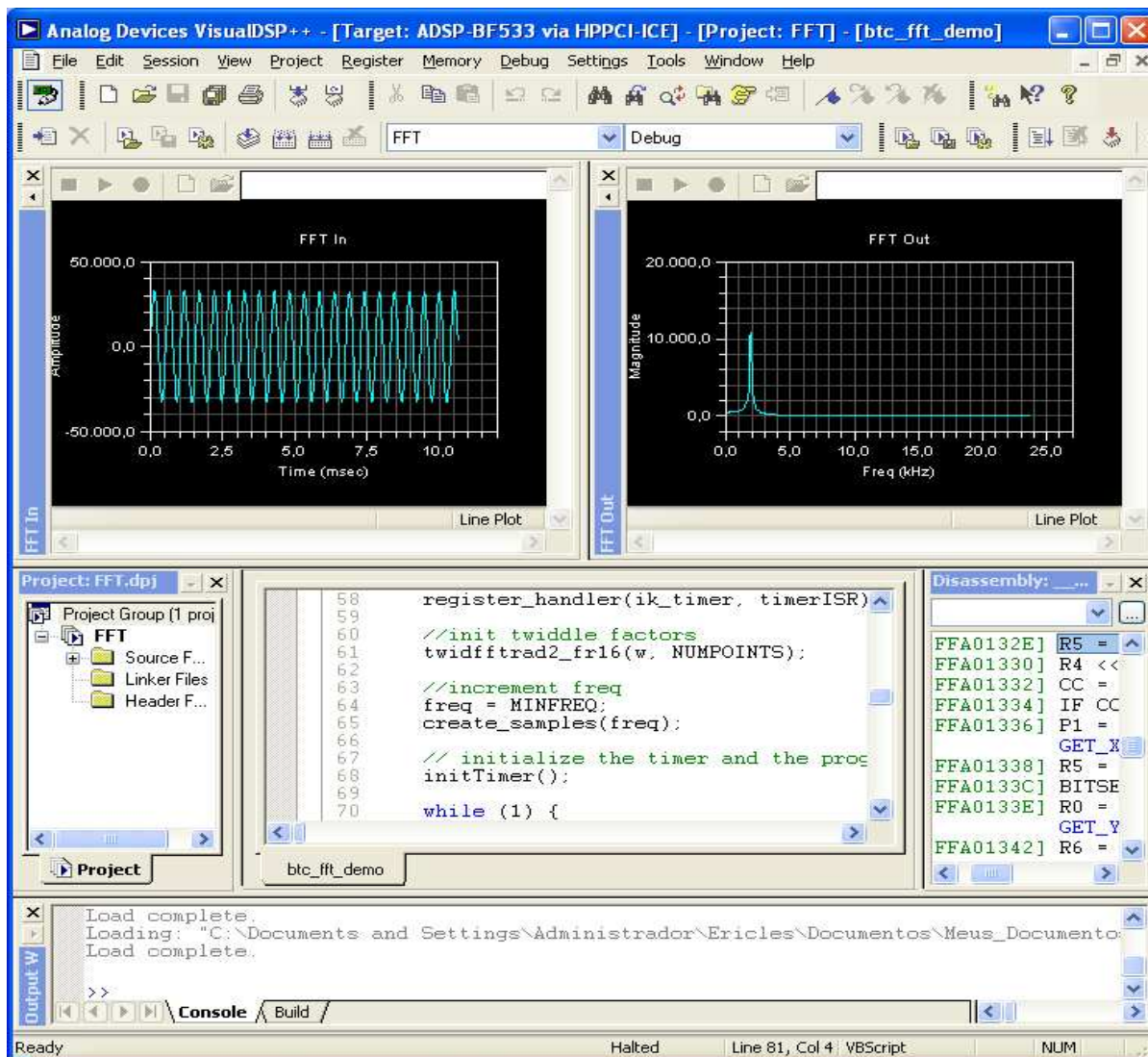


Figura 18 – Ambiente de desenvolvimento do VisualDSP++5.0

A partir da Figura 18 é possível visualizar as diferentes ferramentas disponibilizadas pelo VisualDSP++ 5.0. No exemplo ilustrado, o programa está executando uma FFT e as janelas superiores, apresentam o sinal de entrada e a sua resultante após o processamento.

Ainda é importante destacar, que mesmo em ambientes de desenvolvimento sofisticados como o VisualDSP, tendo em vista as limitações físicas dos sistemas embarcados é necessário ter

o conhecimento pleno da arquitetura utilizada para que se possa escrever um código capaz de explorar os recursos disponíveis de forma otimizada, permitindo o aumento do desempenho do sistema e evitando a inserção de redundâncias por parte do compilador.

5.2 Cenário de Testes

Para constatar a viabilidade da arquitetura computacional elaborada, foi realizado um cenário de testes capaz de ilustrar o ganho em desempenho da proposta em comparação aos modelos de processamento centralizado. As métricas apresentadas foram obtidas a partir da aplicação do modelo matemático analítico descrito no Capítulo 4.

O cenário de teste, ilustra um ambiente aplicado ao processo de codificação de vídeo, também utilizado pelo padrão H.264/AVC (*Advanced Video Coding*), o qual normatiza o método de compressão de vídeo digital em alta definição utilizado pelos principais sistemas de televisão digital do mundo, dentre eles, o SBTVD (Sistema Brasileiro de Televisão Digital).

5.2.1 Estimação de Movimento

Este cenário de testes, utiliza uma rotina computacional amplamente utilizada no processo de codificação de imagens digitais e que apesar de ser um algoritmo relativamente simples, demanda elevada carga de processamento devido ser aplicado constantemente à todos os quadros de imagens das cenas e nos seus respectivos pixels.

Em aplicações de processamento de vídeo digital, as imagens subsequentes a um determinado quadro de referência, onde ambos compõem uma mesma cena, são muito semelhantes, esse fato resulta em uma elevada redundância temporal entre eles [15].

A diferença entre essas imagens correspondem ao movimento dos elementos da cena com relação ao quadro anterior, ou ainda à movimentação da câmera. Para determinar a diferença entre as imagens e eliminar a redundância entre os quadros, durante o processo de codificação são utilizados algoritmos capazes de realizar a estimação de movimento, ou ME (*Motion Estimation*) da cena.

Um conjunto de sequências finitas de instruções são capazes de identificar a relação entre os quadros vizinhos e compor um mapa contendo a relação entre eles utilizando os vetores de estimação de movimento [15 e 16].

Os vetores de movimento resultantes do cálculo da ME são codificados no *bitstream* que representa o vídeo compactado. No processo de decodificação do vídeo é realizado um processo conhecido como compensação de movimento, ou MC (*Motion Compensation*).

O MC consiste na reconstrução do vídeo originalmente codificado utilizando a informação contida nos vetores de movimento. Dessa forma, devido o processo de decodificação já conhecer a possível posição dos elementos da imagem a ser descompactada, ele necessita de um esforço computacional muito menor ao se comparar com o processo de compressão [15].

Para determinação da estimação de movimento é necessário a definição de uma região de referência, uma região de procura e um esquema capaz de detectar o vetor que melhor representa a relação entre elas. Para tanto, três critérios são utilizados para calcular a distorção entre as regiões dos quadros da imagem: MAE (*Mean Absolute Error*), MSE (*Mean Square Error*) e SAD (*Sum of Absolute Difference*) [15 e 16].

As regiões analisadas pelo algoritmo de estimação de movimento, são divididas em macro blocos. Quando os macro blocos contêm múltiplos objetos que se movem em diferentes direções, isso faz com que a escolha de apenas um vetor de movimento não seja suficiente para representar de forma satisfatória o verdadeiro movimento da cena. Por isso, para elevar desempenho da predição, o padrão H.264/AVC utiliza diferentes dimensões de macro blocos para analisar os quadros [28 e 31].

Embora, sejam geralmente citados três critérios para estimação de movimento, a SAD, também conhecida por SAE (*Sum of Absolute Errors*) é método mais utilizado, porque implica em um menor custo de processamento e fornece um resultado satisfatório, equivalente ao método MAE, o qual possui maior complexidade computacional [16]. A SAD é o método utilizado em um dos principais códigos de referência para o H.264/AVC [17].

A Equação 5.9, apresenta a soma das diferenças absolutas entre os pontos correspondentes de uma região de referência R e os pontos correspondentes da região de

procura do quadro atual, definido por C . A região de procura atende a um fator $M \times N$, que corresponde ao tamanho dos macro blocos a serem analisados para cada uma das diferentes posições dos quadros de imagens analisados.

$$SAD = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |C_{i,j} - R_{i,j}| \quad (5.9)$$

Os algoritmos utilizados para determinar o vetor de movimento influenciam diretamente na qualidade de imagem reproduzida e conseqüentemente no desempenho do sistema. A característica que mais influencia no desempenho do algoritmo de ME é a quantidade de interações de M e N .

Em [16] os autores relatam a existência de diversos tipos de busca a ser realizado dentro do quadro de imagem, como: Busca Hierárquica, Casamento de Borda, Busca Logarítmica, Busca Circular, Busca Ordenada, dentre outros.

Ainda com relação à [16] é possível constatar que todos os métodos de busca citados, possuem como principal objetivo substituir a utilização do algoritmo de Busca Completa, porque ele exige grande capacidade computacional, devido percorrer todos os macro blocos das imagens.

Com a finalidade de comparar o custo computacional da metodologia de Busca Completa, considera-se em [16] que para estimar o movimento de um vídeo com resolução 640x480 à uma taxa de 30fps são necessárias aproximadamente 27 vezes mais MOPS (milhões de operações por segundo) em relação às metodologias de menor complexidade.

Mas, como o algoritmo de Busca Completa é o único a atingir o valor ótimo, o presente cenário de testes utiliza essa metodologia para demonstrar o potencial da arquitetura híbrida proposta submetida à uma lógica exaustiva em termos computacionais.

5.2.2 Concepção do Código de Referência

Para visualizar o tempo necessário para computar a estimação de movimento em um processo de codificação de vídeo, em [29] os autores apresentam um gráfico, conforme ilustrado na Figura 19, onde é possível constatar que o módulo de estimação de ME consome cerca de 35% do tempo de processamento do codificador *x264*, um dos *softwares* de código aberto mais eficientes da atualidade, capaz processar cerca de 45 vezes mais rápido a codificação do padrão H.264/AVC, em comparação ao código de referência citado em [17] [29].

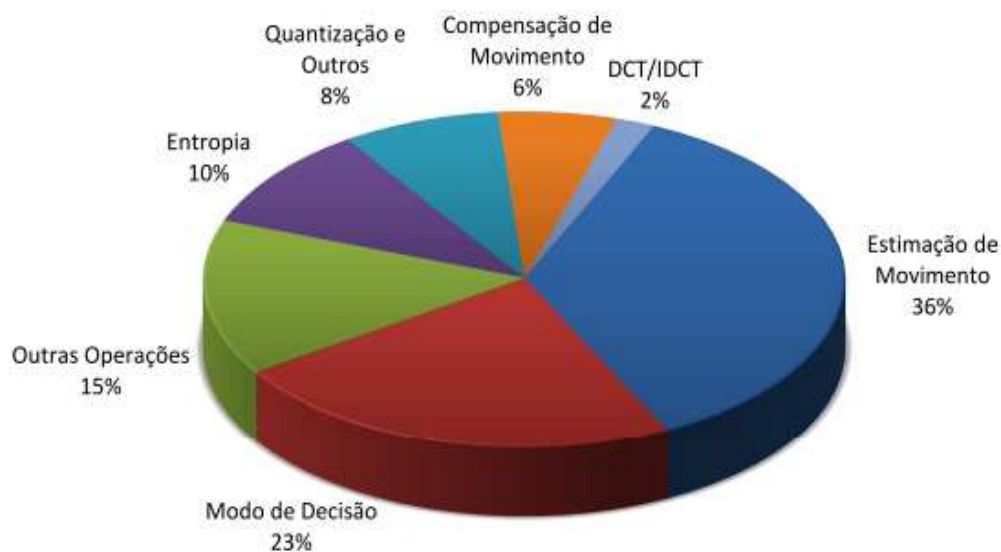


Figura 19 – Análise do tempo de execução do codificador *x264* [29]

Para se ter uma referência em que se possa comparar os resultados obtidos com a arquitetura proposta neste estudo, foi escrito um código em linguagem C, estruturado de tal forma a obter elevado desempenho do processador Blackfin ADSP-BF533. O algoritmo escolhido para calcular a estimação de movimento é a Soma das Diferenças Absolutas, o mesmo utilizado por [17] e [29].

O código desenvolvido foi baseado no algoritmo da Intel [30], que calcula todo o bloco de estimação de movimento para o padrão H.264/AVC. Com base nessa referência, portou-se o mesmo código para o Blackfin utilizando a mesma linguagem de programação, mas,

reescrevendo alguns trechos do código com a finalidade de otimizar o desempenho do processador digital de sinais.

As técnicas de otimização utilizadas para conceber o código de referência, consistem basicamente na substituição de ponteiros de memória por vetores de armazenamento de dados em regiões de memória interna ao chip, as quais possuem maior velocidade de acesso. Também se fez o uso de bibliotecas matemáticas fornecidas pelo fabricante do DSP. Essas alterações possibilitaram um ganho em desempenho de cerca de 46%, para o cálculo de uma SAD 16x16.

A Figura 17, mostra o fluxograma do processo de computação da SAD para o código escrito no Blackfin e baseado em [17 e 30].

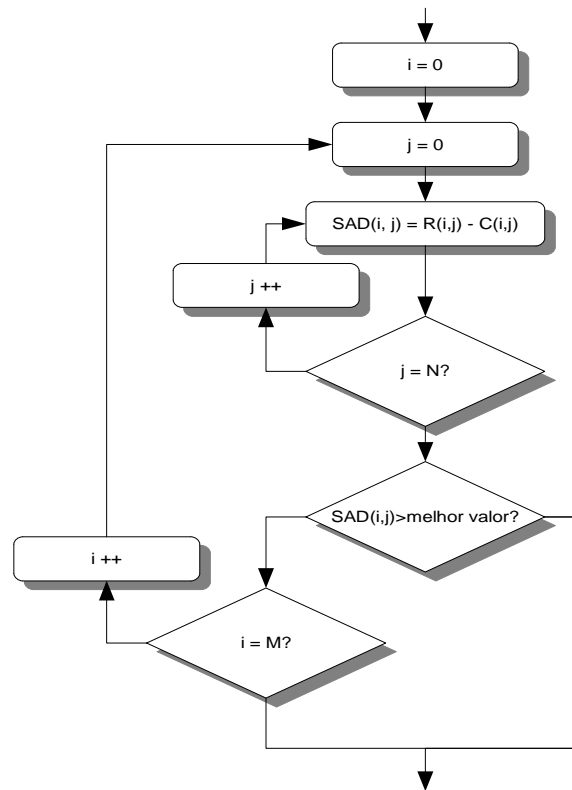


Figura 20 – Modelo tradicional de computação da SAD

A Figura 20, permite observar a existência de um laço de repetição encadeado. Necessário, porque as interações compartilham recursos ou informações, isso gera uma redução do desempenho do uso eficiente do pipeline do Blackfin.

O mais agravante é que devido às implementações práticas do padrão H.264/AVC exigirem um número de interações variáveis, conforme a característica do vídeo a ser codificado, isso dificulta a implementação de técnicas de otimização para o DSP, sem que isso não exija dele um grande consumo de área de memória, o que é consequência da aplicação de uma das mais utilizadas técnicas de otimização de laços de repetição, como, o *Loop Unrolling* [32].

5.2.2.1 Particionamento do Código

Considerando o código de referência [30], foi realizado um levantamento para identificar quais rotinas computacionais concentram maior esforço por parte do Blackfin. A Figura 21 apresenta a distribuição das cargas de processamento.

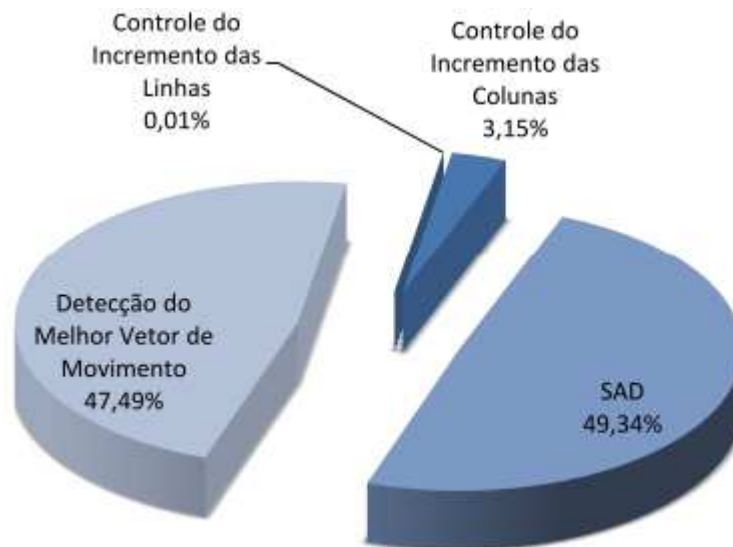


Figura 21 – *Statistical profiling* do código de referência que calcula a SAD

O *statistical profiling* do código de referência revela que o cálculo da SAD corresponde a maior parte do processamento, cerca de 49,34%. Isso já era um fato esperado, apesar de que a função utiliza apenas executa a soma e subtração em valores absolutos.

Entretanto, a rotina de detecção do vetor de movimento também gera um custo computacional muito grande, representando 47,49% da carga presente no processador. Essa rotina, basicamente compara os vetores de movimento calculados pela SAD com a melhor hipótese considerada pelo processo de codificação. Uma vez detectado que soma das diferenças absolutas de uma determinada coordenada dos quadros corresponde à melhor estimativa de movimento, previamente estabelecida pelas demais rotinas do processo de codificação, a continuação do cálculo da SAD se torna desnecessária e a continuação dos cálculos deve ser interrompida.

Também é importante destacar que o percentual representado pelo controle do incremento das colunas (3,15%), aparentemente não representa um valor expressivo. Mas, devido a simplicidade da rotina, a qual realiza apenas um laço para incremento das posições das colunas, o percentual representado por ela passa a ser um valor de grande expressão. Isso se deve ao fato de que o laço de repetição está encadeado dentro de outro, o que impede o DSP de utilizar de forma eficiente o pipeline de sua própria arquitetura.

Logo, de posse das características de processamento do algoritmo da SAD no Blackfin, foi tomada a decisão de particionamento do código visando proporcionar o ambiente de total sinergia. Ou seja, aquilo que é crítico para o DSP passa a ser processado pela FPGA e vice-versa.

Portanto, foram distribuídas as cargas do processamento em dois caminhos, os quais são denominados: Módulo DSP, encarregado de realizar as rotinas condicionais e de comparação e o Módulo FPGA, responsável pela execução da SAD.

5.2.2.1.1 Módulo DSP

O Módulo DSP é o responsável por determinar se a melhor estimativa para o vetor de movimento já foi encontrada. Ele controla a execução de todo o algoritmo da SAD, ou seja, é ele quem determina o momento de início e término do processo.

Inicialmente, o Módulo DSP realiza a configuração do sistema, determinando a forma de acesso aos bancos de memória assíncrona, no qual está conectada a FPGA. Após isso, são configurados os canais de transferência e recepção de dados utilizando mecanismos de acesso

direto à memória (DMA). Por último, o DSP envia os dados referentes aos quadros de imagens e os parâmetros de configuração para a FPGA.

Após realizada a etapa de inicialização, descrita no parágrafo superior, o DSP detecta o melhor vetor de estimação de movimento com base nos cálculos retornados pela FPGA, uma vez que o processador detectou o vetor que representa a melhor hipótese de ME, ele envia um comando para a FPGA suspender o processo dos demais pixels do quadro.

Todos os dados retornados pela FPGA geram uma interrupção no Blackfin, possibilitando que os processos se mantenham totalmente desacoplados, sendo que a entrada de um novo valor, correspondente a um vetor de movimento, ocorra baseado em um processo de interrupção gerada por *software*, isso assegura que todos os dados serão detectados pelo processador, que por sua vez não precisa ficar preso à uma rotina de supervisão ou monitoramento de algum sinal entrante.

5.2.2.1.2 Módulo FPGA

O Módulo FPGA executa o cálculo da soma das diferenças absolutas, operando sob controle do Módulo DSP. Semelhante à [16], o código escrito na FPGA tem a capacidade de processar e enviar para o DSP uma hipótese do vetor de movimento a cada ciclo de *clock*.

Existe ainda a preocupação com a ocorrência de contenção de barramento e o sincronismos dos processos entre DSP e FPGA. Para tanto, como medida preventiva, enquanto o processo de DMA do Blackfin está alimentando a memória RAM da FPGA com os dados referentes ao quadro atual e ao quadro de referência, a FPGA processa a SAD para os quadros já armazenados em uma *Dual Port* RAM, acessando-a em modo *double-buffer*, conforme descrito no Capítulo 4.

Os valores calculados na FPGA são enviados para o Blackfin, responsável em comparar se o valor atual retornado pela SAD corresponde à melhor hipótese de movimento, caso isso seja verdadeiro, o DSP desabilita a continuidade do processo executado pela FPGA.

Para solucionar o problema de pipeline detectado no código de referência, foi desenvolvida uma técnica, a qual permite extinguir totalmente os laços de repetição. Tendo em

vista que os laços compartilhavam recursos durante as suas interações, pôde-se plicar o conceito de pipeline, o qual consiste no processamento de múltiplos segmentos de código concorrentemente sem a duplicação de recursos.

O circuito implementado na FPGA é capaz de processar em uma borda de descida do *clock* o incrementado dos índices dos macro blocos e a leitura dos quadros C e R. E na borda de subida calcula a SAD, enviado o valor para o DSP. A Figura 22, utiliza a tela do ModelSim para apresentar o ciclo de processamento realizado pela FPGA.

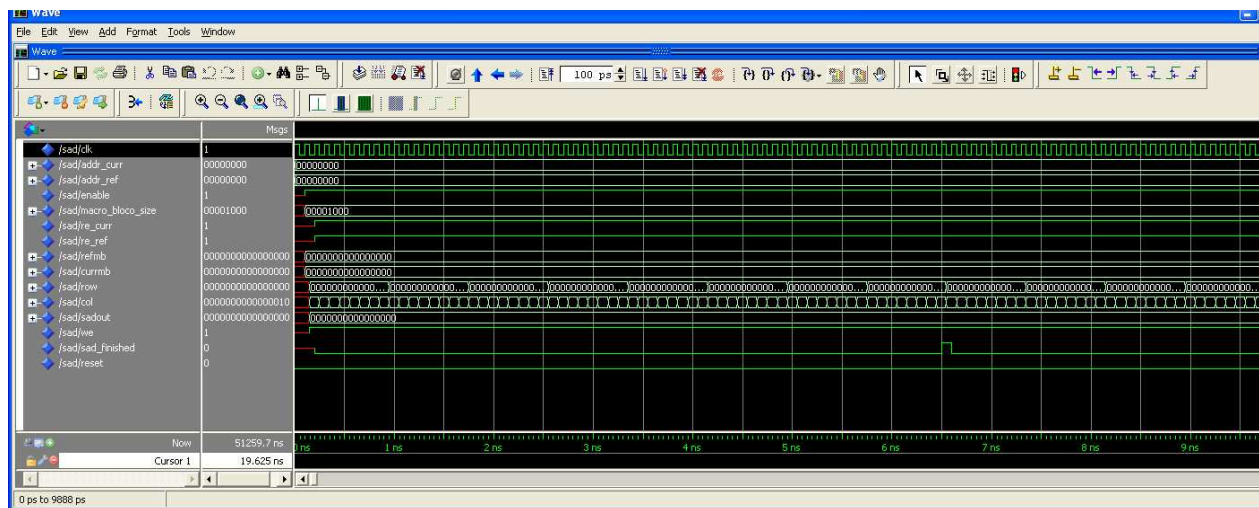


Figura 22 – Cálculo da SAD na FPGA sob controle do DSP

Alguns sinais de controle são necessários para proporcionar o sincronismo entre o DSP e a FPGA. Os sinais de maior destaque, são: CLK, BLOCK_SIZE, WE, ENABLE, SAD_OUT, SAD_FINISHED.

O CLK corresponde ao *clock* do sistema. Este sinal determina a frequência em que as rotinas serão processadas pela FPGA.

BLOCK_SIZE é um sinal enviado pelo DSP que contém o tamanho dos macro blocos a serem percorridos na imagem. Na prática isso se refere ao incremento dos valores i e j descritos em (5.9).

WE é uma abreviação para *write enable*. Ele informa para o DSP que existem dados prontos a serem lidos, essa informação faz com que o controlador de acesso à memória assíncrona do Balckfin ative o sinal \overline{ARE} executando a leitura dos dados.

ENABLE é um sinal de entrada na FPGA, esse sinal é enviado pelo processador. Quando ele for ALTO inicia-se a execução da SAD e quando for BAIXO, deve-se suspender a operação para o quadro de imagem atual. Isso apenas irá ocorrer quando o DSP já tiver encontrado a melhor hipótese para o vetor de movimento do quadro.

SAD_OUT é o valor calculado pela FPGA. E SAD_FINISHED informa ao DSP que todas as posições da imagem já tiveram as hipóteses de vetores de movimento calculados.

5.2.2.1.3 Integração dos Módulos

Após terem sido definidos quais trechos seriam processados pelo Módulo DSP e pelo Módulo FPGA, foi possível realizar a integração do sistema, como ilustrado na Figura 23.

Conforme proposto no Capítulo 4, o princípio de cooperativismo entre DSP e FPGA foi empregado durante a integração dos Módulos e particionamento do código. Mas, mesmo utilizando o princípio de total paralelismo dos processos e havendo um processo mais lento, existem determinadas condições que o processo paralelo consiga devolver os dados computados antes mesmo que eles sejam necessários para o processo principal. Entretanto, dependendo da complexidade do código envolvida e a frequência da execução das rotinas, pode ocorrer o fato em que o processo mais rápido tenha que esperar pelo mais lento, isso é algo factível.

Tendo em vista que durante uma rotina de codificação o DSP já está envolvido em outras operações da cadeia de processos, cabe a ele ser o responsável pela decisão de início e término da execução do código na FPGA. Também, por esse motivo, optou-se em manter a rotina de verificação do melhor vetor de estimação de movimento dentro do Blackfin. Além do mais, caso esse processo fosse desenvolvido em FPGA, iria ocorrer alta dissipação de energia devido a ocorrência de comutação dos sinais em circuitos.

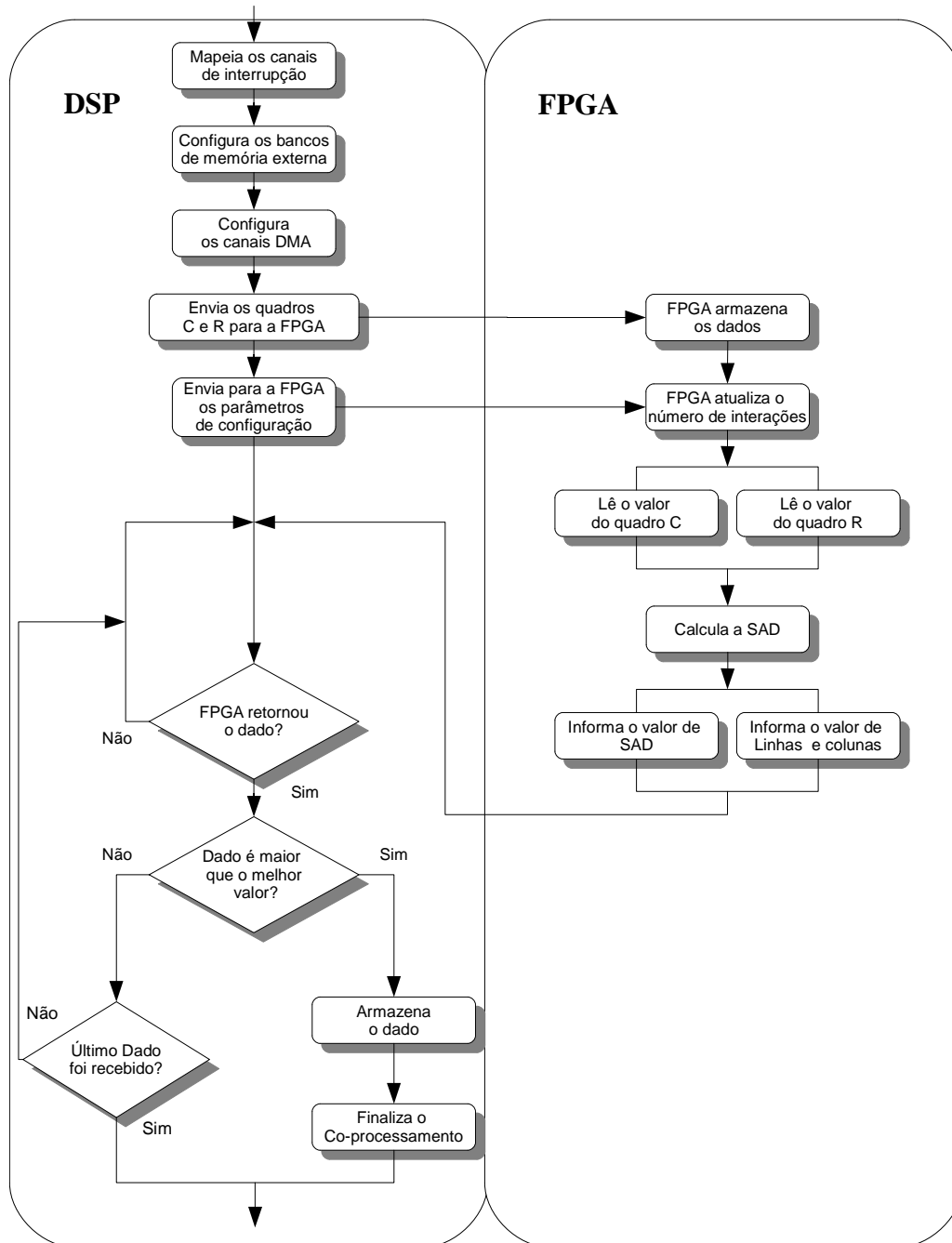


Figura 23 – Integração dos Módulos

Devido às limitações de processamento serial do DSP, os problemas advindos do uso de laços encadeados e a capacidade de exploração de paralelismo e auto-reconfigurabilidade aplicados ao *hardware* programável, optou-se em utilizar a FPGA para processar o código da SAD sem a utilização de laços de repetição e aplicando os conceitos de execução em paralelo.

5.2.2.2 Resultados e Análises

Uma vez definidos os trechos de código executados pelos diferentes Módulos da arquitetura, foi possível comparar algumas métricas para constatar a viabilidade da proposta realizada neste estudo.

A visualização gráfica dos resultados obtidos com a primeira grandeza analisada é apresentado pela Figura 24. A partir do eixo y, expresso em escala logarítmica é possível comparar a quantidade de ciclos de *clock* necessários para a computação da SAD, em relação ao código de referência executado no DSP.

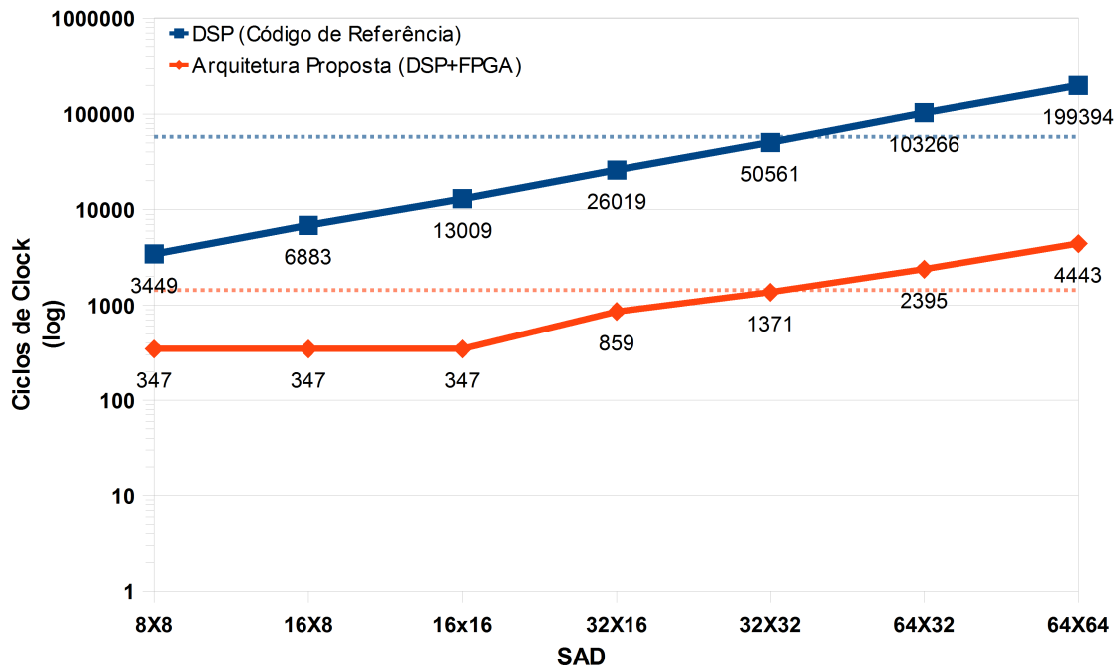


Figura 24 – Ciclos de *clock* para o cálculo de um quadro de imagem

A Figura 24 permite verificar que a quantidade de ciclos de *clock* consumidos pelo sistema híbrido corresponde a um valor muito menor em comparação com a quantidade de ciclos necessários para o DSP computar o código de referência.

O número de ciclos para o sistema híbrido se mantém estável até a computação da SAD 16x16, isso se deve ao fato de que até esse nível, o paralelismo dos processos é pleno, ou seja os

347 ciclos correspondem somente ao processo executado no DSP e consequentemente, o processo da FPGA não afeta o desempenho do sistema. Isso significa que antes mesmo do processador necessitar dos dados a FPGA já os retornou completamente.

A partir do quarto ponto analisado, computação da SAD 32x16, a quantidade de ciclos de *clock* da estrutura computacional híbrida cresce. Mas, em níveis muito abaixo que o necessário para o código de referência. Tal elevação dos ciclos ocorre, porque o processo executado na FPGA passa interferir de forma negativa no desempenho global do sistema, ou seja, o DSP em um determinado momento ficou ocioso, aguardando os valores retornados pela FPGA. Esse tempo de espera, no caso da SAD 32x16, pode ser estimado considerando os 859 ciclos totais do sistema subtraídos dos 347 ciclos do DSP, o que resulta em um total de 512 ciclos.

Outro resultado satisfatório alcançado com o uso da arquitetura é referente à redução efetiva dos ciclos de *clock*, os quais apresentaram um percentual mínimo em torno de 90%. A Figura 25, dispõe de forma gráfica esses valores, de acordo com a complexidade do algoritmo.

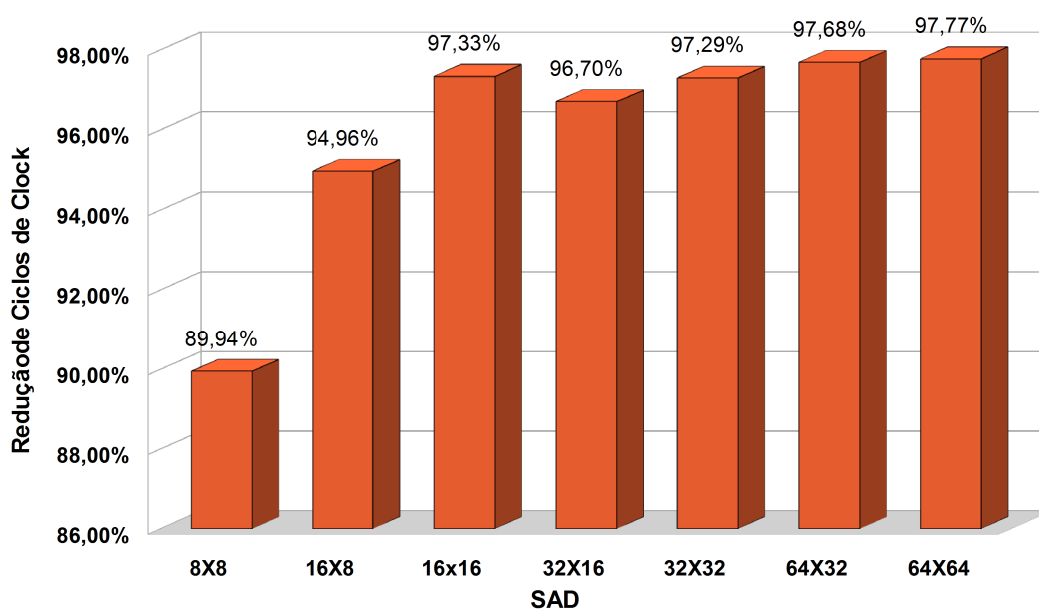


Figura 25 – Percentual de redução de ciclos de *clock*

De forma semelhante ao exposto pela Figura 25, a redução do percentual de ciclos também sofre um impacto a partir do momento em que é necessário para o DSP aguardar a

conclusão do processamento da FPGA. Tal fato é observado na transição da computação da SAD 16x16, onde a redução alcança 97,33% para a SAD 32x16, onde o percentual decresce para 96,70%. Entretanto, a partir desse ponto, mesmo com o DSP em *idle* os percentuais voltam a crescer atingindo o valor de 97,77% de eficiência para a SAD 64x64.

Após essas análises preliminares, representadas pelas Figuras 24 e 25, percebe-se que o potencial de processamento paralelo da arquitetura reduz em níveis elevados a quantidade total de ciclos de *clock* para o processamento do algoritmo de estimação de movimento.

Tendo em vista, que a partir do uso de uma arquitetura híbrida a quantidade de ciclos é reduzida, teoricamente o DSP passa a ter seu potencial de processamento elevado, isso permite que o sistema possa ser atualizado comportando maior carga de processamento sem que haja a necessidade de modificação do *hardware* ou migração para o uso de processador mais potente. Com a finalidade de mensurar a capacidade de expansão do sistema, foi proposto no Capítulo 4 a Equação 4.4, essa formulação permite avaliar a quantidade de MIPS que o DSP poderá calcular a mais em relação ao processamento serial, como o realizado pelo código de referência.

A Figura 26 revela a capacidade de expansão em milhões de instruções por segundo.

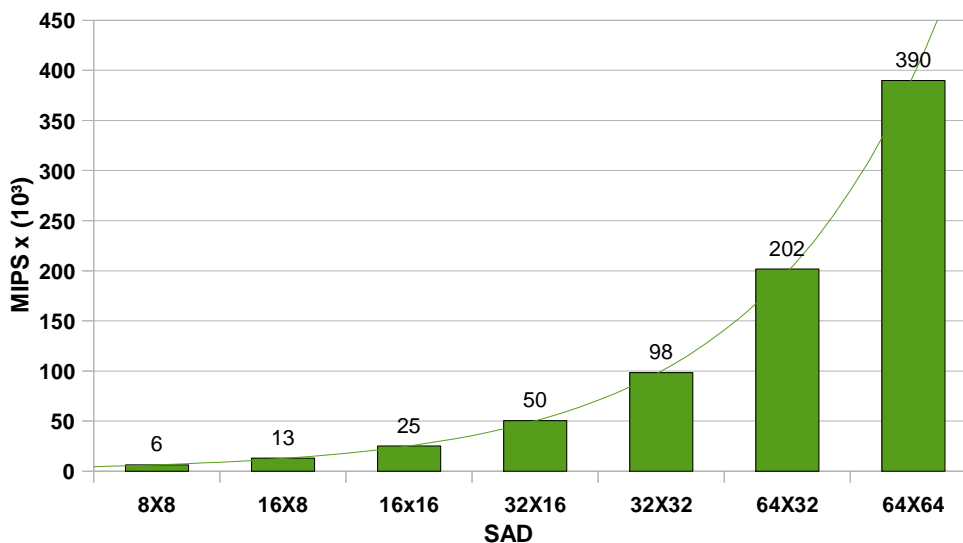


Figura 26 – Capacidade de Expansão em MIPS

Considerando que o Blackfin é capaz de executar duas operações de soma e multiplicação em um único ciclo de *clock*, constata-se que utilizando o modelo de processamento distribuído a capacidade de expansão do sistema em termos de MIPS do DSP cresce exponencialmente conforme a complexidade do código aumenta.

A arquitetura possui diversas variáveis que determinam as características do sistema. Portanto, torna-se necessário mensurar de forma mais precisa o tempo de *idle* do DSP ao longo do processamento. Logo, a Equação 4.7a, proposta no Capítulo 4, calcula de forma criteriosa essa grandeza.

A Figura 27, dispõe graficamente a relação entre o tempo de *idle* do Blackfin e a relação das diferentes frequências de operação de CLK e SCLK do processador, conforme listado na Tabela V.

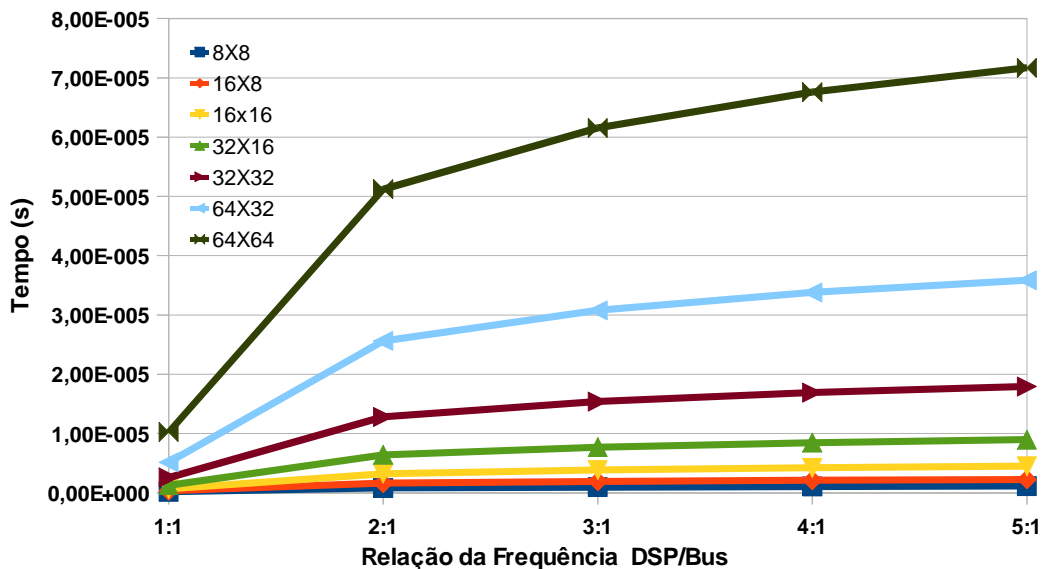


Figura 27 – Tempo de *idle* do DSP

Devido a complexidade do código para o cálculo da SAD 64x64 ser maior em relação à SAD 8x8, o tempo que a FPGA gasta para processar a rotina computacional é muito maior e conseqüentemente o tempo de *idle* do DSP também se torna superior.

Outro aspecto relevante a destacar na Figura 27 é a relação entre a frequência do núcleo do processador CLK e a frequência do barramento de transferência de dados, determinado pelo

clock do sistema (SCLK). É notório que a partir do momento em que as frequências aumentam, também se eleva o tempo de espera do processador. Isso porque a velocidade de processamento para o DSP é maior e, como a frequência de operação da FPGA permanece constante em 80MHz, isso justifica o fato do tempo aumentar para cada um dos diferentes níveis de complexidade do código.

De posse de todas as análises listadas anteriormente, torna-se possível mensurar o ganho efetivo do sistema, considerando todas as variáveis envolvidas no processo de computação descentralizada. Deste modo, faz-se necessário aplicar a Equação 4.8, para avaliar o ganho efetivo total do particionamento do código aplicado à arquitetura proposta. A Figura 28 apresenta esse ganhos expresso em percentuais.

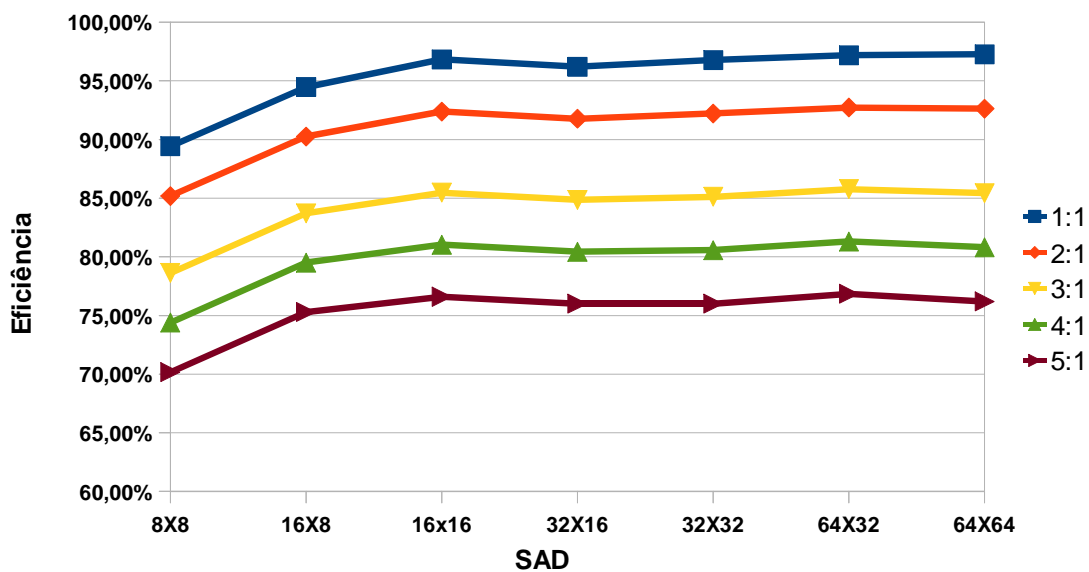


Figura 28 – Eficiência da Arquitetura Proposta

É notório o fato de que quando há um nível de paralelismo total, ou seja, quando o processo de menor frequência de operação (FPGA) não interfere no processo de maior velocidade (DSP) a eficiência computacional cresce rapidamente. Isso pode ser observado na computação da SAD 8x8, 16x8 e 16x16.

Entretanto, mesmo quando o DSP necessita entrar em um estado de espera para capturar os dados retornados pela FPGA, o ganho efetivo permanece crescente, mas, em uma intensidade menor.

Devido existirem níveis aceitáveis para o processador permanecer em *idle*, pode-se observar que a partir do momento em que o tempo de ociosidade do Blackfin atingiu um patamar elevado, a eficiência passou a decrescer, mas mantendo um nível de otimização em torno de 76%, para a relação 5:1 e de 97% para a relação 1:1 entre CLK e SCLK.

Outra análise realizada considera o fato de que em padrões como o H.264/AVC o valor dos macro blocos são variáveis durante o tempo de execução. A partir dessa característica e com base nos diversos tamanhos de macro bloco apresentado pela Figura 28, foi calculado o valor médio da eficiência obtida considerando diferentes frequências de operação da FPGA. Os valores obtidos são apresentados na Figura 29.

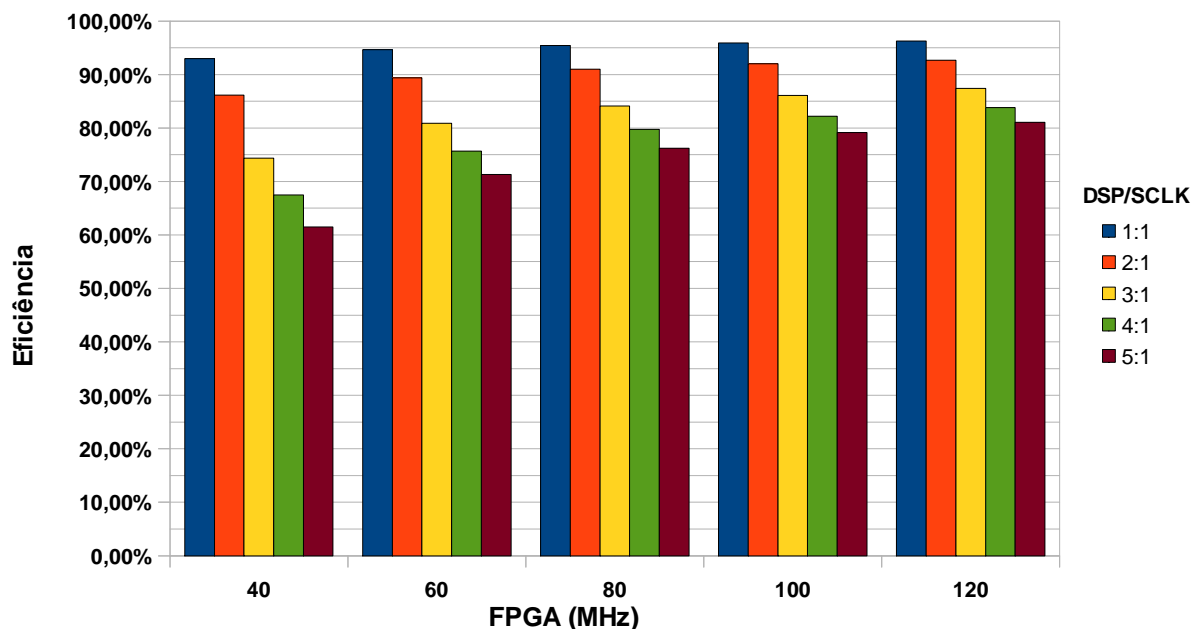


Figura 29 – Eficiência média da arquitetura proposta considerando diferentes frequências de operação de uma FPGA

Primeiramente é possível observar que mesmo a FPGA operando em 40MHz, uma velocidade 15 vezes menor que o DSP, o qual em uma relação 5:1 possui um CLK de 600MHz e

um SCLK de 120MHz, ainda é possível atingir um nível de eficiência média para o cálculo da SAD, superior à 60%.

As análises apresentadas neste estudo ainda permitem identificar qual é o fator limitante ou em outras palavras, qual é o gargalo do sistema que impede a obtenção de maiores níveis de otimização.

Para tanto, é necessário considerar o fato que a FPGA operando em uma velocidade 5 vezes menor que o DSP, ou seja, FPGA em 120MHz, CLK igual a 600MHz e o barramento de memória determinado por SCLK de 120MHz, o valor médio do ganho efetivo ultrapassa 80%. Quando a FPGA opera em uma taxa cerca de 6,65 vezes menor que o DSP, ou seja, FPGA em 60MHz, CLK do DSP em 400MHz e o barramento à 133MHz, obtêm-se o mesmo desempenho, cerca de 81%.

Isso demonstra que o para o caso citado anteriormente, ajustando adequadamente as configurações das frequências dos dispositivos consegue-se a mesma eficiência de desempenho reduzindo a velocidade de operação da FPGA em 50% e elevando o potencial do barramento de comunicação em apenas 11%.

Sabe-se, no entanto, que quanto menor a velocidade de operação da FPGA, menor é o consumo de energia e conseqüentemente menor será a dissipação de calor. A redução desses itens é algo imprescindível para sistemas com escassez de recursos, tais como, sistemas embarcados alimentados por baterias.

6. Conclusões

Este estudo apresentou uma proposta de uma arquitetura computacional híbrida e reconfigurável para o processamento de sinais digitais, a qual permite a computação de algoritmos de forma mais otimizada em relação aos sistemas de processamento centralizado.

O uso cooperativo entre DSP e FPGA se mostra muito vantajoso devido à possibilidade de unir em um único sistema as vantagens fornecidas por esses dispositivos, caracterizando um ambiente de sinergia aplicado à sistemas embarcados.

Além do mais, hoje existem no mercado FPGAs mais baratas que DSPs. Isso demonstra a viabilidade econômica de se utilizar um *hardware* reconfigurável ao invés de se optar por um único processador mais potente e conseqüentemente mais caro que a combinação de um DSP de menor capacidade com uma FPGA de baixo custo.

A sugestão apresentada neste trabalho, além da alta capacidade de processamento, possui como diferencial a capacidade de reconfiguração de circuitos de *hardware* e alto domínio da construção e manutenção de códigos complexos.

Para avaliar a eficiência da proposta descrita neste estudo, foi desenvolvido um cenário de testes que permitiu avaliar o desempenho da arquitetura para acelerar o processo de compressão de vídeo, conforme o padrão H.264/AVC, o qual utiliza o algoritmo de estimação de movimento, que por sua vez é a função de maior custo operacional do processo de codificação.

Durante os testes realizados, os resultados e análises apresentados permitiram concluir que a partir do momento em que houve um aumento na quantidade de macro blocos, ou seja, quando a complexidade computacional do algoritmo de estimação de movimento aumentou, o sistema proposto se tornou mais eficiente em termos de MIPS disponíveis para o DSP, isso mostra a viabilidade da utilização de um coprocessador para otimizar o processamento de rotinas computacionais exaustivas, presentes principalmente em sistemas de tempo real. Nos testes ainda foi possível constatar um ganho efetivo de até 97%.

Outra contribuição relevante desse trabalho é o fato de propor um modelo matemático capaz de mensurar e prever o quanto eficiente será um particionamento de código. Até o presente momento não foi encontrado nenhum trabalho que realize uma avaliação criteriosa para sistemas embarcados de processamento descentralizado.

Por fim, a abordagem realizada neste estudo, permitiu verificar a eficiência de se paralelizar trechos de código que possuem repetidas instruções e elevado consumo de processamento, estes por sua vez, devem ter a prioridade de processamento no módulo constituído pela FPGA o que garantirá um melhor desempenho global do sistema híbrido.

Referências Bibliográficas

1. E. R. Sousa e Meloni, L. G. P. "**High-Performance Computing Based on Heterogeneous Architectures**". *IEEE International Conference on Information Systems and Computational Intelligence - ICISCI 2011*. Harbin, China, janeiro de 2011. ISBN: 978-1-4244-9646-4
2. E. R. Sousa e Meloni, L. G. P. "**FPGA as Coprocessor Improving Performance for Embedded Digital Signal Processing**". *International Conference on Embedded Systems and Intelligent Technology – ICESIT 2011*. Phuket, Tailândia, fevereiro de 2011.
3. E. R. Sousa, J. M. L. Filho e L. G. P. Meloni, "**Arquiteturas Híbridas Reconfiguráveis de Múltiplos Propósitos Baseadas em DSP e FPGA**", Encontro Anual de Computação, Universidade Federal de Goiás, v. 1, p. 1-8, 2010, ISSN 2178-6992, Outubro de 2010.
4. E. R. Sousa e Meloni, L. G. P. "**Exploração de Paralelismo Computacional em Sistemas Embarcados para Compressão de Imagens Digitais**". I Simpósio de Processamento de Sinais da UNICAMP. Universidade Estadual de Campinas-SP, outubro de 2010.
5. Actel, Inc. "**CoreFFT v4.0 Handbook**", maio de 2010.
6. L. V. Agostini, "**Projeto de Arquiteturas Integradas para Compressão de imagens JPEG**", Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, março de 2002.
7. M. Lemes Fh e L.G.P. Meloni. "**Estudo de Técnicas de Otimização da Programação de Códigos de DSP em FPGA**". Dissertação de Mestrado. Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, São Paulo, Brasil, 2009.
8. Altera Inc. "**Nios II Performance Benchmark**", Junho de 2011.
9. F. Franz Rinnerthaler, et al. "**Boosting the Performance of Embedded Vision Systems Using a DSP/FPGA Co-processor System**", *IEEE International Conference on Systems, Man and Cybernetics*, pp.1142–1146, 2007.

10. M. Brogioli, et al. *“Hardware/Software Co-design Methodology and DSP/FPGA Partitioning: A Case Study for Meeting Real-Time Processing Deadlines in 3.5G Mobile Receivers”*, 49th IEEE International Midwest Symposium on Circuits and Systems, San Juan, Porto Rico, 2006.
11. U. Meyer-Baese, *“Digital Signal Processing with Field Programmable Gate Arrays”*, 2ª edição, Springer, Outubro de 2003.
12. K. Parnell e R. Bryner, *“Comparing and Contrasting FPGA and Microprocessor System Design and Development”*, Xilinx Inc., Julho de 2004.
13. Analog devices, *“System Optimization Techniques for Blackfin Processors”*, Julho de 2007.
14. Analog devices, *“Cycle Counting and Profiling”*, março de 2008.
15. D. Zandonai e S. Bampi. *“Uma arquitetura de Hardware para Estimação de Movimetro Aplicada à Compressão de Vídeo Digital”*. Tese de Mestrado, Universidade Federal do Rio Grande do Sul, março de 2003.
16. I. E. G. Richardson. *“H264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia”*, Wiley, 2003.
17. Instituto Fraunhofer. *“H.264/AVC Reference Software Encoder”*, disponível em: <http://iphome.hhi.de/suehring/tml/>, acessado em 08 de abril de 2011.
18. Mentor Graphics, Inc. *“ModelSim Reference Manual”*, disponível em: http://portal.model.com/modelsim/resources/references/modelsim_ref.pdf, acessado em 11 de abril de 2011.
19. Actel Inc. *“SmartGen Cores Reference Guide”*, disponível em: www.actel.com/documents/gen_refguide_ug.pdf, acessado em 11 de abril de 2011.
20. Analog Devices Inc. *“ADSP-BF533 Blackfin Processor Hardware Reference”*, (Part Number: 82-002005-01), revisão 3.4. Norwood, Massachusetts, abril de 2009.
21. Analog Devices Inc. *“VisualDSP++ 5.0 - User’s Guide”*, (Part Number: 82-000420-02), revisão 3.0. Norwood, Massachusetts, agosto de 2007.

22. Analog Devices Inc. *"Blackfin Embedded Processor – ADSP-BF533 High_Speed"*, Norwood, Massachusetts, 2004.
23. Mentor Graphics, Inc. *"ModelSim Tutorial"*, disponível em: http://portal.model.com/modelsim/resources/references/modelsim_tut.pdf, acessado em 11 de abril de 2011.
24. D. A. Petterson e J. H. Hennessy. *"Computer Organization and Design: The Hardware/Software Interface"*. 4ª edição, Morgan Kaufmann.
25. Digikey Inc. **Cotação de valores para diversos dispositivos**. Realizado em: <http://www.digikey.com>, em 13 de abril de 2011
26. J. Nurmi. *"Processor Design: System-On-Chip Computing for ASICs and FPGAs"*. Springer, abril de 2007.
27. P. Gupta e R. Korada *"MPEG-4 Video Encoder on ADI Blackfin DSP for Digital Imaging Applications"*.
28. A. Sudarsanam, A. Dasu e K. Vaithianathan. *"Analysis and Design of a Context Adaptable SAD/MSE Architecture"*
29. D. Tung e G. Yang. *"H.264/AVC Video Encoder Realization and Acceleration on TI DM642 DSP"*
30. Intel, Inc. *"Absolute-Difference Motion Estimation for Intel Pentium 4 Processors"*, disponível em: <http://software.intel.com/en-us/articles/absolute-difference-motion-estimation-for-intel-pentiumr-4-processors/>, acessado em 01 abril de 2011
31. K. Oh Y. Ho *"Adaptive Rate-Distortion Optimization for H.264"*.
32. D. Liu *"Embedded DSP Processor Design"*, Elsevier, 2008
33. Impulse Accelerated Technologies Inc. *"Impulse CoDeveloper C-to-FPGA Tools"*. Disponível em: http://www.impulseaccelerated.com/products_universal.htm, acessado em 26 de abril de 2011.
34. G. M. Amdahl. *"Validity of the single processor approach to achieving large scale computing capabilities"*. AFIPS Spring Joint Computer Conference, 1967.

35. J. L. Gustafson. "*Reevaluating Amdahl's Law*", *Communication ACM*, 1988. pág. 532 – 533.
36. Altera Inc. "*Developing and integrating FPGA coprocessors*", 2003.
37. Dake Liu, Morgan Kaufmann. "*Embedded DSP Processor Design: Application Specific Instruction Set Processors*", 2008.
38. Yu Hen Hu, "*Programmable Digital Signal Processor: Architecture, Programming and Applications*". CRC Press, 1º edição, 2001.
39. R. Oshana. "*DSP Software Development Techniques for Embedded and Real-Time Systems*", Elsevier, 3º edição, 2007
40. G. R. Smith. "*FPGAs 101: Everything you need to know get started*". Elsevier, 2010.