

José Matias Lemes Filho

Estudo de Técnicas de Otimização da Programação de Códigos de DSP em FPGA

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica.

Área de concentração: Telecomunicações e Telemática.

Orientador: Luís Geraldo Pedroso Meloni

Campinas, SP
2009

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

L543e Lemes Filho, José Matias
Estudo de Técnicas de Otimização da Programação de Códigos de DSP em FPGA
José Matias Lemes Filho. – Campinas, SP:
[s.n.], 2009.

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

1. Otimização em FPGA 2. Ferramentas de Prototipagem Rápida
3. Xilinx System Generator 4. Matlab/Simulink
5. Códigos de Processamento Digital de Sinal

I. Meloni, Luís Geraldo Pedroso. II. Universidade Estadual de Campinas.
Faculdade de Engenharia Elétrica e de Computação. III.

Título

Título em Inglês: Study of Optimization Techniques for DSPs Codes
Programming in FPGA.
Palavras-chave em Inglês: FPGA Optimization, Rapid Prototyping Tools
Xilinx System Generator, Matlab/Simulink,
Digital Signal Processing Codes
Área de concentração: Telecomunicações e Telemática
Titulação: Mestre em Engenharia Elétrica
Banca Examinadora: Osamu Saotome
Paulo Cardieri
Data da defesa: 31/03/2009

José Matias Lemes Filho

Estudo de Técnicas de Otimização da Programação de Códigos de DSP em FPGA

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

Banca Examinadora:

Prof. Dr. Luís Geraldo Pedroso Meloni - FEEC-UNICAMP

Prof. Dr. Osamu Saotome - CTA-ITA-IEE

Prof. Dr. Paulo Cardieri - FEEC-UNICAMP

Campinas, SP
2009

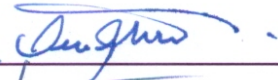
COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: José Matias Lemes Filho

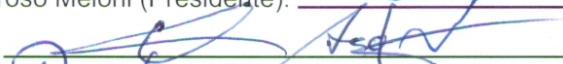
Data da Defesa: 31 de março de 2009

Título da Tese: "Estudo de Técnicas de Otimização da Programação de Códigos de DSP em FPGA"

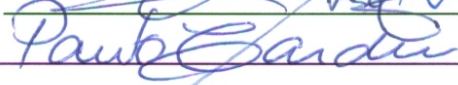
Prof. Dr. Luís Geraldo Pedroso Meloni (Presidente):



Prof. Dr. Osamu Saotome:



Prof. Dr. Paulo Cardieri:



Resumo

Este trabalho descreve o estudo, a pesquisa e compilação de técnicas de otimização de códigos em FPGA (*Field Programmable Gate Arrays*) utilizando uma ferramenta de prototipagem rápida. Para isso, foram implementados alguns algoritmos para auxiliar na apresentação e avaliação de quatro técnicas de otimização: uso de recursos alternativos, multiplexação no tempo, algoritmos alternativos e mudança da frequência sistêmica. As principais contribuições do presente trabalho foram: compilar em um único documento diversas técnicas para geração eficiente de códigos de processamento digital de sinais; o estudo das etapas de fluxo de projeto baseado em ferramentas de prototipagem rápida; implementações de diversos algoritmos para demonstrar as técnicas de otimização, visando-se o estudo da minimização da área de ocupação em FPGA. Com o uso das técnicas pode-se alcançar uma redução de área da FPGA de até 90%, conforme a complexidade do sistema alvo.

Palavras-chave: FPGA, Prototipagem Rápida em DSP, Técnicas de Otimização.

Abstract

This work describes the study, research and compilation of programming optimization techniques for FPGA (*Field Programmable Gate Arrays*) using a tool technology for rapid prototyping. For this purpose, some algorithms have been implemented to help the presentation and evaluation of four optimization techniques: alternative resources usage, time multiplexing, alternative algorithms and systemic frequency change. The main contributions of this work are: compilation in one document several efficient techniques for generation code in digital signal processing; study of the phases of design flow were based on rapid prototyping tools; implementations of several algorithms to demonstrate the optimization techniques, looking for the minimization of the FPGA occupation area. With the use of these techniques, it is possible to reach a FPGA area reduction of up to 90%, depending of the complexity of the target system.

Keywords: FPGA, DSP Rapid Prototyping, Optimization Techniques.

Agradecimentos

A Deus, por inúmeras razões.

À minha família, em especial aos meus PAIS e IRMÃOS, que não permitiram que nada me faltasse durante esses anos, da matéria ou do espírito.

Ao professor Luís G. P. Meloni, por me aceitar como aluno e orientar pacientemente todas as etapas deste trabalho.

Aos professores Alexandre Ribeiro e Antônio Baleeiro, que pavimentaram meu caminho até Campinas.

Aos amigos de ontem e hoje da república: Donatti, Colferai, João Marcos, Rodrigo Nogueira, Walter Biassi, Gussa, André, Clóvis e Killner pela convivência sempre pacífica e agradável.

Aos amigos e amigas do DECOM, por sempre me receberem muito bem em suas reuniões e confraternizações.

Aos colegas do RT-DSP: Lésnir Porto, Erick Rocha e Karlo Lenzi, pela pronta ajuda sempre que necessária.

A FINEP pela bolsa de estudo concedida.

A todos que, embora não citados acima, têm seus nomes escritos nas páginas que narram, em algum lugar, meus dias ao longo desses últimos anos.

Dedico este trabalho aos meus pais...

José Matias Lemes e Zilma Maria Lemes

*...que estiveram sempre presentes na minha vida,
me apoiando em cada decisão e permitindo acreditar
na conquista de mais uma etapa. Obrigado por tudo!*

“Honra a teu pai e tua mãe...”
(Êxodo, 20.12)

Sumário

Lista de Figuras	x
Lista de Tabelas	xii
Glossário	xiv
1 Introdução	1
1.1 Contribuições deste Trabalho	2
1.2 Estrutura da Dissertação	3
2 Considerações iniciais da tecnologia dos PLDs	4
2.1 Visão geral das tecnologias de CIs aplicadas em Sistemas Digitais	4
2.2 Histórico da Evolução dos PLDs	6
2.2.1 <i>Programmable Read-Only Memory</i>	6
2.2.2 <i>Programmable Logic Array</i>	7
2.2.3 <i>Programmable Array of Logic</i>	8
2.3 <i>Programmable Logic Devices</i>	8
2.3.1 <i>Simple Programmable Logic Devices</i>	9
2.3.2 <i>Complex Programmable Logic Devices</i>	9
2.3.3 FPGAs	10
3 Introdução ao FPGA	12
3.1 Arquitetura e Configuração	12
3.2 Tecnologias de Programação em FPGA	15
3.3 Arquitetura de Roteamento	15
3.4 Recursos de DSP em FPGAs	16
4 Ferramentas de Prototipagem Rápida para FPGA da Xilinx	18
4.1 XSG - <i>Xilinx System Generator for DSPTM</i>	18
4.1.1 Implementação Utilizando XSG	19
4.1.2 Compilação Utilizada no XSG	21
4.1.3 Aspectos da Implementação em FPGA	23

5	Técnicas de Otimização em FPGA	24
5.1	Uso de Recursos Alternativos	25
5.1.1	Emprego de Recursos Lógicos	26
5.1.2	Emprego de Recursos de Memória como <i>Look-Up Table</i>	27
5.1.3	Emprego do <i>MCode</i> do XSG	29
5.1.4	Comparação dos Métodos de Implementação	30
5.2	Uso de Multiplexação no Tempo	31
5.3	Uso de Algoritmos Alternativos	36
5.3.1	Algoritmo CORDIC	36
5.3.2	Usando o algoritmo CORDIC em ambiente XSG	39
5.3.3	Algoritmo de Hung	41
5.3.4	Usando algoritmo de Hung em ambiente XSG	41
5.3.5	Comparação entre os Algoritmos	42
5.4	Uso de Recursos DSP48	43
5.4.1	Filtro FIR	43
5.4.2	Filtro FIR utilizando ASR	45
5.4.3	Filtro FIR utilizando ASR com recurso DSP48 da família Virtex-4	47
5.4.4	Comparação entre as implementações	48
5.5	Considerações sobre Frequência Sistêmica	48
6	Conclusão	54
	Referências bibliográficas	56
A	Aritmética do Ponto-Fixo no XSG	58
A.1	Representação com Sinal na Complemento de 2	59
A.2	Fator de Escala em Potência de 2	59
B	Funções do Matlab utilizado no MCode do XSG	61
B.1	Função de Modulação	61
B.2	Função de Demodulação	63
C	Funções calculadas pelo CORDIC	64
C.1	Sistema de Coordenadas - m	65
C.2	Modos de Operação - d_i	66
C.2.1	Modo de Rotação - <i>Z-Reduction</i>	67
C.2.2	Modo de Vetorização - <i>Y-Reduction</i>	68
C.3	Convergência do Algoritmo - γ	69
C.4	Funções CORDIC	70

Lista de Figuras

2.1	Tecnologias diversas para o projeto de sistemas digitais.	5
2.2	Arquitetura de um PROM.	6
2.3	Arquitetura de um PLAs.	7
2.4	Arquitetura de um PALs.	8
2.5	Programação lógica.	9
2.6	Arquitetura de um CPLD.	10
2.7	Estrutura de blocos de uma FPGA destacando recursos de DSP [1].	11
3.1	Arquitetura comum de um FPGA.	13
3.2	Arquitetura simplificada do CLB de um FPGA Xilinx.	13
3.3	Arquitetura do IOB de um FPGA XC4000E da Xilinx.	14
3.4	Arquitetura de dois DSP48 de um FPGA da Xilinx da família 4 [2].	17
4.1	<i>Xilinx Blocksets</i> para o desenvolvimento de sistemas.	19
4.2	<i>Browser</i> de desenvolvimento XSG.	20
4.3	Compilação utilizando do XSG da Xilinx.	21
4.4	Bloco co-simulação criado após a compilação.	22
4.5	Comparação entre bloco de co-simulação e o modelo feito no XSG.	22
5.1	Diagrama de constelação do QAM16.	25
5.2	Sistema de modulação e demodulação QAM16 em ambiente XSG, uso da técnica de emprego de recursos lógicos.	26
5.3	Sistema de modulação e demodulação QAM16 em ambiente XSG, uso da técnica de recursos de memória como <i>look-up table</i>	28
5.4	Estimativa de recursos utilizando o bloco <i>MCode</i> do próprio XSG.	30
5.5	Modelo simplificado da camada física de um <i>Transceiver</i>	32
5.6	Modelo de uma estação base comunicando com estações cliente.	32
5.7	Sistema da BS ou SS em ambiente XSG.	33
5.8	Sistema da BS ou SS em ambiente XSG multiplexada no tempo.	35
5.9	Rotação vetorial.	37
5.10	Algoritmo do CORDIC em ambiente XSG.	40
5.11	Algoritmo interno ao bloco CORDIC em ambiente XSG.	40
5.12	Algoritmo de Hung em ambiente XSG.	42
5.13	Comparação entre Algoritmos de Hung e CORDIC.	43
5.14	Estrutura de um filtro digital FIR.	44

5.15	Algoritmo do filtro FIR em ambiente XSG.	45
5.16	Algoritmo do FIR utilizando ASR em ambiente XSG.	46
5.17	Algoritmo do MAC FIR - 4 parcelas em ambiente XSG.	50
5.18	Algoritmo do MAC FIR - 8 parcelas em ambiente XSG.	51
5.19	Relação de ordem do filtro FIR e a frequência de amostragem do sinal de entrada. . .	53
A.1	Representação em ponto-fixa no XSG.	58
C.1	Trajétórias das rotações em cada sistema de coordenadas.	66
C.2	Trajétórias das rotações - modo de rotação.	67
C.3	Trajétórias das rotações - modo de vetorização.	68
C.4	Trajétórias das rotações CORDIC no cálculo do seno de 68°.	71

Lista de Tabelas

4.1	Recursos de uma FPGA família Virtex-4 FPGA XC4VSX35 do <i>kit</i> de desenvolvimento da <i>Nallatech</i>	23
5.1	Estimativa de recursos utilizando recursos lógicos.	27
5.2	Mapeamento das memórias ROMs.	29
5.3	Estimativa de recursos utilizando na técnica de recursos de memória.	29
5.4	Estimativa de recursos utilizando o bloco <i>MCode</i> do próprio XSG.	31
5.5	Comparação dos dados obtidos pela técnica de estimativa de recursos utilizando. . .	31
5.6	Comparação de recursos da transmissão Normal e a Multiplexada.	36
5.7	Ângulos fixos para a seqüência dos números naturais.	38
5.8	Comparação de recursos Algoritmo de Hung e Bloco CORDIC.	42
5.9	Estimativa de recursos para implementação de um filtro FIR.	44
5.10	Estimativa de recursos de um filtro FIR ASR de 16 coeficientes.	47
5.11	Estimativa de recursos de um filtro FIR com DSP48 de 16 coeficientes.	47
5.12	Comparação dos dados obtidos pelos diversos algoritmos FIR.	48
5.13	Estimativa de recursos de um filtro MAC FIR de 4 parcelas e 32 coeficientes.	50
5.14	Estimativa de recursos de um filtro MAC FIR de 8 parcelas e 32 coeficientes.	52
A.1	Obtenção do número negativo a partir do positivo na representação complemento de 2 com três <i>bits</i>	59
C.1	Seqüência de deslocamento do CORDIC.	70
C.2	Funções calculadas pelo CORDIC [3].	71
C.3	Exemplo das iterações CORDIC para o cálculo do seno de 68° [3].	72

Glossário

ASIC	-	<i>Application Specific Integrated Circuit</i>
ASR	-	<i>Addressable Shift Register</i>
BS	-	Estação Base
CI	-	Circuito integrado
CLB	-	<i>Configurable Block Logic</i>
CPLD	-	<i>Complex Programmable Logic Device</i>
DSP	-	<i>Digital Signal Processing</i>
EDA	-	<i>Electronic Design Automation</i>
FFT	-	Transformada Rápida de Fourier
FPGA	-	<i>Field Programmable Gate Array</i>
HDL	-	Linguagem de Descrição de <i>Hardware</i>
IFFT	-	Transformada Rápida Inversa de Fourier
IOB	-	<i>Input e Output Block</i>
IP	-	Propriedade Intelectual
JTAG	-	<i>Joint Test Action Group</i>
LUT	-	<i>Lookup Tables</i>
MPGA	-	<i>Mask Programmable Gate Array</i>
MUX	-	Multiplexadores
PAL	-	<i>Programmable Array of Logic</i>
PCI	-	<i>Peripheral Component Interconnect</i>
PIA	-	<i>Programmable Interconnect Array</i>
PIP	-	<i>Programmable Interconnect Point</i>
PLA	-	<i>Programmable Logic Array</i>
PLD	-	Dispositivos Lógicos Programáveis
PROM	-	<i>Programmable Read-Only Memory</i>
QAM	-	<i>Quadrature amplitude modulation</i>
RAM	-	<i>Random Access Memory</i>
RAM	-	<i>Random Access Memory</i>
ROM	-	<i>Random Only Memory</i>
SBTVD	-	Sistema Brasileiro de Televisão Digital
SPLD	-	<i>Simple Programmable Logic Device</i>

Glossário

SRAM	-	<i>Static Random Access Memory</i>
SRL	-	<i>Shift Register LUT</i>
SS	-	<i>Estação Cliente</i>
USB	-	<i>Universal Serial Bus</i>
VHDL	-	<i>VHSIC Hardware Description Language</i>
VLSI	-	<i>Very Large Scale Integration</i>
WiMAX	-	<i>Worldwide Interoperability for Microwave Access</i>
XSG	-	<i>Xilinx System Generator for DSPTM</i>

Trabalhos Publicados Pelo Autor

1. Lemes Filho, J. M.; Lenzi, K. G.; Sousa, E. R; Porto, L. F., Meloni, L. G.; “Método Eficiente para Cálculo de uma Função Exponencial em FPGA”. *XXVI Simpósio Brasileiro de Telecomunicações (SBrT'07)*, Recife, Pernambuco, Brasil. Artigo, Setembro 2007.
2. Sousa, E. R; Lemes Filho, J. M.; Lenzi, K. G.; Porto, L. F., Meloni, L. G.; “Compressão e Expansão de Amplitude de Sinais em Sistemas OFDM”. *XXVI Simpósio Brasileiro de Telecomunicações (SBrT'07)*, Recife, Pernambuco, Brasil. Artigo, Setembro 2007.

Capítulo 1

Introdução

Nos últimos anos, a tecnologia de dispositivos FPGAs (*Field Programmable Gate Arrays*) tem evoluído significativamente, alcançando elevados níveis de densidade, altos índices de desempenho e um menor custo de fabricação. Esta evolução tem tornado cada vez menor a distância entre FPGAs e circuitos ASICs (em inglês, *Application Specific Integrated Circuit*). Além dos avanços em capacidade, desempenho e custo, os fabricantes de FPGAs têm introduzido, no decorrer dos anos, cada vez mais ferramentas de reconfigurabilidade dos recursos dos dispositivos.

Entre as ferramentas de reconfigurabilidade, existem algumas ferramentas de prototipagem rápida, que são plataformas de desenvolvimento onde os diversos recursos de um FPGA podem ser reconfigurados, atualizando-se em tempo real seus circuitos, e assim, adequando-os a uma determinada aplicação. Deste modo, o FPGA é uma solução desenvolvida para atender a um grande número de aplicações, combinando-se a velocidade do *hardware* com a flexibilidade do desenvolvimento do *software*.

A programação dos dispositivos pode ser feita através de linguagens de programação (VHDL ou Verilog) ou através de ferramentas de *software* visuais de geração de sistemas, tal como o XSG¹ (*Xilinx System Generator for DSPTM*). O XSG trata eficientemente as especificidades dos recursos mais complexos da FPGA, bem como oferece um roteamento otimizado dos mesmos, desta forma torna-se transparente para o desenvolvedor a tarefa de interconexão dos recursos. Esta ferramenta de prototipagem rápida cria um fluxo de projeto que permite focar no algoritmo que se deseja implementar, ao invés do desenvolvedor se preocupar com os circuitos que serão implementados.

A reconfiguração dinâmica permite também que as alterações da configuração do roteamento da FPGA se deem em tempo de execução, durante a operação do sistema por meio da troca do *bitstream*. Assim, o *hardware* pode ser especializado não somente para uma determinada aplicação, mas também, para diferentes aplicações. De maneira similar, com a evolução dos padrões, uma arquitetura

¹http://www.xilinx.com/ise/optional_prod/system_generator.htm

reconfigurável pode ser vista como um *hardware* dinâmico, em que configurações do FPGA são carregadas e retiradas da arquitetura quando necessário, permitindo inclusive que implementações diferentes em *hardware* possam ser realizadas.

Arquiteturas reconfiguráveis foram projetadas e vêm sendo pesquisadas ao longo das últimas décadas. Estas arquiteturas foram utilizadas para implementar aplicações de diversos projetos e algumas permitem num curto espaço de tempo convalidar um sistema em *hardware*. Alguns exemplos de aplicações de FPGA são: segurança, criptografia, compressão de dados, comunicações, processamento de imagens, processamento de sinais e operações aritméticas complexas.

Este trabalho teve sua origem no projeto do canal de interatividade do SBTVD (Sistema Brasileiro de Televisão Digital), coordenado pelo Professor Luis G. P. Meloni, concluído em 2006. O projeto permitiu a definição de um novo perfil de frequência do WiMAX, abaixo de 1 GHz, que foi denominado WiMAX-700. Foram empregados sistemas de desenvolvimento da *Nallatech* com FPGA *Xilinx Virtex-4*. Todos os resultados apresentados nesta dissertação foram implementadas neste circuito reprogramável.

1.1 Contribuições deste Trabalho

Este trabalho é parte do desenvolvimento de projetos e pesquisas no Laboratório de Processamento Digital de Sinais e Multimídia em Tempo Real (RT-DSP)² da Faculdade de Engenharia Elétrica e de Computação da UNICAMP, cujo objetivo não é apenas a formação acadêmico-profissional, mas também o desenvolvimento do conhecimento tecnológico.

O objetivo deste trabalho é estudar, pesquisar e compilar algumas técnicas de otimização de códigos em FPGA utilizando uma ferramenta de prototipagem rápida. Para isso são implementados alguns algoritmos para auxiliar na demonstração e desenvolvimento destas técnicas de otimização.

As principais contribuições do presente trabalho são:

- Compilar em um único documento diversas técnicas para geração eficiente de códigos de DSP (Processamento Digital de Sinais).
- O estudo das etapas de fluxo de projeto, baseado em ferramentas de prototipagem rápida, visando o roteamento de um dispositivo FPGA.
- Implementações de algoritmos para a demonstração das técnicas de otimização, visando-se minimizar a área de ocupação em FPGA.
- Experimentos mostraram que com o uso das técnicas permite uma redução da área da FPGA de até 90%, segundo a ferramenta de estimativa de recursos utilizada.

²<http://www.rt-dsp.fee.unicamp.br>

1.2 Estrutura da Dissertação

A organização desta dissertação é descrita a seguir. A partir desta introdução, apresenta-se no Capítulo 2, inicialmente, uma síntese da evolução dos PLDs (Dispositivos Lógicos Programáveis) e suas características mais relevantes até a evolução aos FPGAs. É apresentado um breve histórico dos dispositivos reconfiguráveis dedicando atenção especial aos FPGAs no Capítulo 3. São apresentadas ferramentas de prototipagem rápida no Capítulo 4, quatro técnicas de otimização no Capítulo 5, e finalmente, no Capítulo 6, é apresentada a conclusão.

No Apêndice A, é apresentado como o XSG trata os sinais a partir de uma representação em ponto-fixa; no Apêndice B, tem-se o código fonte de uma das técnicas de otimização e, no Apêndice C, as funções que podem ser implementadas pelo algoritmo CORDIC.

Capítulo 2

Considerações iniciais da tecnologia dos PLDs

Este capítulo apresenta uma síntese da evolução dos PLDs (Dispositivos Lógicos Programáveis) até os FPGAs. São indicadas as características mais relevantes dos: PROMs (*Programmable Read-Only Memories*), PLAs (*Programmable Logic Arrays*), PALs (*Programmable Array of Logics*) e PLDs, considerando sua arquitetura simplificada dos blocos lógicos e as estruturas de roteamento. O capítulo é encerrado com uma descrição sucinta dos dispositivos PLDs.

2.1 Visão geral das tecnologias de CIs aplicadas em Sistemas Digitais

Saltos tecnológicos têm transformado de forma significativa todo o processo de projeto de um *hardware*. Um exemplo consiste nos circuitos digitais que evoluíram de transistores discretos, nos quais era necessário que o projetista projetasse adequadamente todas suas interconexões, para CIs (Circuitos Integrados) com milhares de componentes por *chip* chamados de VLSI (do inglês, *Very Large Scale Integration*) [4]. Este último, a partir da utilização de ferramentas denominadas EDA (do inglês, *Electronic Design Automation*) e o aperfeiçoamento dos PLDs, tem simplificado e acelerado o ciclo de projetos do meio acadêmico e industrial. Dessa maneira, com o aperfeiçoamento das HDLs (Linguagens de Descrição de *Hardware*) tornou-se desnecessário o desenvolvimento a nível de portas lógicas individuais, e com isto, o uso de linguagens de descrição de *hardware* reduz-se o tempo e os custos de projeto, facilitando o desenvolvimento [5][6].

Para atender aos requisitos de projetos de CIs, como custo unitário, prazo, tamanho, desempenho e potência, é fundamental fazer a escolha adequada dentre diversas tecnologias existentes. Na Figura

2.1 são apresentadas algumas das principais tecnologias de projetos de circuitos digitais.

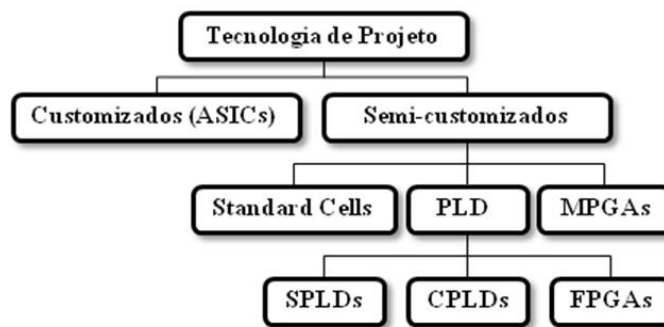


Figura 2.1: Tecnologias diversas para o projeto de sistemas digitais.

Considerando que o foco deste trabalho está relacionado a técnicas de otimização em FPGA, são aqui apresentadas as tecnologias que as precedem:

- *ASICs (Application Specific Integrated Circuits)*: Os circuitos ASICs, também denominados CIs customizados, necessitam de um processo de fabricação especial que requer máscaras específicas para cada projeto. Algumas das características desse tipo de implementação são o alto custo do projeto e o tempo longo de desenvolvimento [7] [8];
- *Standard Cells*: os projetos de CIs, baseados em *Standard Cells*, são facilitados pelo uso de módulos pré-projetados. Esses módulos, ou "cells", são geralmente salvos em banco de dados. O custo de desenvolvimento é baixo comparando com os CIs customizados, citado anteriormente, mas os circuitos são menos eficientes em desempenho e maiores em tamanho [7];
- *MPGAs (Mask Programmable Gate Arrays)*: Essa tecnologia se parece com a tecnologia *Standard Cells*, pois o seu processo de fabricação é agilizado pelo uso de máscaras genéricas de módulos pré-projetados. Conforme a necessidade do projeto, a interconexão é especificada por outra máscara, proporcionando um tempo de desenvolvimento curto e baixo custo em relação aos CIs customizados [7]; e
- *PLDs*: os dispositivos de lógica programável são circuitos integrados que podem ser programados pelo próprio usuário, facilitando assim as mudanças em projetos. Sua principal característica é a capacidade de programação de funções lógicas, eliminando-a do processo de concepção do *chip* e proporcionando as prováveis mudanças de projeto. Os PLDs são tipicamente maiores que os ASICs e podem ter maior custo por unidade e consumo, mas, por outro lado, são excelentes quando é preciso gerar protótipos rapidamente [4] [7];

Dentre as opções disponíveis no mercado, cabe ao projetista pesquisar e selecionar aquela que melhor atenda a necessidade do seu projeto. Em relação às diversas tecnologias para os projetos

de sistemas de circuitos digitais, atualmente, os PLDs constituem possibilidades para soluções e aplicações, as quais se desejam flexibilidade, velocidade e baixo custo de projeto.

2.2 Histórico da Evolução dos PLDs

Na década de 70, simples PLDs eram utilizados na implementação de múltiplos dispositivos lógicos discretos. Hoje em dia, PLDs integram em um único dispositivo uma grande quantidade de blocos capazes de implementar muitas funções lógicas pelo usuário, e são também geralmente preferidos do que os ASICs. O alto volume de dispositivos PLDs confeccionados reduz o custo por unidade, e assim é possível encontrarmos no mercado PLDs com integração, densidade, desempenho, e custo equivalentes ao de um dispositivo não programável. Outro fator que faz com que os PLDs tenham seu custo de confecção reduzido é a utilização de novas tecnologias.

Os dispositivos lógicos programáveis são utilizados para implementar funções lógicas e são programados, por meio de campos elétricos induzidos no dispositivo, por meio de *softwares* especiais fornecidos pelos próprios fabricantes desses produtos. Entende-se por PLD todo circuito de lógica digital configurado pelo usuário final, e os circuitos de lógica digital configurado pelo usuário evoluíram de PROMs, PLAs à PALs e são descritos nas subseções a seguir.

2.2.1 Programmable Read-Only Memory

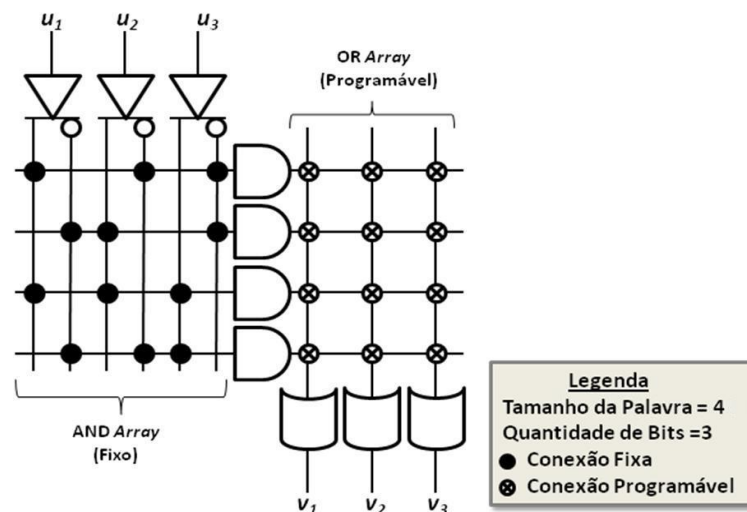


Figura 2.2: Arquitetura de um PROM.

Os PROMs são os primeiros *chips* programáveis capazes de implementar circuitos lógicos por meio de programação realizada pelo usuário. Utiliza-se a tecnologia *fuse* ou *antifuse* para realizar a

programação dos *chips*, entretanto, este é realizado uma única vez, visto que, uma vez programada é irreversível. A Figura 2.2 consiste em dois níveis lógicos: um de portas AND fixas e outro de portas OR programáveis. Demonstra ainda uma arquitetura simplificada que utiliza de linhas de endereço e dados respectivamente com entradas, u , e saídas, v , dos circuitos.

Quando usadas como memória, as PROMs contêm 2^v palavras de n -bits cada. Mas quando as entradas são ligadas a sinais lógicos independentes uma PROM torna-se equivalente a v saídas de circuitos lógicos independentes, cada qual gerando um função das u entradas.

A aplicação da memória PROM, para circuitos integrados, não foi bem aceita em decorrência de dois fatores: o baixo desempenho alcançado pelos circuitos [9], devido aos atrasos impostos pelas estruturas internas de programação das PROMs e a sub-utilização do decodificador de endereços desses dispositivos na implementação de circuitos lógicos muito simples.

2.2.2 Programmable Logic Array

A Texas Instruments¹ criou o termo *Programmable Logic Array* ao desenvolver, em 1970, um componente que é um circuito integrado programável por máscara baseado na memória associativa. Este componente, o TMS2000, era programado alterando-se a camada de metal durante a fabricação. O TMS2000 tinha até 17 entradas e 18 saídas com 8 *flip-flops* como memória. Dessa forma, os PLAs são os primeiros dispositivos desenvolvidos especificamente para a implementação de circuitos lógicos, sendo formados por dois níveis de portas lógicas programáveis, tal como ilustrado na Figura 2.3, onde se tem um plano de portas AND e um plano de portas OR.

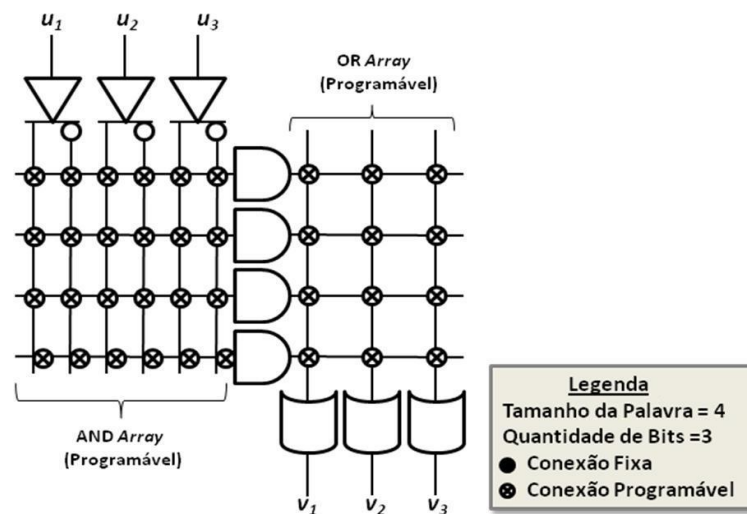


Figura 2.3: Arquitetura de um PLAs.

¹<http://www.ti.com/>

A estrutura dos PLAs é versátil e bastante propícia às implementações de funções lógicas na forma de soma de produtos [9]. Entretanto, os PLAs possuem muitas conexões possíveis, o que encarece o componente.

2.2.3 Programmable Array of Logic

Os PALs (Figura 2.4) foram desenvolvidos devido ao alto custo de fabricação dos PLAs e também pelo desempenho bastante limitado do mesmo que conta com a presença de dois níveis de lógicas programáveis. Os PALs buscaram reduzir os problemas dos PLAs através da simplificação de suas estruturas [9][8], visto que, apresentam apenas um nível lógico programável, um plano AND que se conecta à portas OR fixas.

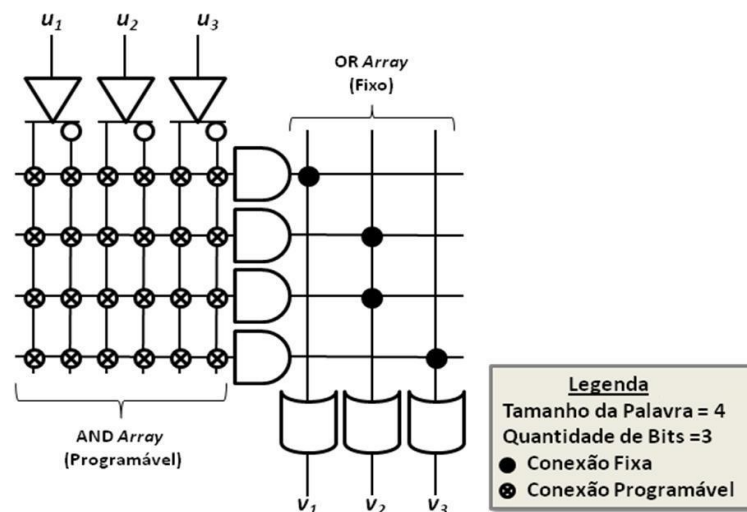


Figura 2.4: Arquitetura de um PALs.

É importante lembrar que diversas variações no número de entradas e saídas desses dispositivos foram fabricadas para compensar a ausência de generalidade imposta pelo plano OR fixo. E, além disso, PALs contendo *flip-flops*, visando a implementação de circuitos sequenciais também foram fabricados [8].

2.3 Programmable Logic Devices

Os PLDs são dispositivos criados para desempenharem inúmeras funções lógicas, sendo programados pelos desenvolvedores [7]. A sua programação é feita através de *softwares* especiais fornecidos pelo fabricante. Conforme podemos ver na Figura 2.5 e nas subseções seguintes, os PLDs tornam-se cada vez mais densos e complexos.

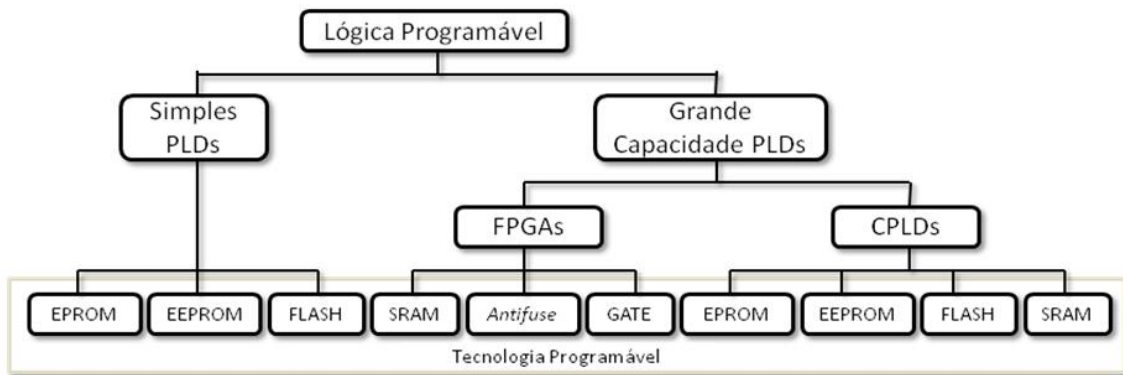


Figura 2.5: Programação lógica.

Estes dispositivos são configurados utilizando comutadores programáveis eletricamente, existindo vários tipos de tecnologia de programação, conforme pode-se ver na Figura 2.5. As propriedades desses comutadores, tais como tamanho, resistência, capacitância, tecnologia, afetam principalmente o desempenho e definem características como volatilidade e capacidade de reprogramação.

2.3.1 Simple Programmable Logic Devices

Os SPLDs (*Simple Programmable Logic Devices*) são *chips* simples, pequenos e baratos quando comparados aos demais tipos de lógica programáveis. Os *chips* são constituídos por 4 a 22 conexões *macrocells* que usam alguma forma de lógica combinacional como portas AND e OR e os *flip-flops*. Suas características mais importantes são baixo custo e alto desempenho [5].

2.3.2 Complex Programmable Logic Devices

Os CPLDs (*Complex Programmable Logic Devices*), Figura 2.6, são dispositivos que apresentam, em geral, maior capacidade de lógica e são obtidos pela interconexão programável de múltiplos SPLDs, integrados num único *chip*. Essa é a estrutura básica de muitos PLDs atualmente disponíveis no mercado, tal como as famílias MAX da Altera e XC9500 da Xilinx, dois dos maiores fabricantes de PLDs da atualidade [5].

Eles são definidos pela combinação de SPLDs. Estes SPLDs consistem em PLAs implementados como uma matriz AND/OR e blocos lógicos de entrada e saída. Os SPLDs usados nos CPLDs são interligados pela matriz de interconexão programável (PIAs - *Programmable Interconnect Arrays*).

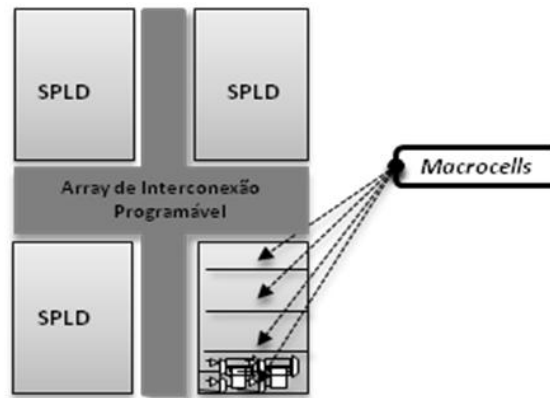


Figura 2.6: Arquitetura de um CPLD.

2.3.3 FPGAs

As FPGAs são arquiteturas reconfiguráveis baseadas em dispositivos de lógica programável, consistindo em uma matriz de blocos lógicos muito simples e regulares cuja funcionalidade é determinada pela configuração desses blocos. Estes blocos são conectados por uma rede de interconexões também programável [5][7], conforme pode-se observar na Figura 2.7. Entretanto, as arquiteturas reconfiguráveis dos FPGAs, ao longo da sua história, raramente foram utilizadas para executar tarefas DSPs (Processamento Digital de Sinais). Até recentemente, os recursos de FPGAs não podiam implementar algoritmos de DSP exigentes e também não havia boas ferramentas de apoio para o desenvolvimento de algoritmos de DSP. Além disso, esses dispositivos eram caros e consumiam muita potência. Entretanto, tudo isto foi mudando com a introdução de novos recursos para DSP nos *chips* de FPGA da Xilinx² e da Altera³.

No ano de 2002, tanto a Altera, que anunciou a família Stratix, quanto a Xilinx, que anunciou a família Virtex-II, passaram a oferecer importantes propriedades apenas presentes nos DSPs. Por exemplo, ambas as famílias de produtos oferecem multiplicadores embarcados reconfiguráveis por toda a matriz lógica que são destinados a acelerar o processo de MAC (Multiplica Acumula) e outras operações comuns em DSP. Ao incluir esses recursos na sua arquitetura reconfigurável, os FPGAs podem melhorar a sua eficiência de consumo de potência, custo e, ao mesmo tempo, oferecer maior desempenho para algoritmos de processamento digitais de sinais. Além disso, ambas as empresas citadas oferecem sofisticadas ferramentas de desenvolvimento tecnológico de técnicas de processamento digital de sinais, tais como as bibliotecas de IP (Propriedades Intelectuais) em blocos com funções de DSP e interfaces com ferramentas visuais de alto nível para processamento digital de sinais junto com o *Simulink/Matlab*. Talvez o mais importante é que esses recursos adicionais na

²<http://www.xilinx.com>

³<http://www.altera.com>

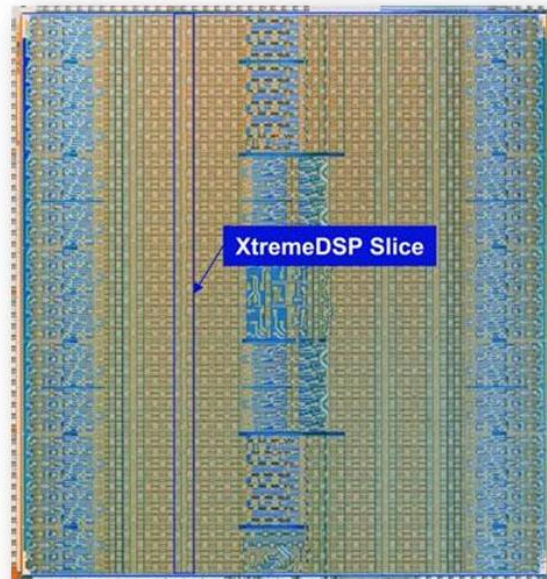


Figura 2.7: Estrutura de blocos de uma FPGA destacando recursos de DSP [1].

arquitetura reconfigurável de um *chip* FPGA têm aumentado a ponto que podem ser executadas até mesmo as tarefas mais complexas dos processadores digitais de sinais.

Os requisitos computacionais dos FPGAs de hoje necessitam de suprir a demanda da tecnologia, principalmente as aplicações de telecomunicações, assim como a procura de processadores DSP cada vez mais rápidos. Isto faz com que novas famílias de FPGAs surjam com uma solução potencialmente atraente para determinadas aplicações. Um dos principais desafios para os projetistas de sistemas é o entendimento e desenvolvimento de novas ferramentas que se adequem a essa nova realidade de dispositivos. Hoje em dia, várias FPGAs comerciais estão disponíveis no mercado, como por exemplo a família Virtex da Xilinx e Stratix da Altera. No próximo capítulo, este tema será melhor detalhado.

Capítulo 3

Introdução ao FPGA

A tecnologia dos FPGAs, nas últimas duas décadas, tem evoluído significativamente, alcançando elevados níveis de densidade, altos índices de desempenho e custos menores de fabricação [5][10]. Com a evolução desta tecnologia, torna-se cada vez menor a distância entre o *chip* do FPGAs e implementação em ASICs.

Além dos avanços em capacidade, desempenho e custo, os fabricantes de FPGAs têm introduzido, no decorrer dos anos, cada vez mais recursos de reconfigurabilidade. Os dispositivos FPGAs possuem como principal característica a capacidade de reconfigurabilidade das suas funções lógicas pelo usuário, facilitando assim, as prováveis alterações do projeto. Outro benefício dos FPGAs é o paralelismo espacial que permite a realização de mais operações por ciclo de relógio. Destacam-se também a ótima flexibilidade dos dispositivos FPGAs e a alta capacidade lógica, requisitos estes que possibilitam a montagem rápida de protótipos de circuitos digitais em menor prazo que os dos métodos precedentes [11]. Neste capítulo, é apresentada uma visão geral dos conceitos fundamentais da tecnologia dos FPGAs como: sua arquitetura, configurações e formas de programação.

3.1 Arquitetura e Configuração

Os FPGAs são circuitos programáveis compostos por um conjunto de células lógicas, ou blocos lógicos, alocados em forma de uma matriz em um único *chip*, vide Figura 3.1. Em geral, a funcionalidade desses blocos, assim como o seu roteamento, é configurável por *software*. A borda externa do arranjo é formada por blocos especiais capazes de realizar operações de entrada e saída.

Um FPGA é estruturado por três elementos básicos:

- CLBs (*Configurable Block Logic*): os blocos lógicos configuráveis são constituídos geralmente por lógicas combinacionais entre registradores, MUX (*Multiplexadores*), LUT (*Look-up Tables*), SRL (*Shift Register LUT*) e RAM (*Random Access Memory*), conforme ilustrados na

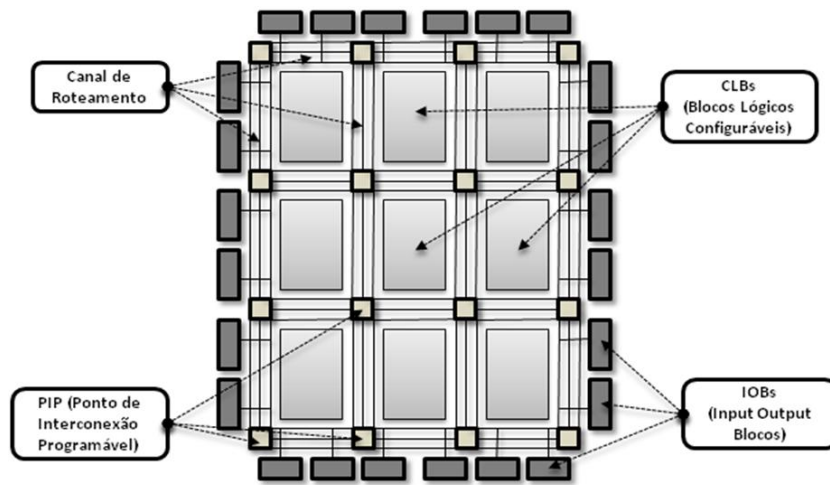


Figura 3.1: Arquitetura comum de um FPGA.

Figura 3.2. Estas configuração de elementos torna este bloco rico em funções, sendo assim, eficaz na construção de diversos circuitos pelo projetista.

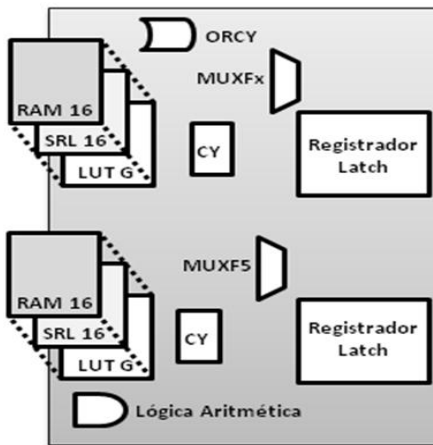


Figura 3.2: Arquitetura simplificada do CLB de um FPGA Xilinx.

- IOBs (*Input e Output Blocks*): nas bordas externas do arranjo existem blocos de entrada e saída que são constituídos por buffers bidirecionais que fazem a interface dos CLBs com o exterior do FPGA. Na Figura 3.3 tem-se uma arquitetura do IOB de um FPGA.
- PIP (*Programmable Interconnect Point*): é responsável pela comunicação entre blocos, sendo este processo denominado de roteamento. Ela ainda forma uma rede que interliga os CLBs e os IOBs, determinando funções estabelecidas pelo usuário final. A reconfigurabilidade das interconexões permite a implementação de diversas topologias entre blocos lógicos. A rede de

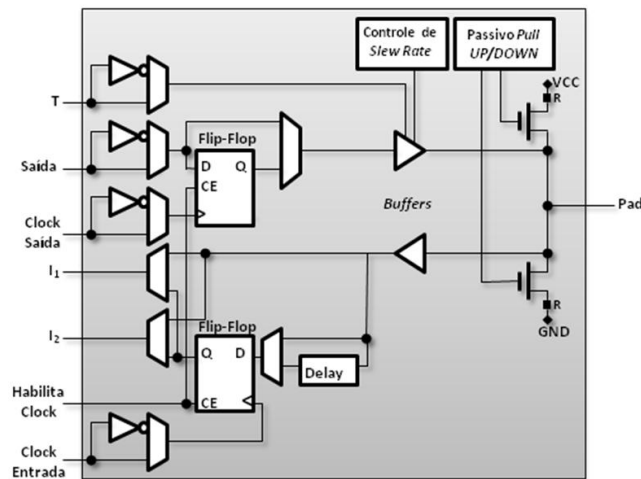


Figura 3.3: Arquitetura do IOB de um FPGA XC4000E da Xilinx.

interconexões contribui significativamente para a área total do componente, e quanto maior for a quantidade de blocos lógicos, mais complexa será esta rede.

A menor unidade lógica configurável é denominada *slice* na família Virtex da Xilinx, que é bastante versátil e pode ser configurado para operar como LUT com quatro entradas e uma saída, RAMs distribuídas de 16 *bits* e registradores de deslocamento de 16 *bits*. Em operações como LUT, os recursos adicionais como *flip-flops D*, multiplexadores, lógica de transporte dedicado, e portas lógicas podem ser utilizados em conjunto com as LUTs para implementar funções booleanas, multiplicadores e somadores com palavras de comprimento bastante flexível. E a operação como SRL (*Shift Register LUT*), tais recursos adicionais podem ser utilizados para se implementar contadores, conversores serial-paralelo e paralelo-serial, entre outras funcionalidades.

Além dos recursos padrões, como *slices*, um FPGA pode disponibilizar no arranjo bidimensional de recursos bastante sofisticados, tais como multiplicadores dedicados, MACs programáveis (multiplicador e acumulador, também denominados de *slice XtremeDSP*), blocos de memória, DCM (*Digital Clock Manager*) utilizados para multiplicar ou dividir a frequência de um sinal de *clock*. A utilização de tais recursos embarcados possibilita otimizar o consumo de área (*slices*) e também desenvolver projetos mais eficientes.

Os recursos adicionais mencionados podem ainda ser utilizados na interconexão de unidades lógicas para se implementar, por exemplo, multiplicadores, contadores, somadores e memórias com praticamente qualquer comprimento de palavra. O limite para este comprimento é determinado pela quantidade de unidades lógicas disponíveis no FPGA, que é proporcional à área do circuito integrado.

Três aspectos principais definem a arquitetura de um FPGA: tecnologia de programação, arquitetura das células e estrutura de roteamento. Nas próximas seções serão descritas com detalhes as

diferentes tecnologias de programação, complexidade dos blocos lógicos e a arquitetura de roteamento.

3.2 Tecnologias de Programação em FPGA

Diferentes tipos de tecnologia de programação são utilizados para se implementar o roteamento em FPGA. Há três tipos de tecnologias de programação atualmente utilizadas:

- Tecnologia *antifuse*: quando programado eletricamente, forma um caminho de baixa impedância, criando, assim, um caminho condutivo entre os eletrodos.
- Tecnologia *gate* flutuante: consiste em comutadores formados por transistor *floating-gates* que pode ser desligado através da injeção de carga; e
- Tecnologia SRAM (*Static Random Access Memory*): consiste em comutadores compostos por um transistor de passagem, controlado pelo estado do bit, armazenado na SRAM.

Estes comutadores programáveis eletricamente são usados para programar um FPGA. As propriedades desses comutadores tais como tamanho, resistência, capacitância, tecnologia, afetam principalmente o desempenho. Outras, características tais como volatilidade e capacidade de reprogramação devem ser avaliadas na fase inicial do projeto para a escolha do dispositivo [4].

3.3 Arquitetura de Roteamento

A arquitetura de roteamento de um FPGA incorpora segmentos de ligação de variados comprimentos, os quais podem ser interconectados via comutadores programáveis eletricamente. A escolha do número de segmentos de ligação a ser incorporado afeta a densidade alcançada pelo FPGA. Se um número inadequado de segmentos é utilizado, tem-se que somente uma pequena fração dos blocos lógicos será utilizada, resultando em uma baixa densidade no FPGA; da mesma forma se um número excessivo de segmentos de ligação for utilizado, tem-se uma baixa utilização dos mesmos e um grande desperdício de área.

A distribuição dos comprimentos dos segmentos de trilhas também afeta de forma significativa a densidade e o desempenho alcançado pelo FPGA. Se todos os segmentos são escolhidos longos, a implementação de ligações locais apresentará um alto custo em área e atraso, por outro lado, se todos os segmentos de ligação forem curtos, as conexões longas serão implementadas de forma a se utilizar muitos comutadores em série, resultando em um grande atraso.

3.4 Recursos de DSP em FPGAs

O recurso denominado *XtremeDSP* nas famílias Virtex-4 compreende duas estruturas denominadas DSP48 eficientes para desenvolvimento de algoritmo de processamento digital de sinal, conforme Figura 3.4. A estrutura possui entradas de 18 *bits*, conjuntos de multiplexadores, multiplicadores (18x18 *bits*), acumulador (Adição e Subtração) e pode ser executada na faixa de frequência de até 500 MHz (FPGA XC4VSX35).

Muitos DSPs possuem multiplicação e adição em sua estrutura, entretanto, os dispositivos Virtex da família 4 apesar de possuírem multiplicadores e somadores dedicados em seu circuitos, não possuem todas as funcionalidades de um DSP comum. Portanto, suportam conexão de múltiplos *slices* DSP48 para formar funções matemáticas, filtros e aritmética complexa sem o uso de outros recursos do FPGA. A sua distribuição em FPGA segue a arquitetura chamada ASMBL (do inglês, *Application Specific Modular Blocks*) que tem como propósito tornar os dispositivos programáveis com uma mistura de lógica, memória, I/O, processador, gerenciador de *clock* e processamento digital de sinais.

Os recursos de DSP48 compreendem em si todos os elementos importantes para funções de processamento digital de sinais [2]. A estrutura de cada DSP48 compreende duas entradas 18 por 18 *bits* que na saída do multiplicador resulta em um valor de 36 *bits*, que são, em seguida, multiplexados (blocos X e Y) e termina com um subtrator/adicionador que aceita 3 operandos de 48 *bits* e produz um resultado de 48 *bits* que pode ser registrado num acumulador. Portanto, a partir dos dados de entrada pode-se produzir diversas combinações e com essas estruturas *XtremeDSP* tem-se a possibilidade da construção de diversas aplicações para processamento digital de sinais. Basicamente, o DSP48 implementa a seguinte equação:

$$P = A \cdot B + C + CIN \quad (3.1)$$

onde o *CIN* é o *carry-in*.

Os controles das operações são realizado pelos multiplexadores X, Y, Z. Além disso, esse recurso *XtremeDSP* fornece uma alta velocidade no barramento que interliga os blocos de DSP48 adjacentes para executar operações matemáticas em palavras superior a 18 *bits*.

No próximo capítulo são apresentadas as ferramentas de prototipagem rápida para FPGA.

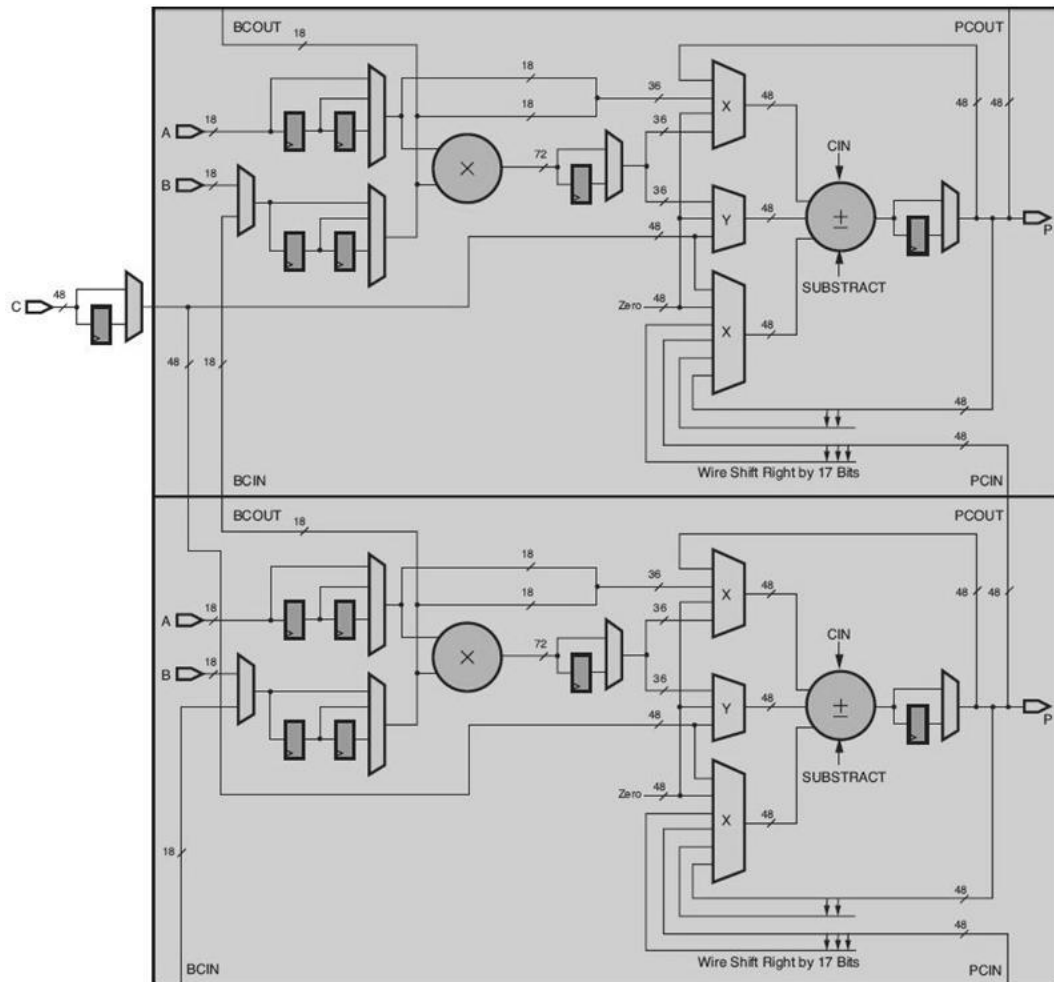


Figura 3.4: Arquitetura de dois DSP48 de um FPGA da Xilinx da família 4 [2].

Capítulo 4

Ferramentas de Prototipagem Rápida para FPGA da Xilinx

4.1 XSG - Xilinx System Generator for DSPTM

O XSG é uma ferramenta de modelagem e simulação visual de sistemas que projetam um fluxo de dados para a implementação eficiente de algoritmos de DSP em FPGAs. Ele se baseia no *MathWorks Simulink^{TM1}* que é uma ferramenta de *software* poderosa do *Matlab*. Com isso, tem-se um ambiente propício para modelagem em tempo discreto de circuitos digitais e também se explora algoritmos específicos para processamento digital de sinal [12]. Da mesma forma que se usam os *blocksets* para modelar um sistema no *Simulink*, pode-se, também, modelar utilizando o XSG. Dessa maneira, bibliotecas são fornecidas aos *blocksets*, que podem ser conectados junto a blocos do *Simulink*, para criar modelos funcionais de um sistema dinâmico, vide Figura 4.1. Os blocos fornecem abstração matemática, lógica, memória, e funções de processamento digital de sinal que podem ser usados para construir sistemas sofisticados de processamento de sinal [13][10][14][15][16].

Os blocos no *Blockset* são organizados a partir de bibliotecas (por exemplo: DSP, comunicações, lógica de controle, etc.). Alguns blocos são de baixo nível, fornecendo a configuração de um dispositivo específico de *hardware*; outros são de alto nível, executando projetos de filtros, algoritmos avançados de processamento digital de sinal e também fornecem blocos que fazem relações com outras ferramentas de *software* (por exemplo: *FDATool²* e *ModelSim³*).

¹<http://www.mathworks.com/products/simulink>

²<http://www.mathworks.com/access/helpdesk/help/toolbox/signal/fdatool.html>

³<http://www.model.com>

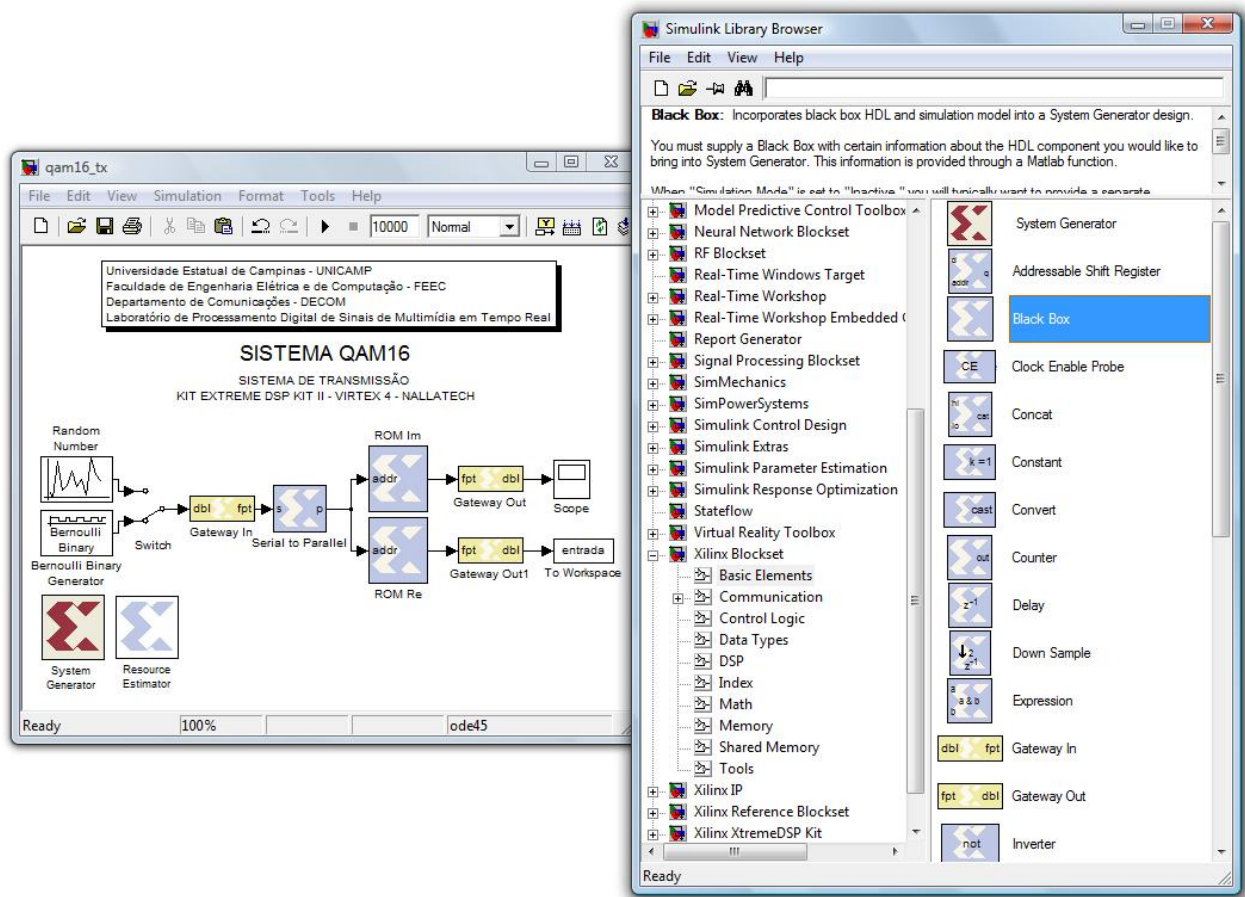


Figura 4.1: Xilinx Blocksets para o desenvolvimento de sistemas.

4.1.1 Implementação Utilizando XSG

Para projetar um sistema usando o XSG, primeiramente, descreve-se este sistema em seu ambiente de desenvolvimento. Como exemplo, na Figura 4.2, tem-se um projeto de um modulador QAM16.

Um sistema desenvolvido em XSG, necessariamente, possui um bloco denominado *System Generator*, disponível no *Xilinx Blocksets* no *Simulink*. Este bloco determina os parâmetros do sistema e da simulação como: o tipo de compilação (por exemplo: *HDL Netlist*, *NGC Netlist*, *Bitstream*, *EDK Export Tool*, *Hardware co-simulação*); a família e o modelo do dispositivo a ser utilizado; o diretório para salvar os resultados da compilação; as possibilidades de síntese do projeto (por exemplo: *Mentor Graphics' Leonardo Spectrum* (apenas com VHDL), *Synplicity's Synplify Pro*, *Synplify*, e *Xilinx's XST*); a linguagem de descrição de *hardware* utilizada (VHDL ou Verilog); o período de relógio de operação do circuito; o endereço de um pino para o clock do circuito; e o período de simulação em ambiente *Simulink* em unidades de segundo.

O emprego de blocos chamados *Gateways In/Out* é a chave para utilização do *System Generator*.

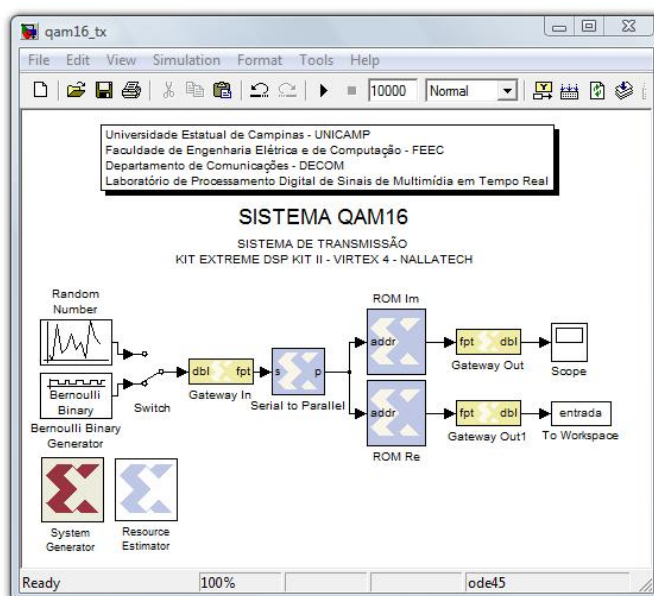


Figura 4.2: *Browser* de desenvolvimento XSG.

O bloco *Gateway In* é a entrada do sistema ou entrada do dispositivo em termos de implementação. Ele representa a interface entre os tipos de dados inteiro, duplo ou ponto-flutuante, no *Simulink*, que são efetivamente sintetizados em *hardware*, em ponto-flutuante. No Apêndice A, tem-se a aritmética de ponto-flutuante empregada no XSG.

Em termos de implementação, cada bloco *Gateway In*, representa uma porta de projeto HDL gerado pelo *System Generator*. No *Gateway Out* ocorre o inverso do *Gateway In*, os dados binários são convertidos em inteiro, duplo ou ponto-flutuante do *Simulink*, conforme seleção pelo usuário. De acordo com as configurações, o bloco *Gateway Out* pode também definir a porta de saída do projeto de HDL gerado pelo *System Generator*, ou pode ser usado simplesmente como um endereço de um pino de teste.

O bloco *Black-Box* é outro recurso disponível no XSG que permite incorporar um modelo HDL ao projeto de um sistema. Este bloco é usado para especificar tanto o comportamento de simulação no *Simulink*, quanto os arquivos de implementação que devem ser usados durante a geração de código com o XSG. As portas geradas por este bloco produzem e aceitam o mesmo tipo de sinal, assim como outros blocos do *System Generator*. É importante ressaltar algumas restrições que o código HDL deve seguir nesta situação, tais como: não pode haver nomes de entidade que coincidam com o de entidades do *System Generator*; não são permitidas portas bi-direcionais nas portas de entrada e saída do bloco *Black-Box*; portas de clock devem ser do tipo *stdlogic*, entre outros.

Alguns blocos geram um atraso nos dados (chamado também de *shift register*), como por exemplo, o bloco *Delay* é inserido para adicionar latência no projeto no fluxo de sinal de forma a ajustar

os tempos de início de processamento entre diversos módulos, e assim, equiparando os dados no mesmo período de tempo. Por último, o bloco *Resource Estimator* faz uma estimativa dos recursos necessários para implementar um sistema ou modelo em XSG em dispositivo FPGA. Os recursos estimados são a quantidade de *lookup tables* (LUTs), *flip-flops* (FFs), *block memories* (BRAM), *18x18 bits multipliers*, *tristate buffers*, e I/Os que um determinado projeto consome.

4.1.2 Compilação Utilizada no XSG

O objetivo do XSG é usar seu conjunto de blocos, que contém códigos de propriedade intelectual (IP), para o roteamento em FPGAs usando *Matlab* e o *Simulink* (Figura 4.3). Ele pode fazer uma extensão de códigos HDL usando o bloco *Black-Box*. O resultado do código HDL pode ser simulado, mapeado e roteado em *chip* FPGA.

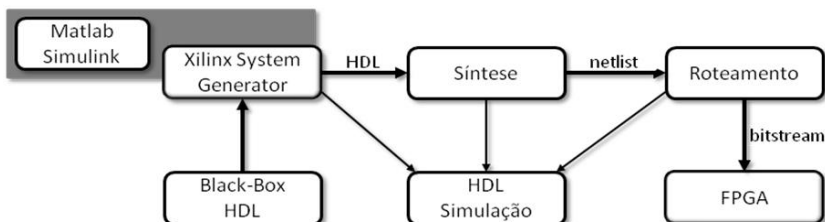


Figura 4.3: Compilação utilizando do XSG da Xilinx.

O XSG gera comandos para a síntese em FPGA, lógica de simulação e ferramentas de implementação (roteamento), além de arquivos necessários para testes e especificações do sincronismo [10]. Uma alternativa de saída de arquivo de compilação usando o bloco *System Generator* é o modo *bitstream*, visto que este, a partir do *System Generator*, gera diretamente o arquivo binário para ser gravado na FPGA. No caso deste tipo de compilação, uma observação é importante: todos os blocos *Gateway In* e *Gateway Out* devem ser informados sobre os pinos do dispositivo para os quais serão mapeados [17].

Outra alternativa é a compilação para co-simulação em *hardware*. Na Figura 4.4 tem-se o bloco de co-simulação do sistema da Figura 4.3. Este recurso de co-simulação permite que um sistema inteiro seja modelado e apresentado dentro de um único bloco, que pode ser utilizado no ambiente *Simulink*.

O bloco da *Co-Simulation* substitui o sistema do *Simulink* compilado para a co-simulação. A Figura 4.5, que ilustra a resposta da simulação em osciloscópio, mostra o resultado do bloco *Co-Simulation* com comportamento idêntico ao do sistema que o originou, entretanto, sua função é executada no *chip* FPGA.

Quando um modelo é executado com a co-simulação, a biblioteca deste bloco é carregada e

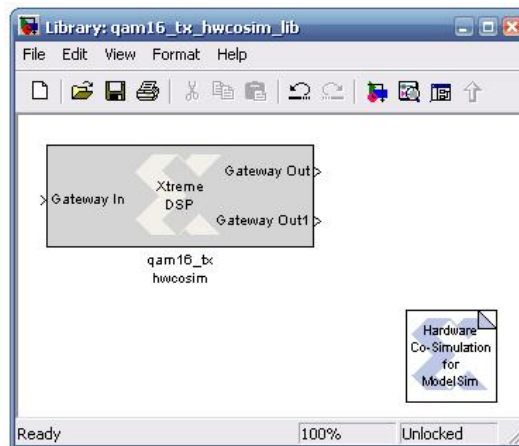


Figura 4.4: Bloco co-simulação criado após a compilação.

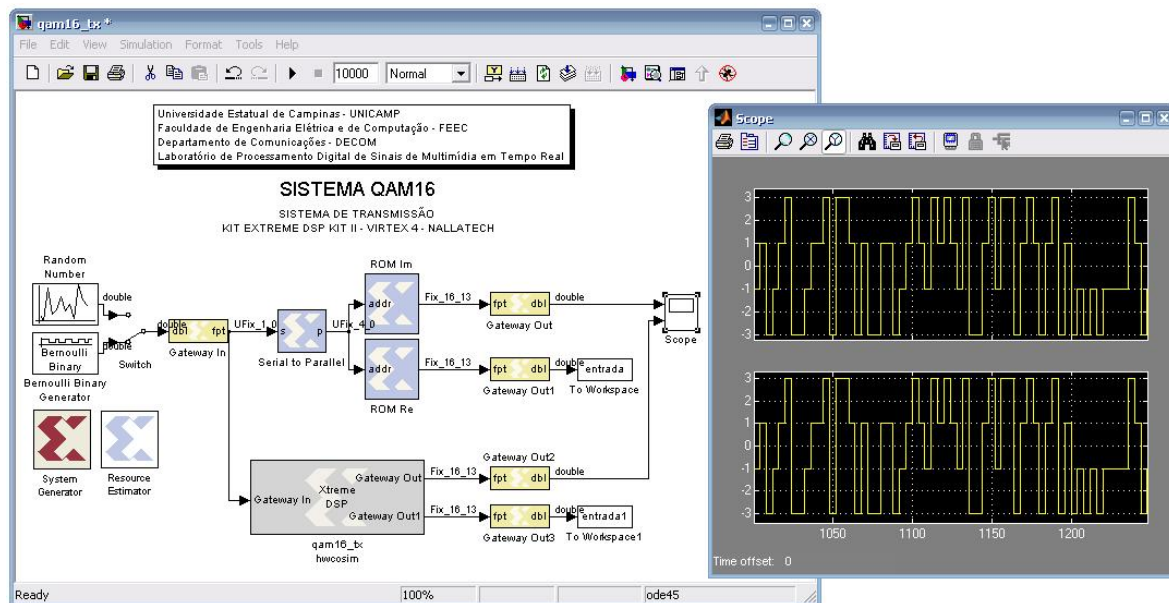


Figura 4.5: Comparação entre bloco de co-simulação e o modelo feito no XSG.

roteada na placa de desenvolvimento *Nallatech Toolkit* durante a simulação em *Simulink*. Os dados da simulação, que são escritos nas portas de entrada do bloco, são passados para o *chip* FPGA. Inversamente, quando os dados são lidos das portas de saída do bloco da co-simulação, este lê os valores do *chip* FPGA e o dirige para as portas de saída, e assim, podem ser interpretados em ambiente *Simulink*.

A interface de transferência de dados entre o FPGA e o microcomputador pode ser feita, durante a co-simulação em *hardware*, através de um barramento PCI, interface USB ou de um cabo JTAG/Paralelo [17]. O que esta compilação faz, na verdade, é gravar o *bitstream* do sistema no

FPGA e disponibilizar um bloco no ambiente *Simulink* que representa o mesmo. Desta forma, diversas vantagens são obtidas: o tempo de simulação diminui, uma vez que o processamento de funções complexas passa do processador do computador para a FPGA; e este recurso torna possível incorporar um FPGA diretamente ao ambiente do *Simulink*. Porém a limitação de frequência entre o *kit* de desenvolvimento da FPGA e o computador torna impossível as simulações de sistemas complexos, pois eles podem operar em diferentes taxas de *clock* e o resultado não condizer com o resultado esperado.

4.1.3 Aspectos da Implementação em FPGA

No próximo capítulo são mostradas as técnicas de otimização, e essas técnicas foram implementadas em uma placa de desenvolvimento, *Nallatech Toolkit*⁴, que é composta de uma placa com dois conversores analógico digital e dois conversores digital analógico, várias interfaces digitais, e uma FPGA família Virtex-4 FPGA XC4VSX35.

Um resumo dos recursos de uma FPGA família Virtex-4 FPGA XC4VSX35 do *kit* de desenvolvimento da *Nallatech* é apresentado na Tabela 4.1.

<i>Recurso</i>	<i>Quantidade</i>
<i>Slice</i>	15360
FFs	30720
BRAMs (kbits)	5760
LUTs	5926
IOBs	448
Emb. Mults	192

Tabela 4.1: Recursos de uma FPGA família Virtex-4 FPGA XC4VSX35 do *kit* de desenvolvimento da *Nallatech*.

⁴<http://www.nallatech.com/>

Capítulo 5

Técnicas de Otimização em FPGA

Neste capítulo, apresentam-se técnicas de otimização de recursos de silício em FPGA fazendo-se uso da ferramenta de prototipagem rápida XSG do fabricante Xilinx, que opera em conjunto com o *Matlab/Simulink*. Para a descrição das técnicas, parte-se do pressuposto que a implementação de um sistema passa por três fases: a primeira consiste em criar um modelo representativo do sistema. Em seguida, um conjunto de algoritmos deve ser implementado para realizar tal modelo e, finalmente, caso necessário, utilizam-se tais métodos de otimização na última fase.

As técnicas de otimização e os algoritmos implementadas são descritos da seguinte forma: inicia-se com a otimização do emprego de recursos para a qual o algoritmo implementado em XSG é uma modulação e demodulação em QAM16. Depois, aborda-se a questão de otimização utilizando multiplexação no tempo, empregando-se um algoritmo de FFT (Transformada Rápida de Fourier), e a otimização algorítmica, comparando-se o algoritmo CORDIC (*COordinate Rotation DIgital Computer*) com o algoritmo de Hung [18]. Por último, apresenta-se a análise de frequência sistêmica, usando os algoritmos de filtragem FIR (Resposta Finita ao Impulso).

Estas técnicas foram utilizadas durante as pesquisas para o desenvolvimento de um sistema de comunicação sem fio em UHF baseado no padrão IEEE 802.16d-WiMAX. A partir da implementação de um *transceiver* numa plataforma FPGA da Xilinx, este novo perfil de frequência abaixo de 1 GHz é denominado WiMAX-700 [19]. O protótipo desenvolvido foi utilizado no âmbito do SBTVD (Sistema Brasileiro de Televisão Digital), como alternativa para o canal de interatividade. O grande mérito deste trabalho é que as técnicas aqui apresentadas não se encontram compiladas na literatura e é resultado de trabalhos especializados do autor e da equipe do laboratório RT-DSP.

5.1 Uso de Recursos Alternativos

Um FPGA, como visto no Capítulo 3, possui diferentes recursos, tais como multiplexadores embarcados, memórias, unidades lógicas especiais, entre outros. Na implementação de um circuito, deve-se levar em conta os elementos disponíveis do FPGA utilizado, para que fatores como ocupação, latência e velocidade sejam otimizados.

Esta seção trata do desenvolvimento de algoritmos que forneçam ao projetista a possibilidade de implementar um código em FPGA otimizado, de forma rápida e eficiente, com a técnica de relocações de recursos. Para tanto, será implementado de três formas diferentes um sistema de modulação e demodulação QAM (Modulação de Amplitude em Quadratura) de 16 pontos utilizando diversos blocos do XSG. O uso desta técnica visa otimizar em área o uso de determinados recursos de uma FPGA.

Será discutida a implementação de uma modulação QAM16 com constelação quadrada, na qual o número de *bits* por símbolo é par, para ilustração dos conceitos de otimização. A modulação QAM16 com constelação quadrada emprega um sinal analítico (parte real - *Re* e parte imaginária - *Im*), sendo que cada parte emprega quatro níveis de amplitude os quais são posteriormente aplicados à entrada do modulador em quadratura, como mostra a Figura 5.1.

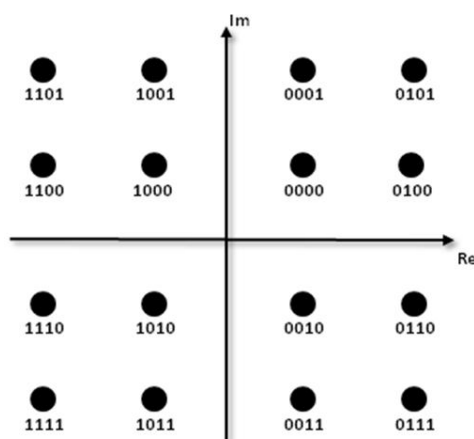


Figura 5.1: Diagrama de constelação do QAM16.

Os 16 pontos do diagrama representam todas as possibilidades que podem ocorrer numa seqüência de quatro *bits*. Nota-se que tanto a amplitude quanto a fase da portadora variam no sinal QAM16. Os dois *bits* mais significativos especificam a posição no eixo real e os dois menos significativos especificam a posição no eixo imaginário. Dessa forma, os *bits* mais significativos 11, 10, 00 e 01 são representados no eixo real pelos símbolos -3, -1, 1 e 3, respectivamente.

5.1.1 Emprego de Recursos Lógicos

Nesta subseção será descrito um sistema de modulação e demodulação com constelação quadrada por amplitude de pulsos que utiliza apenas recursos lógicos de um *chip* FPGA. Na Figura 5.2, está esquematizada, no ambiente do XSG, uma implementação de modulação e demodulação QAM16 com constelação quadrada utilizando apenas blocos de recursos lógicos a serem mapeados em FPGA.

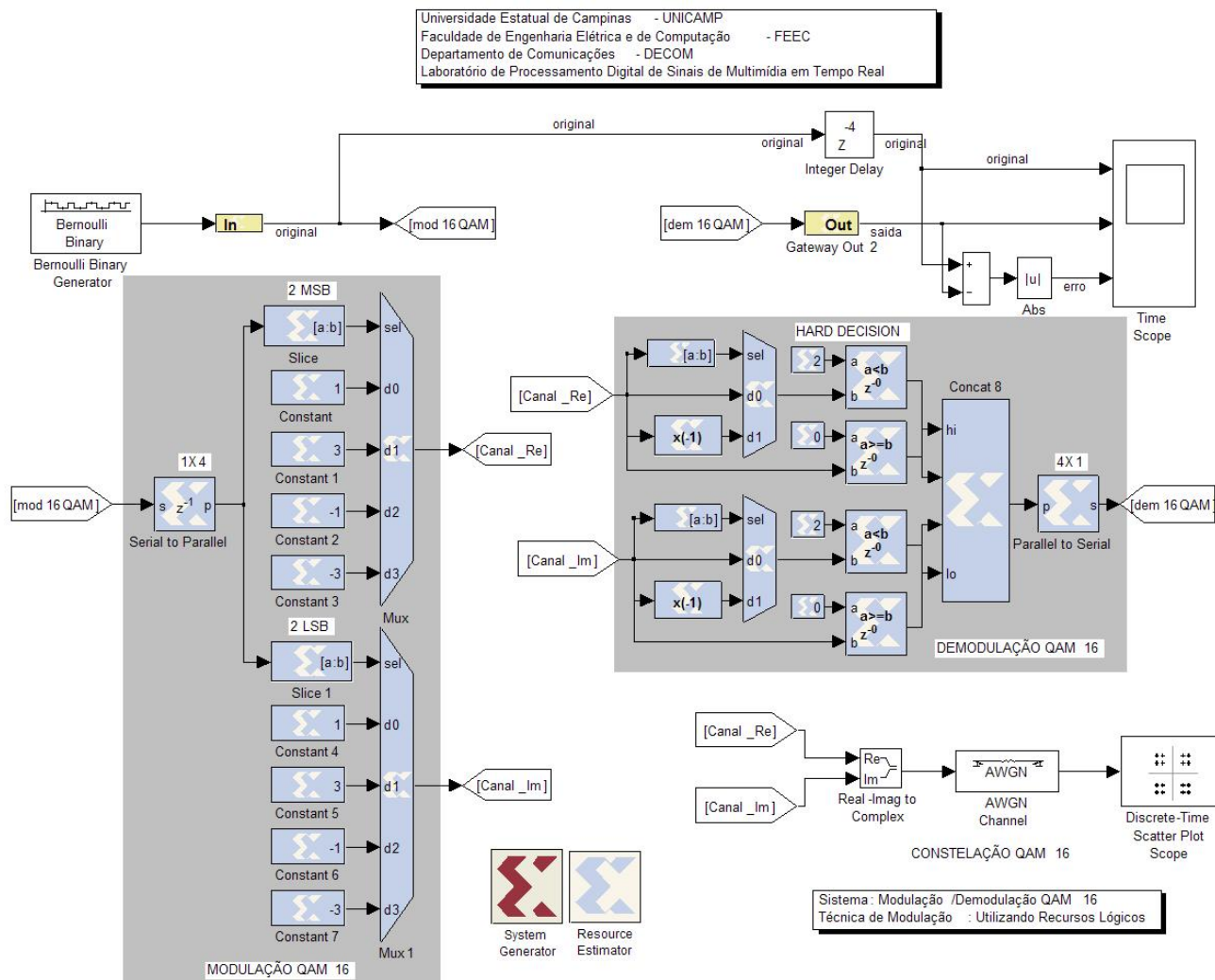


Figura 5.2: Sistema de modulação e demodulação QAM16 em ambiente XSG, uso da técnica de emprego de recursos lógicos.

Na Figura 5.2, o bloco *Serial to Parallel* converte o fluxo de entrada sequencial para um barramento com quatro *bits* paralelos. Os dois *bits* mais significativos são usados para seleccionar pelo multiplexador a parte real da modulação e os dois *bits* menos significativos são usados para seleccionar pelo multiplexador a parte imaginária. Assim, os *bits* de entrada são convertidos para representação

simbólica a fim de serem transmitidos por um canal.

O demodulador executa as funções inversas das que ocorreram no modulador. Foi implementada a decisão por mínima distância, através da utilização de comparadores, que define a qual região de decisão encontra-se o sinal recebido (*Hard Decision*). Na sua implementação é usado um bloco que extrai o bit do sinal com o objetivo de selecionar se o dado é negativo ou positivo. Depois utilizam-se blocos de expressões para selecionar e remontar os *bits* transmitidos de forma adequada. Deve-se observar que este circuito introduz uma latência total de quatro ciclos de *clock*.

Nessa implementação utiliza-se aproximadamente 13 *slices* de lógica em FPGA do fabricante Xilinx, família Virtex-4 FPGA XC4VSX35, conforme Tabela 5.1. Este resultado foi fornecido pela ferramenta *Resource Estimator* do XSG.

<i>Recurso</i>	<i>Quantidade</i>
<i>Slice</i>	13
FFs	12
BRAMs	0
LUTs	12
IOBs	2
Emb. Mults	0
TBUFs	0

Tabela 5.1: Estimativa de recursos utilizando recursos lógicos.

5.1.2 Emprego de Recursos de Memória como *Look-Up Table*

Esta seção ilustra a implementação de um sistema de modulação e demodulação QAM16, mesmo algoritmo implementado anteriormente, só que aqui se utilizam blocos de memória do XSG para implementar este algoritmo. Desta forma, o XSG consegue abstrair recursos mais específicos, neste caso, os BRAMs que estão distribuídos na FPGA e mapeá-los. Como o FPGA permite o carregamento da memória BRAM com valores iniciais, seu comportamento lógico é como uma ROM empregada em *look-up table*. Esse emprego de recursos de memória é uma alternativa de implementar certos algoritmos para se ter um controle de uso de recursos de silício. Assim, o XSG irá gerar um mapeamento que irá conectar várias BRAMs em vez de utilizar as RAMs dos *slices* para implementar o algoritmo desejado. O diagrama esquemático da implementação utilizando os recursos de memória encontra-se ilustrado na Figura 5.3.

Como é possível observar, a entrada é um bloco do *Gerador Binary Bernoulli*. É enviado um *bit* por vez em seqüência e o bloco *Serial to Parallel*, para o endereçamento das memórias das partes reais e imaginárias, convertendo os *bits* de entrada em um barramento de quatro *bits* na saída. Estes

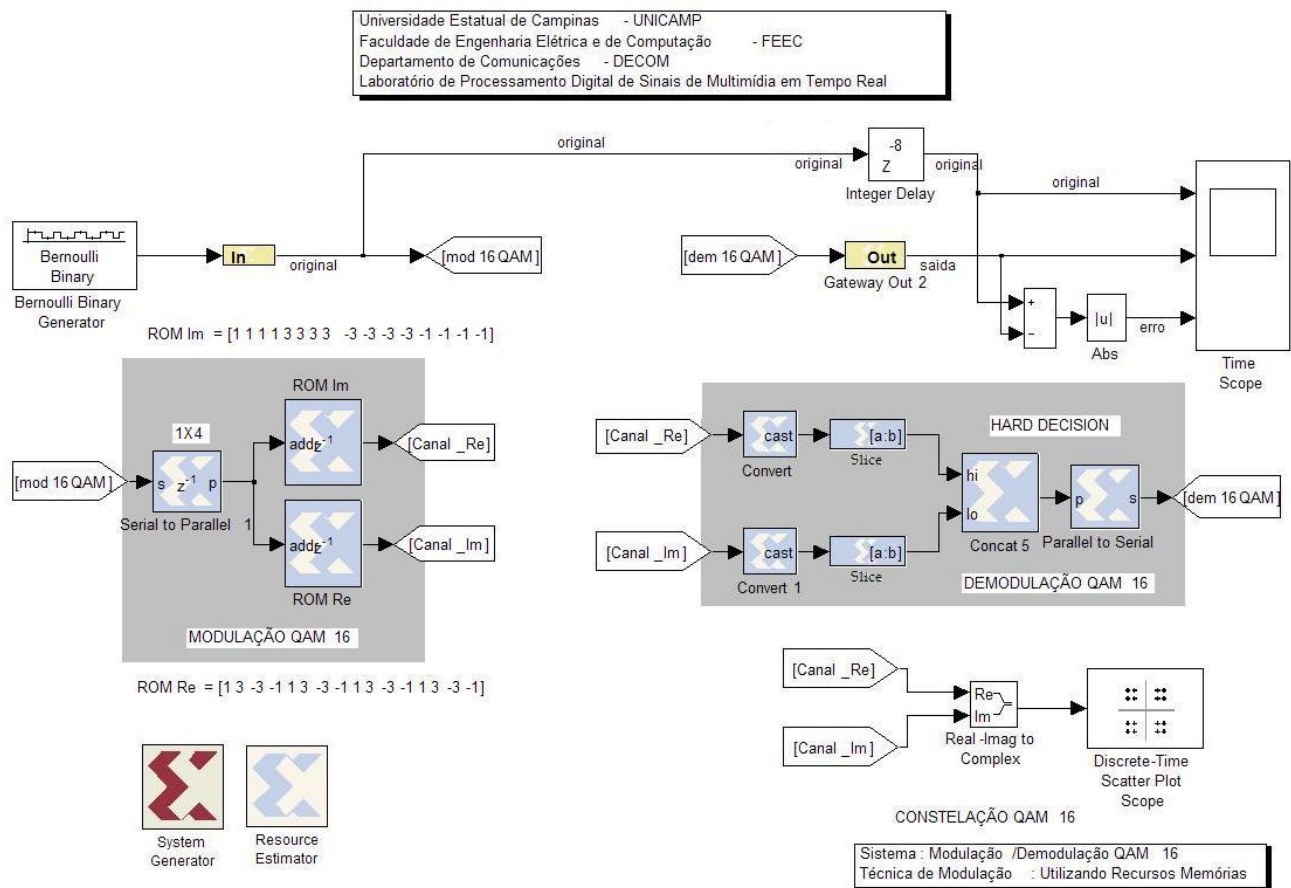


Figura 5.3: Sistema de modulação e demodulação QAM16 em ambiente XSG, uso da técnica de recursos de memória como *look-up table*.

quatro *bits* de entrada formam o endereço das memórias ROM conforme Tabela 5.2, deste modo, mudando a representação da informação de *bits* na entrada para símbolos na saída da modulação.

No início da demodulação do QAM16, é usado o bloco de *Convert* para garantir que o sinal de entrada tem a representação de quatro *bits*, o bloco *slice* superior seleciona apenas os dois *bits* mais significativos, e o inferior, os dois *bits* menos significativos. Depois de concatenar a saída desses dois *slices*, temos os mesmos índices da memória, e desta forma os *bits* transmitidos são remontados de forma adequada. Deve-se observar que este circuito introduz uma latência total de 8 ciclos de *clock*.

Nessa implementação, utilizam-se aproximadamente 8 *slices* de lógica em FPGA e teve um aumento esperado dos recursos de BRAMs do fabricante Xilinx família Virtex-4 FPGA XC4VSX35, conforme Tabela 5.3. Este resultado foi fornecido pela ferramenta *Resource Estimator* do XSG.

<i>Endereço</i>	<i>ROM Im</i>	<i>ROM Re</i>
0	1	1
1	1	3
2	1	-3
3	1	-1
4	3	1
5	3	3
6	3	-3
7	3	-1
8	-3	1
9	-3	3
10	-3	-3
11	-3	-1
12	-1	1
13	-1	3
14	-1	-3
15	-1	-1

Tabela 5.2: Mapeamento das memórias ROMs.

<i>Recurso</i>	<i>Quantidade</i>
<i>Slice</i>	8
FFs	12
BRAMs	2
LUTs	4
IOBs	2
Emb. Mults	0
TBUFs	0

Tabela 5.3: Estimativa de recursos utilizando na técnica de recursos de memória.

5.1.3 Emprego do *MCode* do XSG

A implementação de um sistema QAM16 utilizando uma extensão do *Matlab*, que é o bloco chamado *MCode* do *toolbox* da XSG, está esquematizada no diagrama ilustrado pela Figura 5.4. O que o diferencia dos demais métodos, até então apresentados, é que este utiliza um módulo específico do XSG, chamado *MCode*, para implementar uma função que pode ser desenvolvida a partir do *Matlab*. Mas não são todas as funções do *Matlab* que podem ser desenvolvidas para o *MCode*. Basicamente, este bloco foi projetado para implementar máquinas de estado finito e lógica de controle. Os códigos da modulação e demodulação que estão no Anexo B, utilizam lógica de controle para

implementar um sistema QAM16.

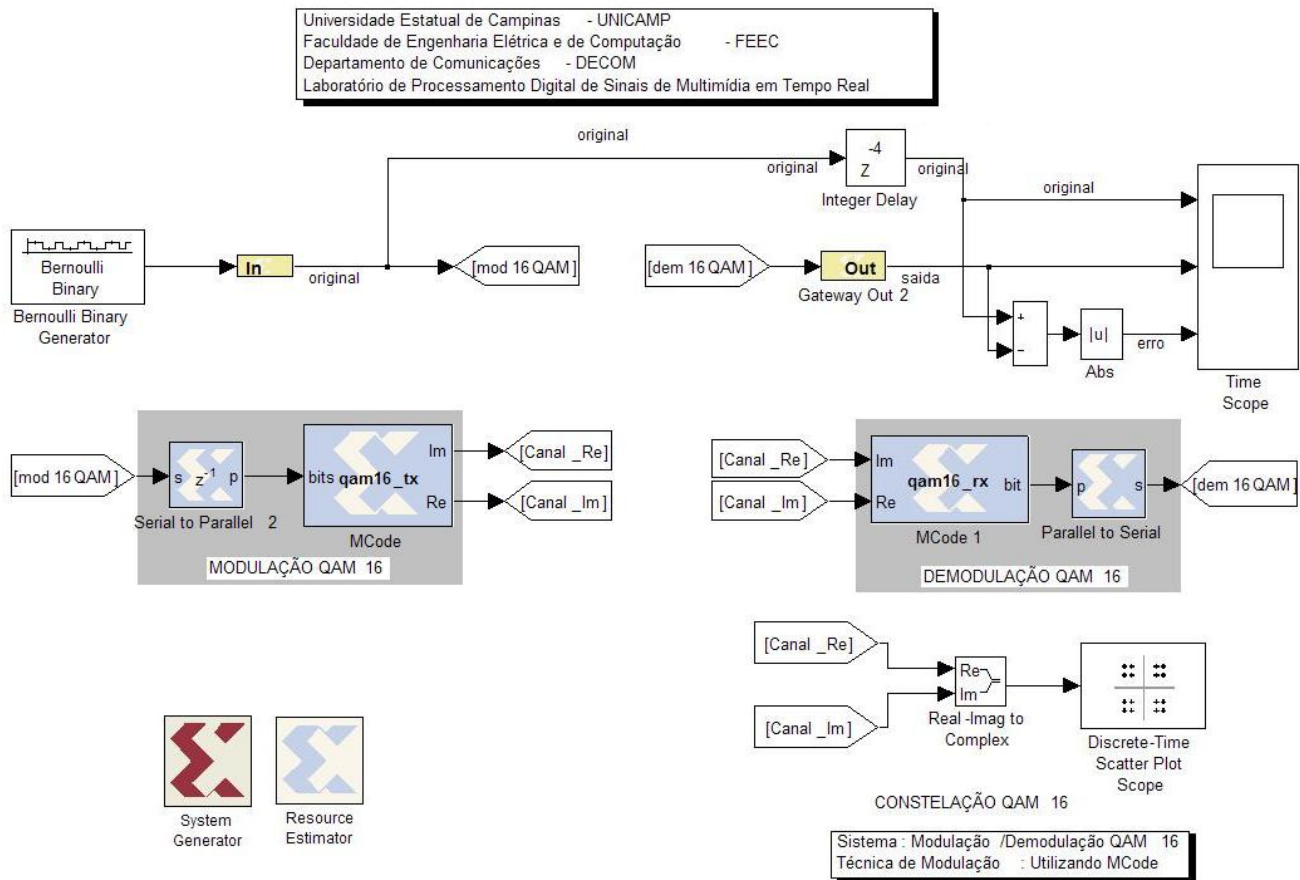


Figura 5.4: Estimativa de recursos utilizando o bloco *MCode* do próprio XSG.

Deve-se observar que este circuito introduz uma latência total de quatro unidades de tempo. Essa implementação utiliza aproximadamente oito *slices* de lógica em FPGA do fabricante Xilinx família Virtex-4 FPGA XC4VSX35, conforme a Tabela 5.4. Este resultado foi fornecido pela ferramenta *Resource Estimator* do XSG.

5.1.4 Comparação dos Métodos de Implementação

Para uma melhor comparação entre os esquemas mostrado, colocou-se na Tabela 5.5 os resultados de todas as implementações referentes a técnica de otimização por uso de recursos alternativos. Aquele que utiliza menos *slices* é o emprego de recursos de memória, em contrapartida, é o que possui maior latência quando comparado aos demais. O emprego de recursos do XSG, utilizando o bloco do *MCode*, e o emprego de recursos lógicos apresentam a mesma latência. Entretanto, o bloco *MCode* apresenta uma redução de *slice*, 13 para 8 unidades, e LUTs, de 12 para 4 unidades, em relação a

<i>Recurso</i>	<i>Quantidade</i>
<i>Slice</i>	8
FFs	12
BRAMs	0
LUTs	4
IOBs	2
Emb. Mults	0
TBUFs	0

Tabela 5.4: Estimativa de recursos utilizando o bloco *MCode* do próprio XSG.

técnica de blocos lógicos.

<i>Recurso</i>	Lógicos	<i>look-up table</i>	<i>MCode</i>
<i>Slice</i>	13	8	8
FFs	12	12	12
BRAMs	0	2	0
LUTs	12	4	4
IOBs	2	2	2
Emb. Mults	0	0	0
TBUFs	0	0	0
Atraso	4	8	4

Tabela 5.5: Comparação dos dados obtidos pela técnica de estimativa de recursos utilizando.

A utilização dos diversos recursos de um dispositivo FPGA deve considerar o algoritmo a ser implementado. Normalmente, certos algoritmos podem causar um grande impacto no custo final, seja em área ou em elementos disponíveis do dispositivo empregado. Essa técnica é de suma importância no entendimento da possibilidade da descrição de algoritmo de maneiras diferentes para reduzir certos recursos que estão sendo utilizados com mais frequência em um projeto.

Na próxima seção, será apresentado um outro método utilizando a mesma idéia de redundância de lógica apresentada em 5.1.1.

5.2 Uso de Multiplexação no Tempo

Nesta seção apresenta-se a técnica poderosa de otimização por multiplexação no tempo utilizando um algoritmo de transformada de Fourier Rápida em um sistema que combina funções para o envio e recepção de sinais. Entretanto, o sistema que combina essas funções é chamado de *transceiver*, e este

é um dispositivo que combina funções de transmissor e receptor utilizando componentes de circuito comuns para ambas funções, como é visto no diagrama de blocos na Figura 5.5.

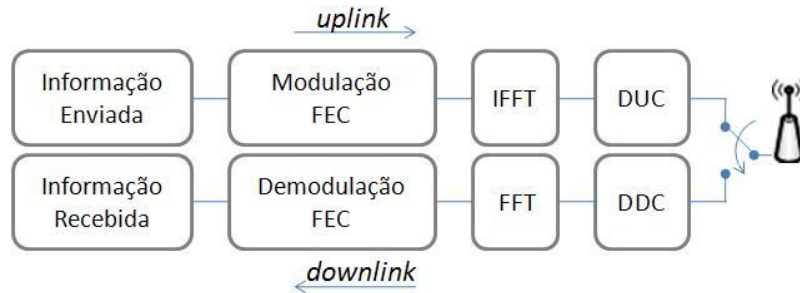


Figura 5.5: Modelo simplificado da camada física de um *Transceiver*.

Uma Estação Rádio Base (BS, do inglês *Base-Station*) transmite para algumas Estações de Usuários (SS, do inglês *Subscriber-Station*), conforme Figura 5.6. As transmissões ocorrem em dois sentidos: um canal de *downlink*, com o fluxo de dados direcionados da BS para a SSs; e outro de *uplink*, fluxo contrário de dados.

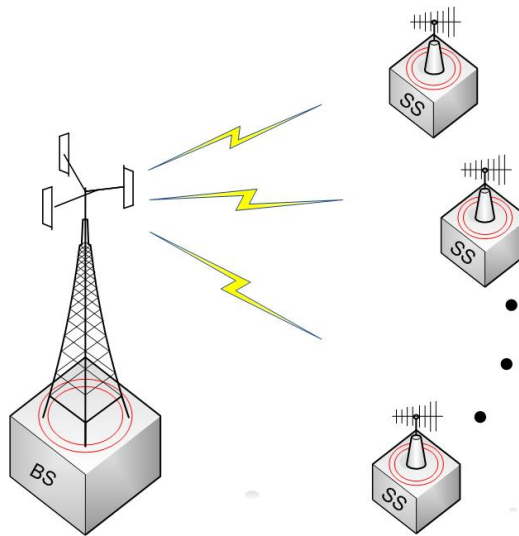


Figura 5.6: Modelo de uma estação base comunicando com estações cliente.

Partindo do modelo da Figura 5.5, estuda-se a questão de otimização utilizando o método de multiplexação no tempo dos algoritmos de transformada rápida de Fourier direta e inversa com blocos do XSG que são utilizados em um *transceiver*. Portanto, não são levados em consideração os blocos referentes à quantização dos dados enviados e recebidos, os blocos de modulação e demodulação e os FEC (do inglês, *Forward Error Correction*) que fazem o tratamento dos *bits* transmitidos, os blocos DUC (do inglês, *Digital Up Converter*) e DDC (do inglês, *Digital Down Converter*) que fazem, como exemplo, a conversão da banda base para uma frequência intermediária.

A modulação OFDM (do inglês, *Orthogonal Frequency-Division Multiplexing*) é uma técnica de modulação baseada na idéia de FDM (Multiplexação por Divisão de Frequência) onde múltiplos sinais são enviados em diferentes frequências e basea-se em FFT para ir do domínio do tempo para o domínio da frequência e vice-versa, combinado com o uso de técnicas avançadas de modulação em cada componente, resultando em um sinal com grande resistência à interferência, denominado OFDM.

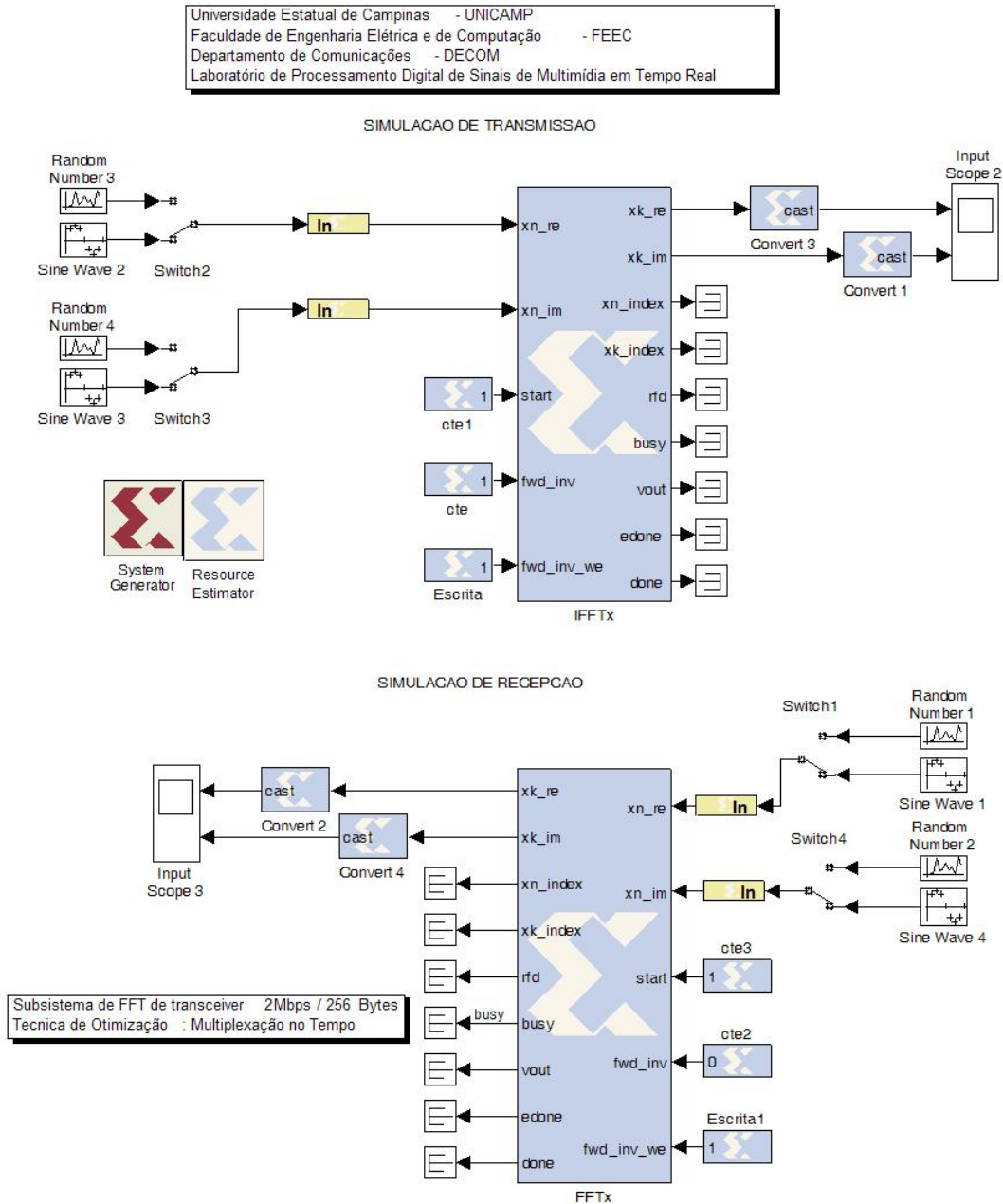


Figura 5.7: Sistema da BS ou SS em ambiente XSG.

Na Figura 5.7, apresenta-se a implementação do canal de *uplink* e o canal de *downlink* das FFTs em blocos do XSG. Para testes de simulação o bloco *IFFT* na Figura 5.7 recebe dois sinais senoidais e transforma esse sinal no domínio da frequência para domínio do tempo; e o bloco da *FFT* recebe dois sinais de dados no domínio da tempo e transforma para o domínio da frequência.

O que diferencia os blocos *IFFT* e a *FFT* na implementação é apenas o valor do *bit booleano* (coloca-se o valor de “0” para a transformada inversa e valor de “1” para a transformada direta) na entrada *fwd_ind* de cada transformada.

No padrão IEEE 802.16d, emprega-se o bloco da transformada de Fourier que recebe um vetor $x(n) = [x(0), x(1), \dots, x(n - 1)]$ de 256 posições de entrada, e é obtido o vetor $X(k) = [X(0), X(1), \dots, X(N - 1)]$, definido por:

$$X(k) = \frac{1}{n} \sum_{m=0}^{n-1} x(m) e^{-jmk2\pi/n}, \text{ para } k = 0, 1, \dots, n - 1. \quad (5.1)$$

E o vetor da transformada inversa de Fourier:

$$x(m) = \sum_{k=0}^{n-1} X(k) e^{jmk2\pi/n}, \text{ para } m = 0, 1, \dots, n - 1. \quad (5.2)$$

Desta forma, calcula-se a transformada rápida de Fourier para a implementação esse sistema utilizando os blocos FFT do XSG.

Utilizando a técnica de otimização por multiplexação, pode-se usar um único bloco do XSG, referenciado por FFTm e assim compartilhar a transmissão e a recepção, conforme Figura 5.8. Entretanto, para que isso seja possível, o bloco tem que operar com o dobro da frequência das amostras de entrada. As amostras que se destinam à IFFT e à FFT chegam em paralelo pelas entradas respectivas *IFFT_in_Re*, *IFFT_in_Im* e *FFT_in_Re*, *FFT_in_Im*. Inicialmente, é feita uma multiplexação desses sinais de forma que o bloco FFTm recebe em um certo momento serialmente o vetor de amostras para IFFT e em outro momento o vetor para a FFT.

O sinal multiplexado possui o dobro da frequência dos sinais de entrada. A entrada *fwd_ind* do bloco FFTm seleciona quando o bloco deve calcular a transformada direta e quando deve calcular a transformada inversa. Após o cálculo da transformada, é feito um escalonamento e um ajuste temporal para colocar cada bloco calculado em uma fronteira de início de um próximo símbolo. Em seguida, é feita a demultiplexação do sinal, de forma que os sinais de saída estão novamente em paralelo, e com a frequência original.

As amostras que se destinam a FFTm chegam em paralelo pelas entradas do bloco *Time Division Multiplexer* e o subsistema *RAM Multiplex* implementa uma multiplexação com uma memória *Dual Port RAM*. Desta forma o bloco FFTm recebe em um certo momento serialmente o vetor de amostras para IFFT e em outro momento o bloco vetor para a IFFT. O sinal multiplexado possui o dobro da

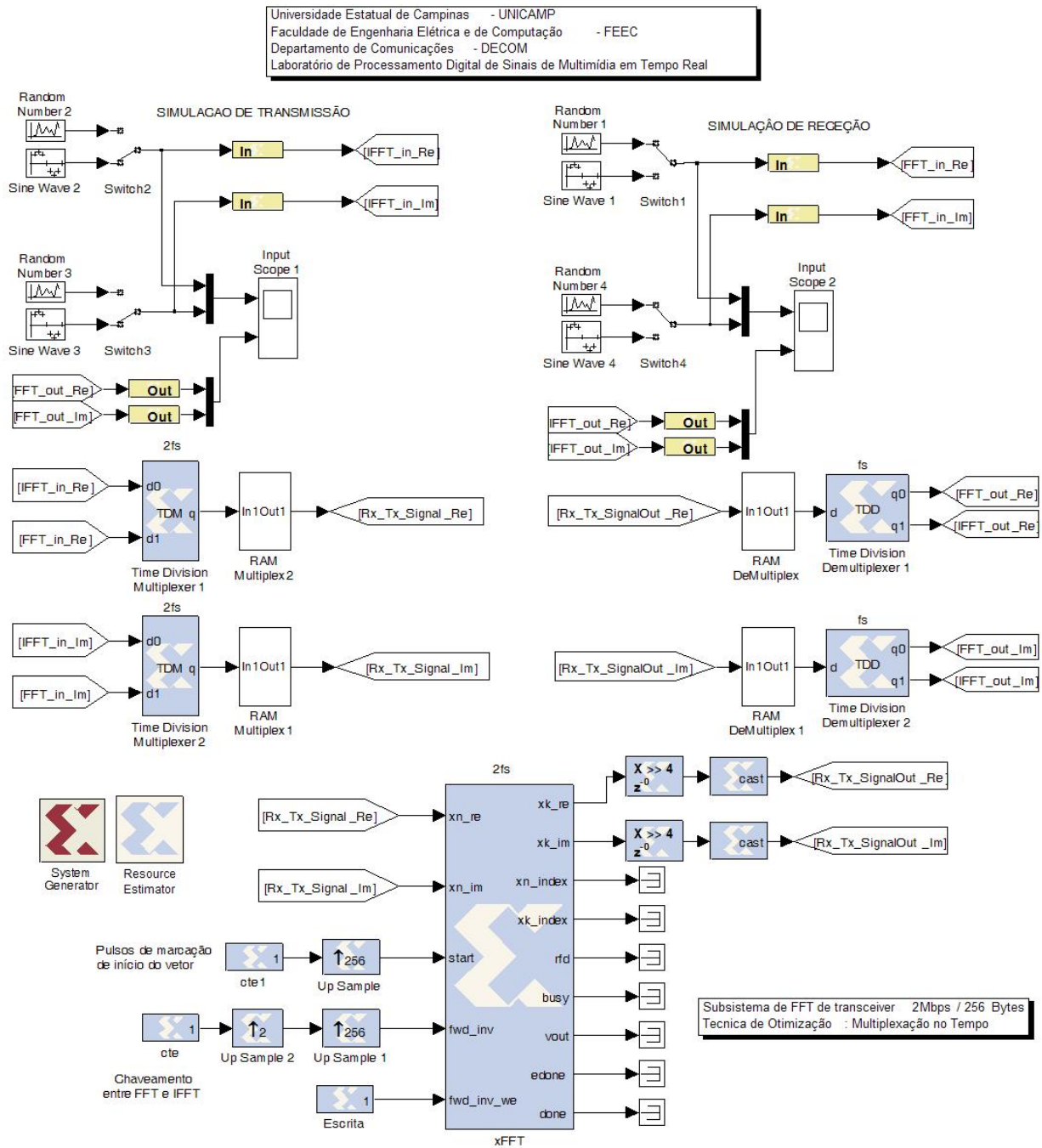


Figura 5.8: Sistema da BS ou SS em ambiente XSG multiplexada no tempo.

frequência dos sinais originais.

A latência total para FFT são de 3 vetores de 256 dados de entrada e para a IFFT são de 4 vetores de 256 dados de entrada. Essa implementação reduz a área cerca de 44,75% de *slices* em uma FPGA família Virtex-4 FPGA XC4V5X35. Os valores de otimização estão na Tabela 5.6. Nesta

implementação, somente a quantidade de memórias BRAMs não foi reduzida, devido à utilização extra das memórias dos blocos *Dual Port RAM* no algoritmo de otimização para aumentar a frequência do sinal.

<i>Recurso</i>	<i>Normal</i>	<i>Multiplexada</i>	<i>Redução(%)</i>
<i>Slice</i>	5021	2774	44,75
FFs	7426	4076	45,11
BRAMs	14	21	-33,33
LUTs	5926	3610	39,08
IOBs	64	64	0
Emb. Mults	32	16	50,00

Tabela 5.6: Comparação de recursos da transmissão Normal e a Multiplexada.

5.3 Uso de Algoritmos Alternativos

Nesta seção faz-se o estudo de uma das técnicas de melhoria de desempenho a partir de um bloco do XSG, chamado CORDIC, que implementa em *hardware* o algoritmo CORDIC (do inglês, *COrdinate Rotation DIgital Computer*) [20]. Nesta seção, este algoritmo é empregado para a divisão em *hardware* de dois números. Foi também implementado alternativamente um outro algoritmo, com alguns blocos do XSG, chamado algoritmo de Hung [18] o qual não está presente no toolbox do XSG. Esses dois algoritmos foram configurados para realizarem o cálculo da inversa de uma função. Ambas implementações realizam a mesma função matemática; porém a consideração de área de silício, bem como a precisão numérica são relevantes.

Será primeiramente descrito o algoritmo CORDIC para calcular o inverso de um número e, posteriormente, será explicado os passos do algoritmo de Hung.

5.3.1 Algoritmo CORDIC

Os algoritmos CORDIC trigonométricos foram desenvolvidos originalmente para soluções digitais para os problemas de navegação em tempo real [21] e o melhoramento do desempenho de determinadas operações aritméticas complexas, possibilitando implementações mais eficientes de arquiteturas e algoritmos no processamento digital de sinais. O trabalho original é creditado a Jack Volder [20], em 1952, que possibilita o cálculo iterativo de funções trigonométricas e transcendentais através de uma técnica de rotações de vetores com operações de soma e deslocamento. A sua primeira aplicação foi substituir os computadores analógicos do sistema de navegação do bombardeiro B-58

por computadores digitais, pois os analógicos eram pouco precisos. Extensões para o CORDIC, em desdobramento baseado no trabalho de John Walther e outros, permitem fornecer soluções para uma ampla classe de funções, tais como relações hiperbólicas e lineares. O algoritmo CORDIC tem diversas aplicações para cálculo de funções transcendentais em hardware [21] [22] [3].

O CORDIC é um algoritmo que utiliza um método iterativo de rotações vetoriais, utilizando apenas deslocamentos e adições. Tal característica o torna muito interessante pela implementação em *hardware*, especialmente sem o uso de multiplicadores. Volder desenvolveu o algoritmo CORDIC baseado na forma geral da matriz de rotação de Givens, que rotaciona um vetor $\vec{V} = (x, y)$ por um ângulo ϕ no plano euclidiano, como é representado na Figura 5.9.

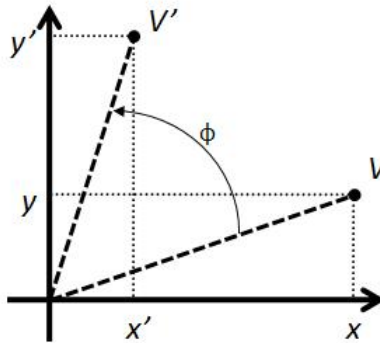


Figura 5.9: Rotação vetorial.

Esta rotação vetorial pode ser expressa por:

$$\begin{cases} x' = x \cos(\phi) - y \sin(\phi) \\ y' = y \cos(\phi) + x \sin(\phi) \end{cases} \quad (5.3)$$

onde $\vec{V}' = (x', y')$ são as novas coordenadas.

As equações acima podem ser modificadas para a seguinte forma:

$$\begin{cases} x' = \cos(\phi)[x - y \tan(\phi)] \\ y' = \cos(\phi)[y + x \tan(\phi)] \end{cases} \quad (5.4)$$

Se os ângulos de rotação forem limitados a $\tan(\phi) = \pm 2^{-i}$, é possível resolver estas equações, onde i é o índice da interação numérica, utilizando apenas operações de soma e deslocamento. A direção da rotação pode ser tanto no sentido horário quanto no sentido anti-horário, pois $\cos(\phi) = \cos(-\phi)$ e os ângulos quaisquer, dentro de uma certa precisão, podem ser formados por sucessivas rotações cada vez menores. Para que não haja rotações desnecessárias, é necessário somar os ângulos correspondentes às rotações já calculadas. A Equação (5.4) pode ser expressa recursivamente como:

$$\begin{cases} x_{i+1} = k_i [x_i - y_i d_i 2^{-i}] \\ y_{i+1} = k_i [y_i + x_i d_i 2^{-i}] \end{cases} \quad (5.5)$$

em que d_i assume valores do sinal ± 1 (5.6) e k_i é um ganho, conforme 5.7.

$$d_i = \pm 1 \quad (5.6)$$

$$k_i = \cos(\phi) = \cos(\tan^{-1}(2^{-i})) = \frac{1}{\sqrt{1 + 2^{-2i}}} \quad (5.7)$$

Se a variável k_i for removida das equações iterativas acima, pode-se rotacionar vetores somente com operações de soma e de deslocamento. A remoção desta variável, chamada de fator de correção de escala, pode ocorrer através de um produto realizado após cada iteração ou pode ocorrer ao final de todas as iterações, como um processo de ganho do algoritmo. O ganho k_i , introduzido pelas rotações inerentes ao algoritmo, depende do número de iterações; quando o número de iterações tende ao infinito, o produto é aproximadamente 0,6073 [22]. Este ganho pode ser compensado antes ou depois das iterações.

Os ângulos de cada rotação são determinados pelo valor de arcotangentes, que variam de acordo com uma seqüência não decrescente. A Tabela 5.7 exemplifica os passos dos ângulos de rotação fixos para a seqüência dos números naturais.

i	$\tan(\phi_i)$	ϕ_i	$\cos(\phi_i)$
1	1^{-1}	45°	0,7071
2	1^{-2}	$26,5650^\circ$	0,8644
3	1^{-3}	$14,0362^\circ$	0,9701
4	1^{-4}	$7,1250^\circ$	0,9922
5	1^{-5}	$3,5763^\circ$	0,9980
6	1^{-6}	$1,7899^\circ$	0,9995
\vdots	\vdots	\vdots	\vdots

Tabela 5.7: Ângulos fixos para a seqüência dos números naturais.

Uma terceira variável z é necessária para acumular estes ângulos rotacionados, dada pela equação a seguir:

$$z_{i+1} = z_i - d_i \tan^{-1}(2^{-i}) \quad (5.8)$$

em que o sinal de d_i vai depender do sinal de z_i , seguindo a seguinte equação:

$$\begin{cases} d_i = 1, & \text{se } z_i > 0 \\ d_i = -1, & \text{se } z_i < 0 \end{cases} \quad (5.9)$$

em que $z_i = 0$ é o ângulo de rotação final.

Como exemplo, se substituir os valores de $\tan(\phi_i)$ e $\cos(\phi_i)$ na Equação (5.4), e iniciar as variáveis x_0 , y_0 e z_0 com os valores respectivamente: $1/k_i$, 0 e um ângulo qualquer ϕ_i ; tem-se ao final de i iterações o cosseno de ϕ_i armazenado na variável x_i . A cada iteração tem-se a precisão de um *bit*.

Portanto, para a implementação CORDIC, é necessário conhecer o sistema de coordenadas, o modo de operação e a seqüência de deslocamento. Com estes três graus de liberdade, diferentes funções podem ser calculadas a partir de um vetor de entrada escolhido (x_0, y_0, z_0) [3]. No Anexo C, tem a tabela de funções que mostra a permutação das combinações acima e as respectivas funções obtidas em cada modo de operação.

5.3.2 Usando o algoritmo CORDIC em ambiente XSG

No *toolbox* da XSG existe uma classe de algoritmo CORDIC que fornece uma maneira de calcular com precisão funções trigonométricas e outras transcendentais que usam deslocamentos, somas e divisões, e também, mecanismos para calcular a magnitude e o ângulo de fase e também funções recíprocas. O bloco chamado *Cordic Divider* calcula a função divisão através do método do algoritmo CORDIC, e é obtida com o sistema de coordenadas linear, no modo de operação de redução em y . Desta forma, pode-se fazer uma divisão CORDIC das entradas de x com as entradas de y no bloco. A variável z é iniciada com o valor 0, e nela é obtida, ao final de algumas iterações, o valor da divisão de y por x .

Para o cálculo do inverso de um número, usou-se o bloco *Cordic Divider*, vide Figura 5.10, da própria XSG, que faz o cálculo de uma função recíproca. Na entrada de x há um gerador de números aleatórios e na entrada de y um sinal constante de valor 1. Desta maneira, consegue-se calcular o inverso dos dados que estão entrando na entrada x de forma aleatória.

Na Figura 5.11 temos a implementação feita pela XSG do Bloco do CORDIC.

Portanto, dado uma par de entradas x e y , calcula-se a saída y/x . O algoritmo CORDIC está dividido em 4 etapas para fazer essa divisão:

1. Rotação de Coordenadas: O algoritmo CORDIC converge apenas para valores positivos de x . O vetor de entrada é sempre mapeado para o 1º quadrante ou 3º quadrante, tornando as coordenadas x e y não negativas. Portanto, o circuito divisor foi desenvolvido para calcular todos os valores de x e y , exceto para o valores negativos.

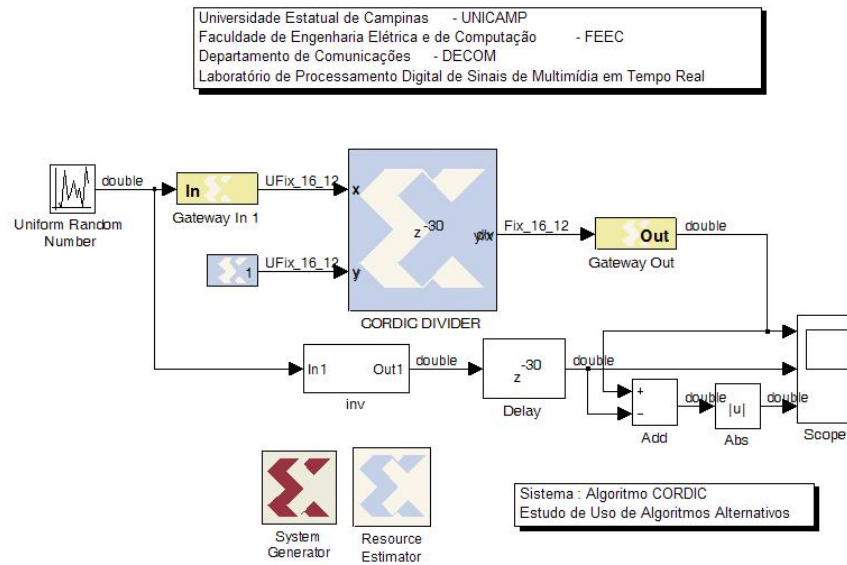


Figura 5.10: Algoritmo do CORDIC em ambiente XSG.

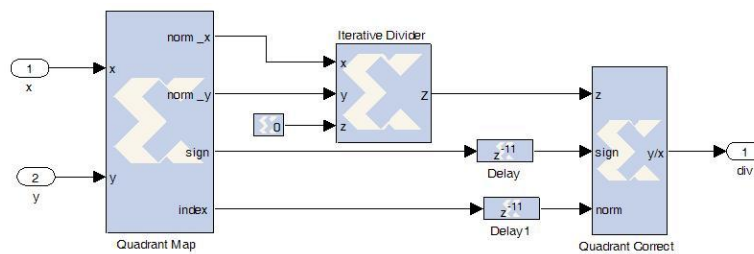


Figura 5.11: Algoritmo interno ao bloco CORDIC em ambiente XSG.

2. Normalização: O algoritmo CORDIC converge apenas para y igual ou inferior a $2x$. Para entradas $x > y$, as entradas x e y são deslocadas para a esquerda até que tenham apenas 1 MSB (bit mais significativo). O deslocamento relativo ao longo de y e x é guardado e transferido para a última etapa de correção.
3. Rotações Lineares: Para o cálculo do inverso de um número qualquer, o vetor resultante $\vec{V}_i = (x_i, y_i)$ é obtido por ângulos progressivos tais que y tenda a 0. Na fase final, temos as interações e o valor da divisão y/x e $z = 0$.

$$\begin{cases} x_{i+1} = x_i - y_i, & \text{se } y_i < 0, & \text{se não } x_{i+1} = x_i + y_i \\ y_{i+1} = y_i + x_i, & \text{se } y_i < 0, & \text{se não } y_{i+1} = y_i - x_i \\ z_{i+1} = z_i - \tan^{-1}(2^{-i}), & \text{se } y_i < 0, & \text{se não } z_{i+1} = z_i + \tan^{-1}(2^{-i}) \end{cases} \quad (5.10)$$

4. Correção das coordenadas: Baseado no eixo de coordenadas e nos deslocamentos aplicados de

y sobre x . Esta etapa atribui o sinal adequado para o quociente resultante e multiplica-o por 2^i , onde i é o relativo ao deslocamento de y sobre x .

5.3.3 Algoritmo de Hung

O algoritmo de Hung é baseado na Série de Taylor para calcular o inverso de um número. Este algoritmo combina os dois primeiros termos desta série, e requer apenas uma tabela para gerar o resultado esperado [18]. A descrição e a implementação deste algoritmo segue abaixo:

Admitindo-se Y um número fixo entre zero e um definido pela Equação (5.11) onde $y_i \in \{0, 1\}$.

$$Y = 2^1 y_0 + 2^0 y_1 + 2^{-1} y_2 + \dots + 2^{-(2m-2)} y_{2m-1} \quad (5.11)$$

Para calcular o inverso, Y é decomposto em duas partes: os *bits* mais significativos Y_h pela Equação (5.12), e os *bits* menos significativos Y_l pela Equação (5.13).

$$Y_h = 2^1 y_0 + 2^0 y_1 + 2^{-1} y_2 + \dots + 2^{-(m-2)} y_{m-1} \quad (5.12)$$

$$Y_l = 2^{-(m-1)} y_m + 2^{-m} y_{m+1} + \dots + 2^{-(2m-2)} y_{2m-1} \quad (5.13)$$

Logo, tem-se a seguinte equação:

$$\frac{1}{Y} = \frac{1}{Y_h + Y_l} = \frac{Y_h - Y_l}{Y_h^2 + Y_l^2} \quad (5.14)$$

A aproximação da Equação (5.14) é equivalente à combinação dos primeiros termos da Série de Taylor.

$$\frac{1}{Y} \approx \frac{Y_h - Y_l}{Y_h^2} \quad (5.15)$$

5.3.4 Usando algoritmo de Hung em ambiente XSG

Implementou-se em XSG, conforme Figura 5.12, a Equação (5.15). O sinal de entrada varia aleatoriamente entre um e dois, e assim, este valor é separado em uma parte alta, Y_h , e uma parte baixa, Y_l , com ajuda dos blocos *slice*. O resultado obtido da multiplicação da subtração $Y_h - Y_l$ e os valores obtido da LUT é o valor aproximado do inverso do dado de entrada. Os valores de Y_h^2 são o índice de entrada da LUT, sendo que nesta LUT encontram-se os 2^8 valores esperados da divisão de $1/Y_h^2$.

Na próxima subseção, faz-se uma comparação entre os dois algoritmos em relação aos recursos utilizados em FPGA da família Virtex-4 e a precisão dos algoritmos.

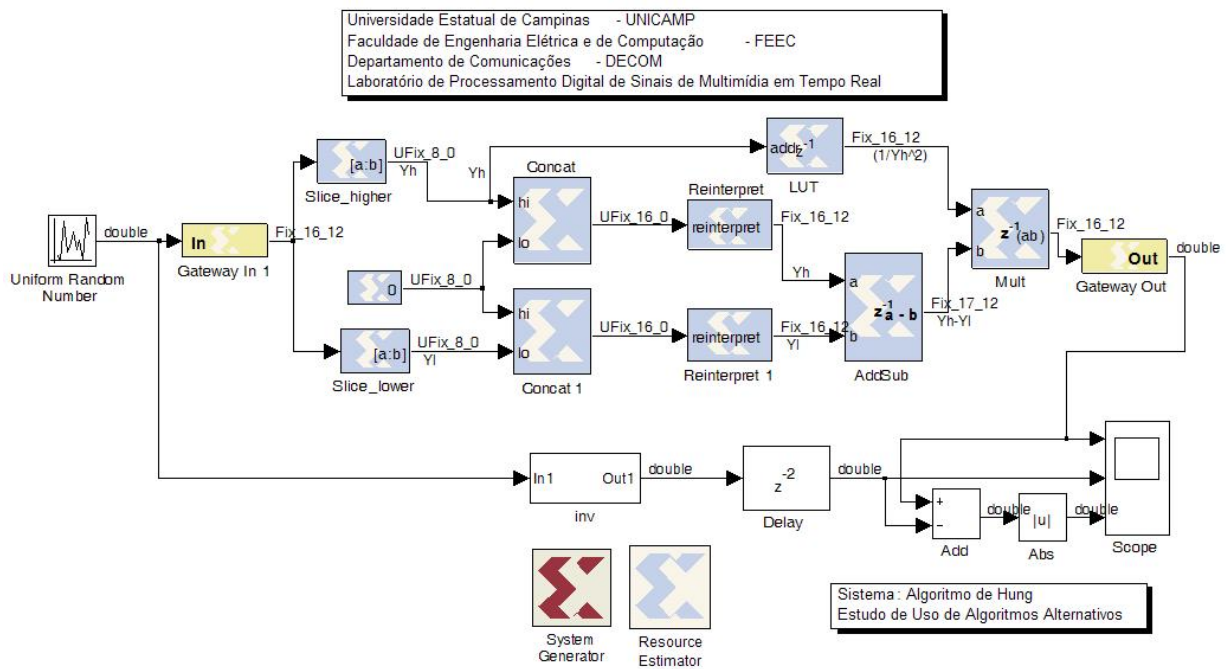


Figura 5.12: Algoritmo de Hung em ambiente XSG.

5.3.5 Comparação entre os Algoritmos

Na Figura 5.13 apresenta-se a simulação, em ambiente *Simulink*, comparativa entre os dois algoritmos apresentado nesta técnica. A faixa dinâmica e a seqüência de *bits* são iguais.

Na Tabela 5.8 tem-se a comparação dos recursos utilizados do algoritmo de Hung e do bloco *CORDIC*. Observa-se apenas a quantidade de *slices* utilizados em ambas as implementações. Conclui-se que a implementação de Hung ocupa aproximadamente 9 vezes menos que a implementação do Bloco *CORDIC* da XSG em FPGA do fabricante Xilinx família Virtex-4 FPGA XC4V5X35.

Recurso	Cordic	Hung	Redução(%)
Slice	493	50	89,85
FFs	806	50	93,79
BRAMs	0	1	-100
LUTs	772	62	91,96
IOBs	32	32	0
Emb. Mults	1	1	0
Atraso	2	30	-
Erro Abs. Médio	$2.4767e^{-7}$	$3.5814e^{-7}$	-
Variância	$3.739e^{-4}$	$9.235e^{-4}$	-

Tabela 5.8: Comparação de recursos Algoritmo de Hung e Bloco CORDIC.

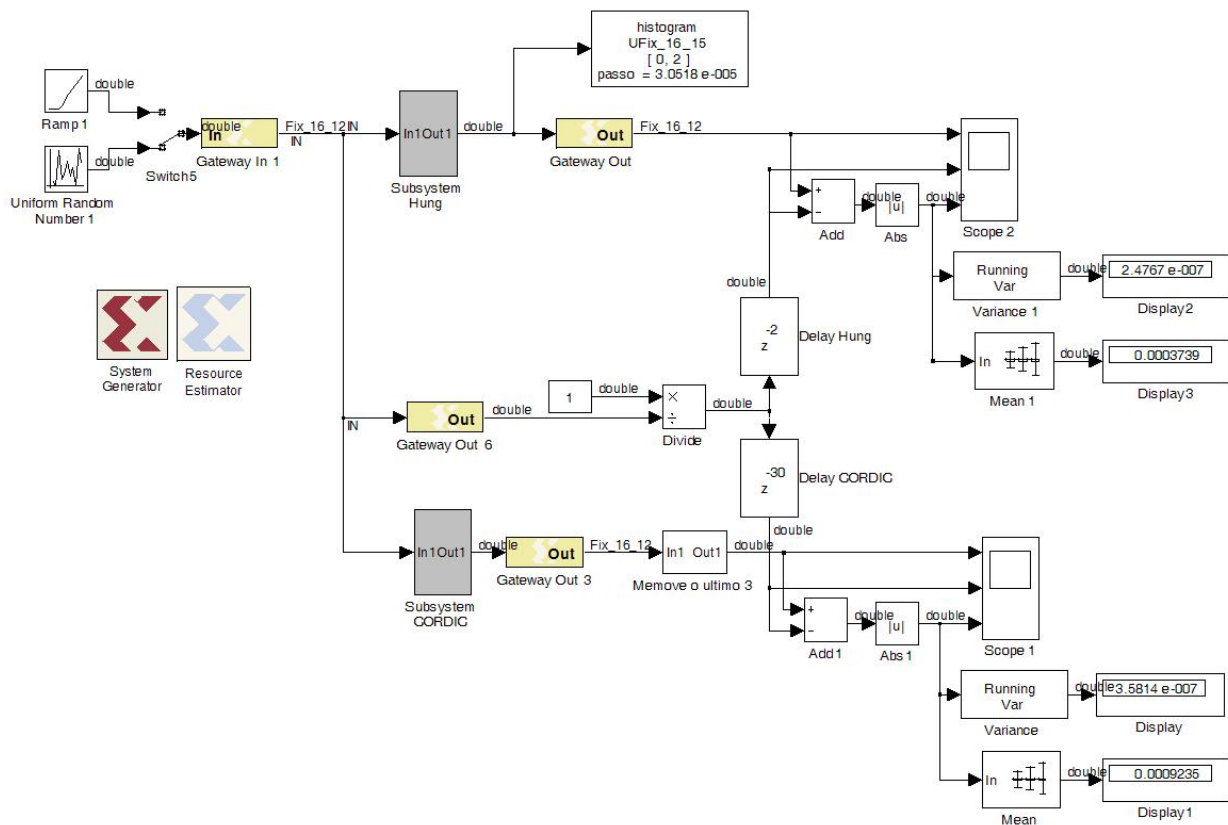


Figura 5.13: Comparação entre Algoritmos de Hung e CORDIC.

Conforme visto na Tabela 5.8, a precisão do cálculo de uma função inversa do algoritmo de Hung é mais precisa do que o código do bloco *CORDIC* desenvolvido pela Xilinx. Além de uma redução de área de silício, tem-se uma precisão melhor ao implementar o algoritmo de Hung.

5.4 Uso de Recursos DSP48

Este tópico poderia ser tratado na seção 5.1 sobre o uso de recursos alternativos. Porém dada a importância deste bloco em processamento digital de sinais, optou-se por criar uma subseção a parte.

5.4.1 Filtro FIR

O filtro digital FIR tem uma resposta impulsiva finita. Na Figura 5.14 pode-se observar a estrutura interna de um filtro digital FIR, composta por registradores, somadores e multiplicadores, os quais são responsáveis pela operação de convolução.

Na figura os blocos z^{-1} são os elementos de atraso e os termos h são os coeficientes.

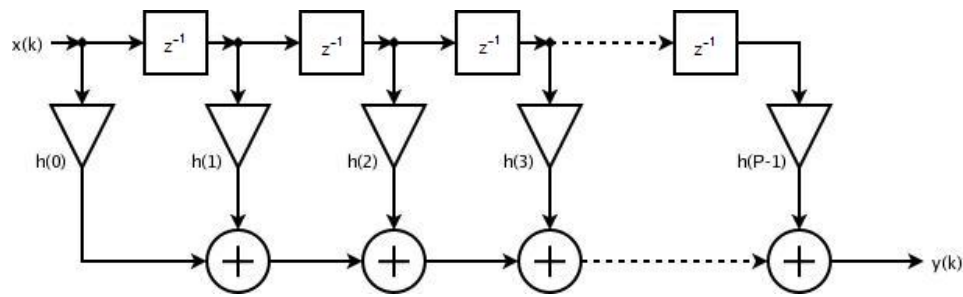


Figura 5.14: Estrutura de um filtro digital FIR.

A representação de um filtro FIR de duração N é dada pela Equação (5.16) da seguinte forma:

$$y_k = h_0x_k + h_1x_{k-1} + h_2x_{k-2} + \dots + h_Nx_{k-N}. \quad (5.16)$$

A Equação (5.16) também pode ser expressa por:

$$y_k = \sum_{n=0}^{N-1} h_nx_{k-n}, \quad (5.17)$$

onde x_k representa o sinal de entrada, h_n são os coeficientes do filtro, y_k é o sinal de saída filtrado e k o índice temporal. Nessa equação, os coeficientes do filtro são equivalentes à sua resposta à amostra unitária devido ao fato de que os valores de saída do passado não influenciam no cálculo dos valores de saída presentes. Desta forma, a implementação de filtros digitais FIR, em XSG, é simples, uma vez que um filtro digital é composto basicamente de atrasos, acumuladores e multiplicadores.

A Figura 5.15 mostra a implementação do algoritmo de um filtro FIR em ambiente XSG com 16 coeficientes.

Essa implementação utiliza aproximadamente 965 *slices* de lógica em FPGA família Virtex-4 FPGA XC4VSX35, conforme Tabela 5.9. Este resultado foi fornecido pela ferramenta *Resource Estimator* do XSG.

<i>Recurso</i>	<i>Quantidade</i>
<i>Slice</i>	965
FFs	256
BRAMs	0
LUTs	1614
IOBs	52
Emb. Mults	0
TBUFs	0

Tabela 5.9: Estimativa de recursos para implementação de um filtro FIR.

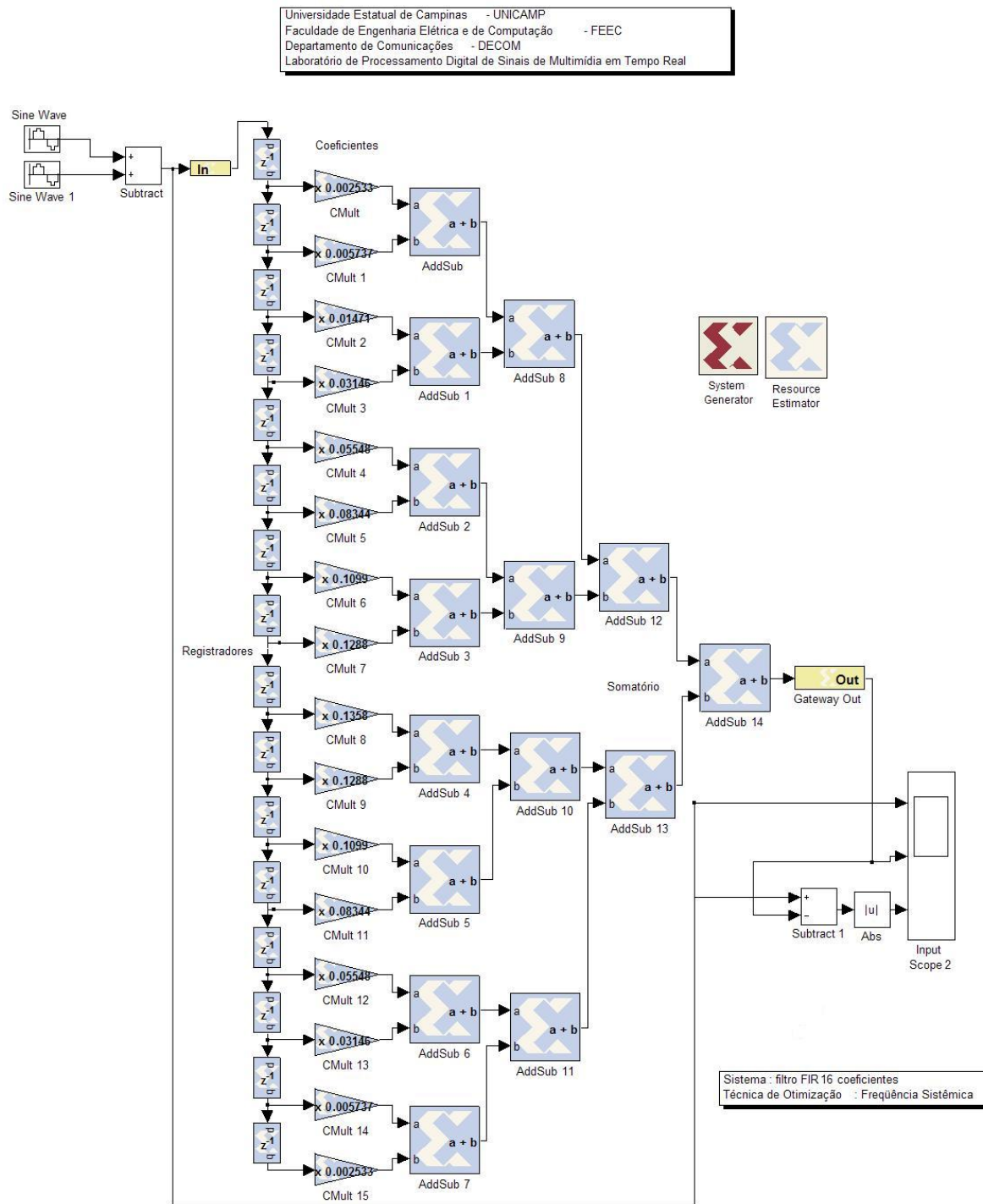


Figura 5.15: Algoritmo do filtro FIR em ambiente XSG.

5.4.2 Filtro FIR utilizando ASR

Nesta seção, busca-se implementar de forma eficiente um filtro digital FIR utilizando ASR (do inglês, *Addressable Shift Register*), a partir de um conjunto de operações do tipo MAC (*Multiplica-*

ACumula) que são executadas para cada amostra do sinal de entrada, multiplicando as N amostras da entrada pelos coeficientes fixos do filtro, que estão no bloco de memória ROM. Pode-se implementar este tipo de filtro conforme mostra a Figura 5.16 com blocos da XSG.

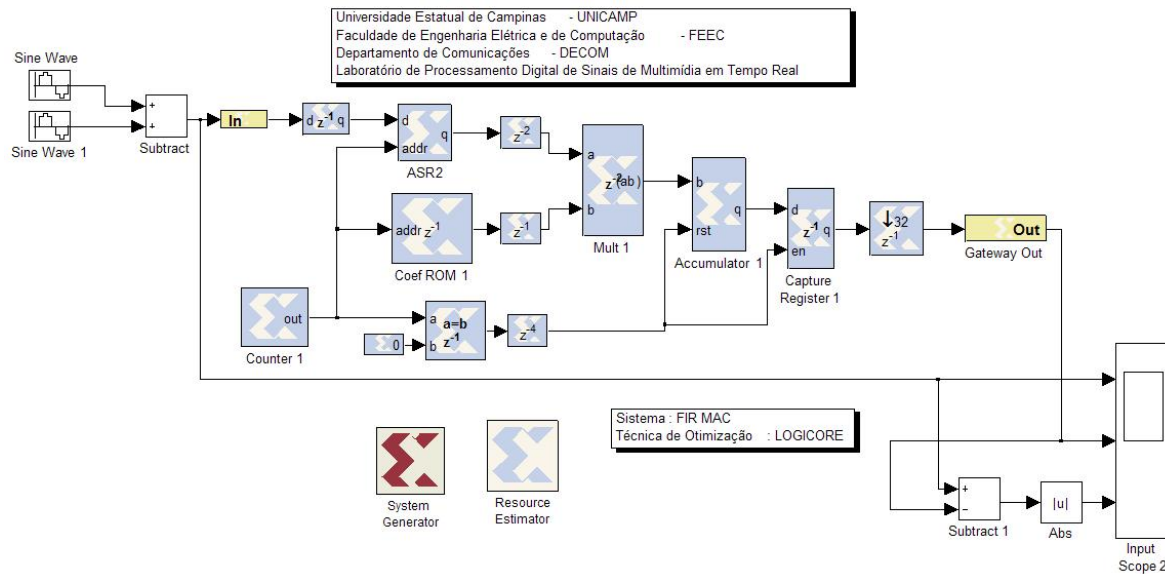


Figura 5.16: Algoritmo do FIR utilizando ASR em ambiente XSG.

O bloco *Counter* é configurado para contar de 1 até N , onde o N é o número de coeficientes do filtro, e os dados que saem desse bloco são o endereço de entrada da ROM e do bloco *ASR*. Este bloco *Counter* possui uma frequência de operação N vezes maior que as amostras de entrada que são escritas no bloco *ASR*. Os coeficientes do filtro são armazenados no bloco de ROM. O multiplicador seguido do acumulador soma os produtos durante o mesmo *clock* do bloco *Counter*. Com esta relação, o desempenho do filtro MAC FIR é calculado pela seguinte Equação (5.18):

$$f_{\max} = f_{\text{clk}}/N, \quad (5.18)$$

onde f_{\max} é a frequência máxima do sinal de entrada do filtro, f_{clk} é a frequência de *clock* da FPGA e N é a quantidade de coeficientes do filtro.

Nesta implementação, utilizou-se aproximadamente 215 *slices* de uma FPGA do fabricante Xilinx da família Virtex-4 FPGA XC4VSX35, e outros recursos utilizados estão na Tabela 5.10. Este resultado foi fornecido pela ferramenta *Resource Estimator* do XSG.

Comparando-se a tabela do filtro FIR exemplo com a tabela do filtro FIR utilizando ASR, implementados em XSG, a quantidade de *slice* teve uma redução de 965 para 215. O aumento de uma memória BRAMs na tabela 5.10 se dá pela utilização do bloco de multiplicação.

<i>Recurso</i>	<i>Quantidade</i>
<i>Slice</i>	215
FFs	217
BRAMs	1
LUTs	308
IOBs	44
Emb. Mults	0
TBUFs	0

Tabela 5.10: Estimativa de recursos de um filtro FIR ASR de 16 coeficientes.

5.4.3 Filtro FIR utilizando ASR com recurso DSP48 da família Virtex-4

Os filtros FIR são normalmente utilizados em aplicações DSP. Com a introdução dos recursos de DSP48 na família da Virtex-4, os algoritmos de filtros podem alcançar uma área menor, enquanto, ao mesmo tempo, produzindo maior desempenho com menor área de de silício. Com isso, o projetista tem uma certa flexibilidade no desenvolvimento de suas aplicações, bem como a capacidade de desenvolver sistemas mais eficientes. Nesta seção, busca-se implementar o mesmo filtro digital FIR, da subseção anterior, que utiliza ASR e um conjunto de operações do tipo MAC, entretanto no bloco de multiplicação é forçado para que a ferramenta de síntese utilize o recurso DSP48 presente nas famílias Virtex-4.

Nesta implementação são utilizadas aproximadamente 106 *slices* de uma FPGA do fabricante Xilinx família Virtex-4 FPGA XC4VSX35, e outros recursos utilizados estão na Tabela 5.11, este resultado foi fornecido pela ferramenta *Resource Estimator* do XSG.

<i>Recurso</i>	<i>Quantidade</i>
<i>Slice</i>	106
FFs	155
BRAMs	1
LUTs	123
IOBs	44
Emb. Mults	1
TBUFs	0

Tabela 5.11: Estimativa de recursos de um filtro FIR com DSP48 de 16 coeficientes.

Comparando-se a tabela do filtro FIR utilizando ASR com a tabela do filtro FIR que utiliza recursos de DSP da Virtex-4, implementados em XSG, a quantidade de *slice* teve uma redução de 215 para 106.

5.4.4 Comparação entre as implementações

Na tabela 5.12, tem-se a comparação dos recursos utilizados dos algoritmos FIRs. Observa-se apenas a quantidade de *slices* utilizados em ambas as implementações. Conclui-se que o primeiro filtro FIR ocupa aproximadamente 4,49 vezes menos que a segunda implementação do filtro FIR com ASR, e 9,10 vezes menos que a terceira implementação do filtro FIR com ASR com DSP48.

Recurso dos Filtros	Sem ASR	Com ASR	Com ASR e DSP48
<i>Slice</i>	965	215	106
FFs	256	217	155
BRAMs	0	1	1
LUTs	1614	308	123
IOBs	52	44	44
Emb. Mults	0	0	1
TBUFs	0	0	0

Tabela 5.12: Comparação dos dados obtidos pelos diversos algoritmos FIR.

5.5 Considerações sobre Frequência Sistêmica

Nesta seção são discutidas algumas implementações de filtros FIR (Resposta Finita ao Impulso), a título de exemplo, para fazer algumas considerações sobre frequência sistêmica em uma FPGA. Para isto, é empregada a implementação de filtros digitais com resposta ao impulso de duração finita, em FPGA, pois são de grande importância em processamento digital de sinais. Uma vez que um filtro digital é composto basicamente de atrasos, somadores e multiplicadores, sendo este último complexo e que consome mais potência, o uso de estruturas alternativas permite uma melhor otimização dos filtros em FPGAs.

Utiliza-se ferramenta de prototipagem rápida do XSG para implementar e calcular os recursos em FPGA, e os principais blocos utilizados foram os registradores lógicos de deslocamento, multiplicadores embarcados, blocos lógicos e de memória de uma FPGA da família Virtex-4 para implementar diferentes filtros FIR, e dessa forma, fazer a comparação de frequência sistêmica e desempenho das diferentes estruturas dos filtros.

Tomando-se como exemplo a implementação do filtro FIR da Figura 5.15, para uma frequência máxima de 500 MHz, a máxima frequência de amostragem do sinal é de 31,25MHz. Colocando-se de outra maneira, tal estrutura tem como limite um filtro máximo de ordem 125 para um sinal de entrada de 4MHz. Uma forma de aumenar este fator é utilizar a dimensão espacial, ou seja, maior área de silício, como mostrado a seguir.

Um único filtro FIR tem o inconveniente de que a frequência de *clock* de operação de cálculo é inversamente proporcional ao número de coeficientes do filtro. Como os coeficientes possuem uma forma simétrica, pode-se escrever a Equação (5.19) do filtro em parcelas da seguinte forma:

$$y_k = \sum_{n=0}^{\frac{N}{2}-1} h_n x_{k-n} + \sum_{n=\frac{N}{2}}^{N-1} h_n x_{k-n}, \quad (5.19)$$

neste caso, na implementação, a frequência do bloco *Counter* é multiplicada por k para fazer o cálculo de toda a entrada do filtro, e dessa forma, o filtro pode ser totalmente paralelizado utilizando os blocos do XSG.

A Equação (5.20), obtida a partir da Equação (5.18), mostra a frequência de operação do contador de um filtro MAC FIR de k parcelas:

$$f_s = f_{\text{clk}} k / N, \quad (5.20)$$

onde f_s é a frequência de amostragem do sinal de entrada do filtro, f_{clk} é a frequência de *clock* da FPGA, k é o número de parcelas do filtro e N é a ordem do filtro FIR.

Na Figura 5.17, utiliza-se quatro parcelas para filtrar um determinado sinal. A concepção desse filtro é o fator multiplicativo, pois, a natureza simétrica dos coeficientes do filtro MAC FIR quadruplica a taxa de amostragem de operação do filtro, e assim, para uma mesma taxa de entrada de dados, é capaz de atingir a frequência máxima de *clock* de uma FPGA, ou frequência sistêmica. Outra técnica é aumentar o número de multiplicadores utilizados para paralelizar o fluxo de dados de um filtro FIR. Este paralelismo introduz mais um recurso de DSP48 em área de FPGA.

Portanto, a Figura 5.17 ilustra um caso de aumento da frequência do filtro MAC FIR que pode ser implementado usando quatro parcelas. Esta figura mostra a acumulação dos coeficientes de cada parcela MAC e estes resultados parciais devem ser combinados entre si e em seguida somados para se obter o resultado final. Esse processo utiliza um atraso extra em relação à Figura 5.16.

A implementação da Figura 5.17 utiliza aproximadamente 211 *slices* de uma FPGA do fabricante Xilinx família Virtex-4 FPGA XC4VSX35, Tabela 5.13.

A Figura 5.18 ilustra um caso de aumento da frequência do filtro MAC FIR que pode ser implementado usando oito parcelas. Esta figura mostra a acumulação dos coeficientes de cada parcela MAC e estes resultados parciais devem ser combinados entre si e em seguida somados para obter o resultado final. Esse processo utiliza um atraso extra em relação à Figura 5.17.

Essa implementação da Figura 5.18 utiliza aproximadamente 309 *slices* de uma FPGA do fabricante Xilinx família Virtex-4 FPGA XC4VSX35, Tabela 5.14, e este resultado foi fornecido pela ferramenta *Resource Estimator* do XSG.

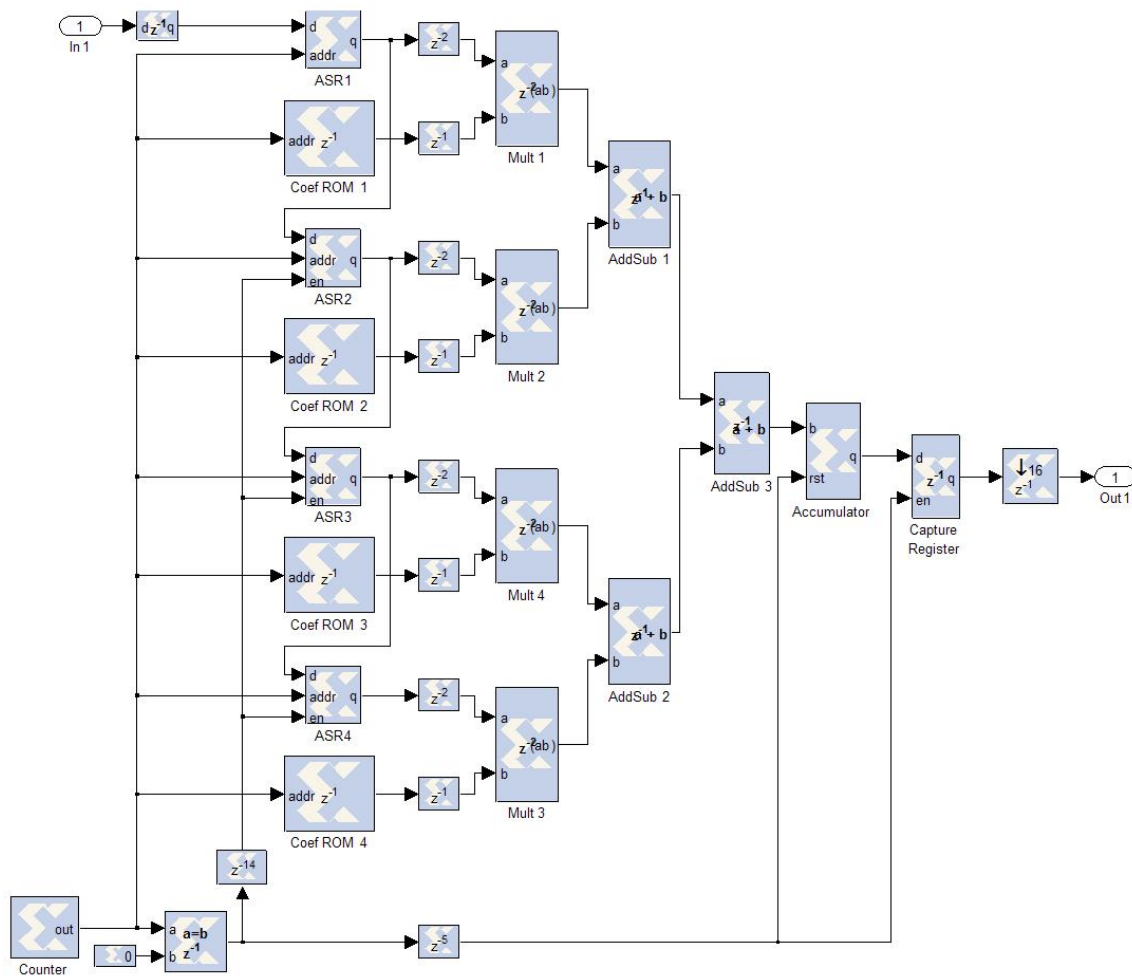


Figura 5.17: Algoritmo do MAC FIR - 4 parcelas em ambiente XSG.

Recurso	Quantidade
Slice	211
FFs	281
BRAMs	4
LUTs	294
IOBs	37
Emb. Mults	4
TBUFs	0

Tabela 5.13: Estimativa de recursos de um filtro MAC FIR de 4 parcelas e 32 coeficientes.

Todos os resultados foram obtidos utilizando o *software* Xilinx ISE 9.2 Series para calcular a frequência máxima sistêmica em uma FPGA do fabricante Xilinx família Virtex-4 FPGA XC4VSX35 de 500 MHz. Todos os modelos utilizam multiplicadores dedicados, bloco de memória que contém

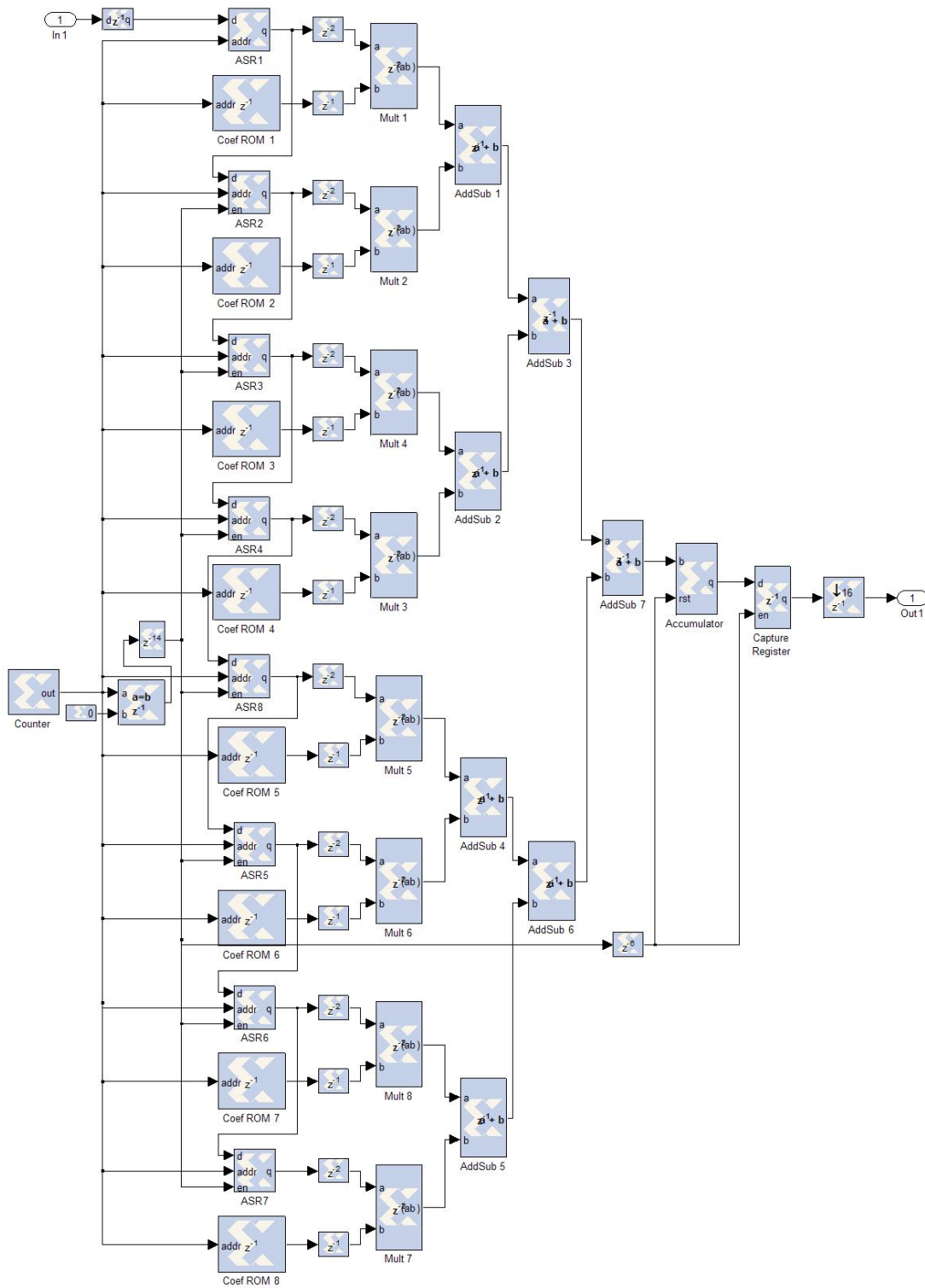


Figura 5.18: Algoritmo do MAC FIR - 8 parcelas em ambiente XSG.

<i>Recurso</i>	<i>Quantidade</i>
<i>Slice</i>	309
FFs	385
BRAMs	8
LUTs	394
IOBs	37
Emb. Mults	8
TBUFs	0

Tabela 5.14: Estimativa de recursos de um filtro MAC FIR de 8 parcelas e 32 coeficientes.

os coeficientes do filtro, e 18 *bits* de precisão do filtro de entrada e dos coeficientes.

O número de coeficientes pela frequência de amostragem de filtros FIR está na Figura 5.19. Quanto mais coeficientes, maior é a aproximação da frequência sistêmica do circuito em relação à frequência máxima de operação da FPGA.

A capacidade de ajuste de um filtro em um sistema existente ou para ter múltiplas configurações de filtros é uma vantagem clara. Com o XSG, os filtros são facilmente modificados para atingir determinados requisitos, tais como diferentes coeficientes, faixa dinâmica de entrada e dos coeficientes. E além disso, a respeito do consumo de energia de um dispositivo FPGA, as frequências mais baixas implicam em menor consumo de potência, requisito de suma importância para dispositivos embarcados.

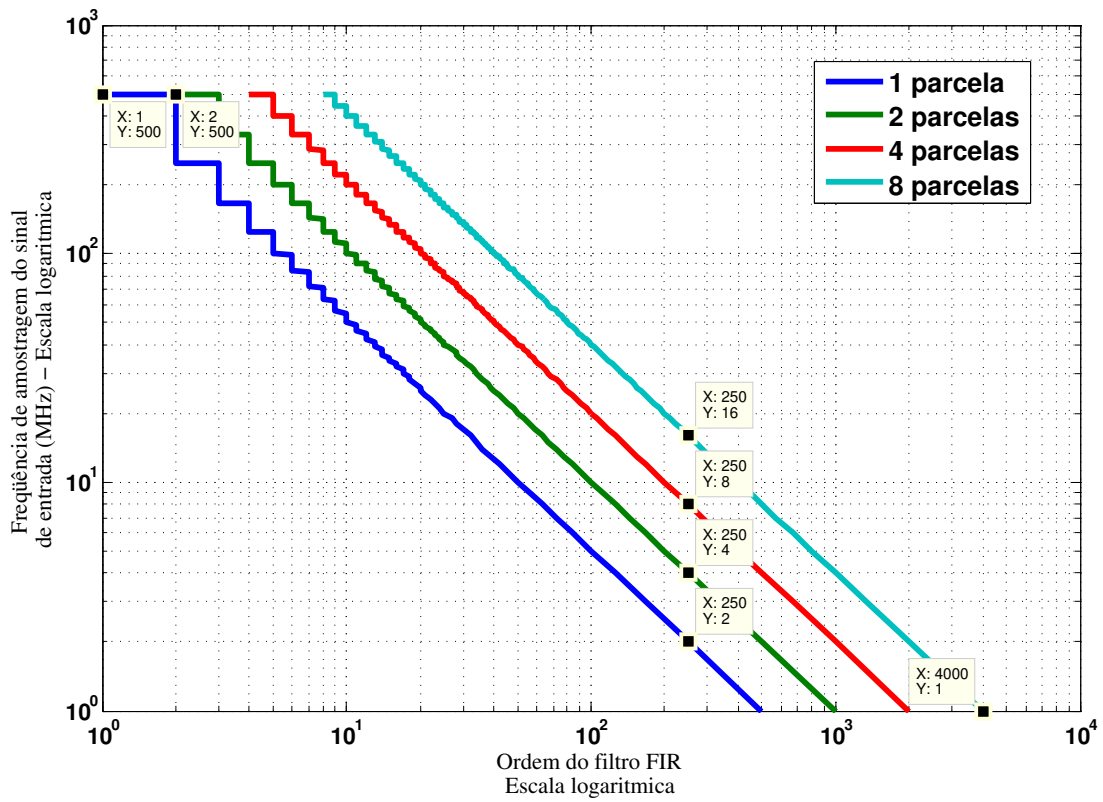


Figura 5.19: Relação de ordem do filtro FIR e a frequência de amostragem do sinal de entrada.

Capítulo 6

Conclusão

Nesta dissertação, procurou-se compilar algumas técnicas de otimização de códigos de processamento digital de sinais em FPGA. Estas técnicas foram utilizadas durante as pesquisas para o projeto de um sistema de comunicação sem fio do WiMAX no laboratório de RT-DSP na UNICAMP, a partir da implementação de um *transceiver* com ajuda de um *kit* de desenvolvimendo da *Nallatech* contendo uma FPGA Xilinx da família 4. Hoje em dia, há uma tendência de se empregar ambientes gráficos para a geração de códigos de processamento digital de sinais, permitindo simular e gerar códigos para serem implementados em DSPs e FPGAs.

Foram implementados e avaliados os algoritmos de modulação por amplitude em quadratura, transformada rápida de Fourier, filtros de resposta ao impulso finita, algoritmo CORDIC e o algoritmo de Hung. Na avaliação dos algoritmos, implementações alternativas foram realizadas para comparar seus resultados. Para isso, a ferramenta de prototipagem rápida e estimativa de recursos foram de extrema importância para a verificação dessa redução em área de FPGA.

A primeira técnica de otimização em FPGA foi um estudo da utilização de recursos alternativos, ou seja, a forma do projetista ao implementar o algoritmo influência sobremaneira na ocupação final dos recursos de uma FPGA. Em alguns casos, pode-se descrever diferentes formas para implementar o mesmo algoritmo com a finalidade de ocupar diferentes recursos de um *chip* reprogramável. Para a demonstração desta técnica, utilizou-se o algoritmo de modulação por amplitude em quadratura de 16 pontos.

A segunda técnica descrita neste trabalho é a otimização por multiplexação no tempo, esta técnica aumenta a frequência de uma determinada região em silício para trabalhar mais rápido, e assim, conseguimos uma redução de 44,75% do *chip* de uma FPGA da Virtex-4 da Xilinx para o exemplo da modulação OFDM com FFT de 256 pontos.

A terceira técnica faz uma comparação algorítmica de algoritmos que fazem o cálculo do inverso de um número. Implementou-se o algoritmo de Hung e comparou-se com um outro algo-

ritmo desenvolvido pela ferramenta de prototipagem rápida de propriedade intelectual da Xilinx, chamado algoritmo CORDIC. O algoritmo de Hung conseguiu uma redução de aproximadamente 90% comparando-se com o algoritmo desenvolvido pela ferramenta. Conclui-se que em determinadas situações, pode-se buscar otimizar algoritmos que ocupem menos recursos em FPGA que os algoritmos desenvolvidos pelas ferramentas.

A última técnica descrita se refere às considerações sobre frequência sistêmica para modificar a frequência de operação de um *chip* FPGA e o uso de determinados recursos existentes na tecnologia da FPGA para ter um melhor desempenho e reduzir a área de ocupação. Os algoritmos utilizados são todos filtros de resposta finita com 16 coeficientes para a demonstração desta técnica.

A qualidade dos resultados obtidos com as estimativas de recursos depende em grande parte das respostas fornecidas pela ferramenta de estimativa de recursos da XSG durante o desenvolvimento do algoritmo. Através de análise dos resultados de várias simulações, foi possível estabelecer maneiras para a escolha dos melhores resultados.

Embora as técnicas tenham sido implementadas separadamente nesta dissertação, os resultados obtidos sugerem a conveniência de combinar as técnicas num só projeto, de tal forma que a redução de recursos otimizados em uma FPGA seja mais eficiente. Com isso, as relações entre os algoritmos desenvolvidos e área estaria sendo explorada de maneira mais eficiente para um determinado sistema.

Essas técnicas foram empregadas com sucesso pela equipe do RT-DSP no desenvolvimento dos algoritmos para implementação do padrão IEEE 802.16d-WiMAX em *hardware* com ajuda do ferramenta de prototipagem rápida XSG. Com o emprego dessas técnicas, houve uma redução significativa em área de ocupação de uma FPGA. As técnicas descritas nesta dissertação não se encontram compiladas em um documento único e são fruto do trabalho da equipe, fato este que enfatiza a importância deste documento.

Referências Bibliográficas

- [1] Jim Simkins, Ben White. *Design FPGA-Based DSPs for Performance and Power*. Technical report, *Chip Design Magazine*, Xilinx, Inc., Março 2005.
- [2] Douang Phanthavong. *Designing with DSP48 Blocks Using Precision Synthesis: Achieve close to custom silicon performance in FPGA DSP designs*. Third Quarter 2005. Disponível em: http://www.xilinx.com/publications/xcellonline/xcell_54/xcell_54_dsp48-54.pdf.
- [3] Bruno de Carvalho Costa. Implementação de filtros de Wiener composto e precisão reduzidos. Tese de mestrado, Universidade Federal do Rio de Janeiro - UFRJ, Março 2006.
- [4] Nahri Balesdent Moreano. Algoritmos para Alocação de Recursos em Arquiteturas Reconfiguráveis. Tese de doutorado, Instituto de Computação, UNICAMP, Novembro 2005.
- [5] Uwe Meyer-Baese. *Signal Processing with Field Programmable Gate Arrays: Signals and Communication Technology*, volume 1, chapter *Introduction*, pages 1–27. 2004.
- [6] P. K. Chan, S. Mourad. *Digital Signal Processing with Field Programmable Gates Array*, volume 1. 1994.
- [7] Richard Munden. *Asic and Fpga Verification - A Guide to Component Modeling*, volume 1. 2005.
- [8] F. Vahid; T. Givargis. *Embedded System Design*, volume 1. 1999.
- [9] David A. Patterson, John L. Hennessy. *Computer Organization and Design*, volume 1. 2004.
- [10] Patrick Lysaght. *Future Design Tools for Platform FPGAs*, volume 1. 2005.
- [11] Alexandre Alves de Lima Ribeiro. Reconfigurabilidade dinâmica e remota de FPGAs. Tese de mestrado, Instituto de Computação, UNICAMP, Novembro 2005.

- [12] Xilinx Inc. *Xilinx System Generator*. 05 de Abril 2005. Disponível em: <http://www.xilinx.com>.
- [13] A. Toledo, C. Vicente-Chicote, J. Suardiaz, S. Cuenca. *Xilinx System Generator Based HW Components for Rapid Prototyping of Computer Vision SW/HW Systems*, pages pp. 667–674. 2005.
- [14] C. Dick, F. Harris, M. Rice. *Implementation of Carrier Synchronization for QAM Receivers*, volume 1, page 5771. 2004.
- [15] M. Licko, J. Schier, M. Tichy, M. Kuhl. *MATLAB/Simulink Based Methodology for Rapid-FPGA-Prototyping*, volume 1, pages 984–987. 2003.
- [16] Z. Pohl, J. Schier, M. Licko, A. Hermanek, M. Tichy, R. Matousek, J. Kadlec. *Logarithmic Arithmetic for Real Data Types and Support for Matlab/Simulink Based Rapid-FPGAPrototyping*, volume 1. 2003.
- [17] Fábio L. Garcia. Implementação de um Codificador LDPC para um Sistema de TV Digital usando Ferramentas de Prototipagem Rápida. Tese de mestrado, Instituto de Computação, UNICAMP, Dezembro 2006.
- [18] P. Hung, H. Fahmy, O. Mencer, M. J. Flynn. *Fast Division Algorithm with Small Lookup Table*, volume 2, pages 1465–1468. 1999.
- [19] Luís Geraldo Pedroso Meloni. *A New WiMAX Profile for DTV Return Channel and Wireless Access*. Technical report, cap. livro Mobile WiMAX, John Wiley e Sons, 2008.
- [20] Jack E. Volder. *The CORDIC Trigonometric Computing Technique*, volume EC-8, pages 330–334. IRE Trans. Electronic Computing, Setembro, 1959.
- [21] H. M. Ahmed, J. M. Delosme e M. Morf. *Hight Concurrent Computing Structure for Matrix Arithmetic and Signal Processing*, volume 15, pages 65–82. 1982.
- [22] Ray Andraka. *A survey of CORDIC algorithms for FPGA based computers*. Technical report, Andraka Consulting Group, Inc, 1998.

Apêndice A

Aritmética do Ponto-Fixo no XSG

A matemática de ponto-flutuante é poderosa, permitindo que se obtenha um alto grau de precisão, mas ao mesmo tempo demanda muitos recursos. Para contornar esta questão, pode-se utilizar a matemática de ponto-fixado, que é mais rápida, e, apesar de não ser tão precisa, é suficiente para implementar alguns algoritmos de DSP em FPGA. A representação dos sinais no XSG é uma representação *booleana* ou em ponto-fixado com ponto binário, com ou sem sinal.

Como exemplo, as configurações do bloco *Convert* na Figura A.1 está em 17 *bits* e ponto binário em 15 do tipo complemento de 2.

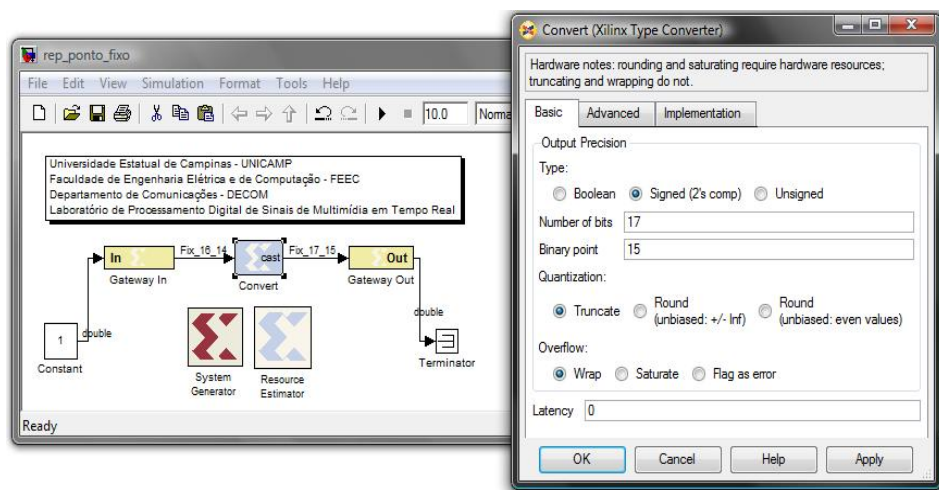


Figura A.1: Representação em ponto-fixado no XSG.

Este número em representação de ponto-fixado é guardado no computador da mesma forma que um inteiro. Ao se programar, assume-se que exista um ponto decimal imaginário num local pré determinado no meio do número. Ao nos referirmos a um número em ponto-fixado, precisamos especificar quantos *bits* este possui na sua parte inteira e na fracionária. Por exemplo, o número 17.15 represen-

taria um número com 2 dígitos na parte inteira e 15 na parte fracionária.

A.1 Representação com Sinal na Complemento de 2

Na representação com sinal é utilizada a notação complemento de dois. Nesta notação, o número negativo é obtido invertendo-se os *bits* do número positivo e em seguida somando-se um. Este procedimento é mostrado na Tabela A.1 para representações com três *bits*.

Positivo	Rep. Binária	Barrado	Rep. Binária	Negativos	Rep. Binária
0	000	$\bar{0}$	111	0	000
1	001	$\bar{1}$	110	-1	111
2	010	$\bar{2}$	101	-2	110
3	011	$\bar{3}$	100	-3	101
4	100	$\bar{4}$	011	-4	100

Tabela A.1: Obtenção do número negativo a partir do positivo na representação complemento de 2 com três *bits*.

Observe que a soma de um número positivo pelo seu complemento de dois vai dar sempre zero se for descartado o que excede o comprimento em *bits* da palavra. Por exemplo, $(1_d) + (-1_d) = (001_{b3}) + (110_{b3}) = (1)000_{b4}$.

A partir do exemplo da Tabela A.1, na qual os limites dos números 4_d e -4_d tem a mesma representação, resultando que apenas um deles deve ser mantido na faixa de valores possíveis. Neste caso, o número 100 é incluído como -4 porque o *bit* mais significativo é 1. Desta forma, note que o *bit* mais significativo pode ser utilizado para indicar o sinal do valor (“0” para positivo e “1” para negativo).

A.2 Fator de Escala em Potência de 2

Na representação em ponto-fixa do XGS, utiliza-se também um fator de escala em potência de dois, chamado *Pb* (Ponto Binário), que possibilita a representação de números reais com precisão finita. Neste caso, vale a seguinte equação A.1:

$$V_r = 2^{-Pb_i} \quad (\text{A.1})$$

onde V_r é o valor real e i é o inteiro.

O XSG define dois tipos básicos $Ufix_Nb_Pb$ e Fix_Nb_Pb , onde Nb é o número de *bits* e Pb é o ponto binário. $Ufix$ representa um tipo sem sinal, enquanto Fix representa um tipo com sinal.

- Para o tipo $Ufix_Nb_Pb$ a faixa de valores possíveis é:

$$[0, 2^{-Pb}(2^{Nb} - 1)], \quad (A.2)$$

com passo 2^{-Pb} .

Por exemplo:

O $Ufix_4_0$ representa os valores $[0, 1, 2, \dots, 15]$ enquanto $Ufix_4_2$ representa os valores $[0, 0.25, 0.5, \dots, 3.75]$.

- Para o tipo Fix_Nb_Pb a faixa de valores é:

$$[-2^{-Pb}2^{Nb-1}, 2^{-Pb}(2^{Nb-1} - 1)] \quad (A.3)$$

com passo 2^{-Pb} .

Por exemplo:

O Fix_4_0 representa os valores $[-8, -7, -6, \dots, 0, 1, 2, \dots, 7]$ enquanto Fix_4_2 representa os valores $[-2, -1.75, -1.5, \dots, 0, 0.25, 0.5, \dots, 1.75]$. Observe que o número de *bits* Nb aumenta a faixa de valores da representação numérica, enquanto que o fator de escala Pb aumenta a precisão dessa mesma representação.

Uma vantagem da notação complemento de 2 está na possibilidade da utilização do mesmo circuito de adição para se implementar a subtração.

Apêndice B

Funções do Matlab utilizado no MCode do XSG

B.1 Função de Modulação

O código fonte do Bloco *MCode* do XSG que implementa a parte de modulação do QAM16.

```
function [Im, Re]= qam16_tx(bits)

width = 16;
binpt = 13;
proto = {xlSigned, width, binpt};

Im = xfix(proto,0);
Re = xfix(proto,0);

switch bits
    case 0
        Im = 1;
        Re = 1;
    case 1
        Im = 1;
        Re = 3;
    case 2
        Im = 1;
        Re = -1;
    case 3
        Im = 1;
        Re = -3;
```

```
case 4
    Im = 3;
    Re = 1;
case 5
    Im = 3;
    Re = 3;
case 6
    Im = 3;
    Re = -1;
case 7
    Im = 3;
    Re = -3;
case 8
    Im = -1;
    Re = 1;
case 9
    Im = -1;
    Re = 3;
case 10
    Im = -1;
    Re = -1;
case 11
    Im = -1;
    Re = -3;
case 12
    Im = -3;
    Re = 1;
case 13
    Im = -3;
    Re = 3;
case 14
    Im = -3;
    Re = -1;
case 15
    Im = -3;
    Re = -3;
otherwise
    Im = 0;
    Re = 0;
end
```

B.2 Função de Demodulação

O código fonte do Bloco *MCode* do XSG que implementa a parte de demodulação do QAM16.

```
function bit = qam16_rx(Im, Re)

width = 2;
binpt = 0;
proto = {xlUnsigned, width, binpt};

bitLsb = xfix(proto, 0);
bitMsb = xfix(proto, 0);

if Im == 1
    bitLsb = 0;
elseif Im == 3
    bitLsb = 1;
elseif Im == -1
    bitLsb = 2;
elseif Im == -3
    bitLsb = 3;
end

if Re == 1
    bitMsb = 0;
elseif Re == 3
    bitMsb = 1;
elseif Re == -1
    bitMsb = 2;
elseif Re == -3
    bitMsb = 3;
end

bit = xl_concat(bitLsb, bitMsb);
```

Apêndice C

Funções calculadas pelo CORDIC

Para qualquer implementação CORDIC, é necessário conhecer o sistema de coordenadas (circular, linear ou hiperbólico), o modo de operação e a seqüência de deslocamento. Com estes três graus de liberdade, diferentes funções podem ser calculadas a partir de um vetor de entrada atribuindo valores de x_0 , y_0 e z_0 [21] [22] [3].

Volder descreve, juntando os conceitos das equações (5.3), (5.4) e (5.5), uma série de iterações composta exclusivamente com operações de soma, subtração e deslocamento da seguinte maneira,

$$\begin{cases} x_{i+1} = x_i - d_i y_i \rho_{m,i} m \\ y_{i+1} = y_i + d_i x_i \rho_{m,i} \\ z_{i+1} = z_i - d_i \gamma_{m,i} \end{cases} \quad (\text{C.1})$$

onde cada variável representa:

- m : pertencente ao conjunto $\{1,0,-1\}$ determina o sistema de coordenadas;
- d_i : assume os valores $\{1,-1\}$ e irá determinar a direção da rotação e, com isso, desempenha um papel fundamental para a convergência do algoritmo;
- $\rho_{m,i}$: é definida como $\rho_{m,i} = d^{-b_{m,i}}$, sendo d a base do sistema numérico e $b_{m,i}$ uma seqüência de deslocamento de números inteiros, geralmente não decrescente. Escolhe-se $\rho_{m,i} = d^{-b_{m,i}}$ por ser uma base binária usada principalmente em operações computacionais; e
- z assume o somatório dos ângulos de rotação $\gamma_{m,i}$.

A cada iteração, um vetor $\vec{v}_i = (x_i, y_i)$ será transformado linearmente num vetor $\vec{v}_{i+1} = (x_{i+1}, y_{i+1})$, e cada uma das variáveis será estudada com intuito de entender o seu propósito dentro do algoritmo CORDIC.

C.1 Sistema de Coordenadas - m

O resultado de qualquer transformação linear depende do sistema de coordenadas utilizado. No caso do CORDIC, o sistema de coordenadas é determinado pela variável m , e pode ser circular, linear ou hiperbólico [3].

Exemplos de sistema de coordenadas:

Considera-se um sistema de coordenadas polares, sendo x e y coordenadas ortogonais de um ponto P , temos que a variável m irá definir a norma de um vetor $\vec{V} = (x, y)$, dada por $\sqrt{x^2 + my^2}$ e o ângulo ϕ , definido por $(\frac{1}{\sqrt{m}}) \tan^{-1}(\frac{y\sqrt{m}}{x})$.

Os sistemas de coordenadas no CORDIC são:

1. Na trajetória da rotação circular onde o m é 1, representado na Figura C.1 a.
 - Raio: $x^2 + y^2 = 1$
 - Ângulo: $\phi = \tan^{-1} \frac{x}{y}$
2. No sistema de coordenadas hiperbólico onde m é -1, representado na Figura C.1 b.
 - Raio: $x^2 - y^2 = 1$
 - Ângulo: $\phi = \tanh^{-1} \frac{x}{y}$
3. No sistema de coordenadas linear onde m é 0, representado na Figura C.1 c.
 - Raio: $x = 1$
 - Ângulo: $\phi = \frac{x}{y}$

A idéia da transformar linearmente um ponto $\vec{P}_i = (x_i, y_i)$ para um ponto $\vec{P}_{i+1} = (x_{i+1}, y_{i+1})$ tal como equação C.2.

$$\begin{cases} x_{i+1} = x_i - m \rho_i y_i \\ y_{i+1} = y_i + \rho_i x_i \end{cases} \quad (\text{C.2})$$

Nas Figura C.1 contém as trajetórias das rotações em cada sistema de coordenadas citadas.

O ângulo de rotação fixa $\gamma_{m,i}$ vai diminuindo em módulo à medida que as iterações vão sendo incrementadas devido à seqüência de deslocamento. O efeito rotacional pode ocorrer tanto no sentido horário quanto no sentido anti-horário, de acordo com o esquema de controle de convergência do algoritmo.

Sendo ϕ o somatório dos ângulos $\gamma_{m,i}$, temos:

$$\begin{cases} \gamma_{m,i} = (\frac{1}{\sqrt{m}}) \tan^{-1}(\sqrt{m}\rho_i) \\ \phi = \sum_{i=0}^{n-1} d_i \gamma_{m,i} \end{cases} \quad (\text{C.3})$$

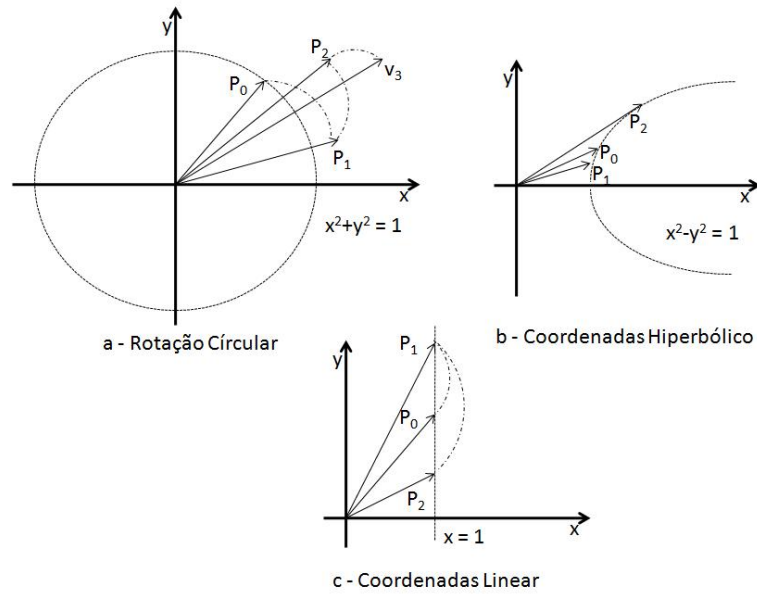


Figura C.1: Trajetórias das rotações em cada sistema de coordenadas.

se acumularmos estas k_i expansões numa variável K_n , podemos corrigir o resultado através de um produto por um escalar, retirando o efeito das expansões. Este é o chamado fator de correção de escala, sendo também um parâmetro para verificar se o algoritmo convergiu no sistema de coordenadas adotado.

$$\begin{cases} k_i = \sqrt{1 + m\rho_i^2} \\ K_n = \prod_{i=0}^{n-1} k_i \end{cases} \quad (\text{C.4})$$

Uma terceira componente z_i armazena o ângulo total acumulado nas iterações sucessivas. Depois de n iterações,

$$\begin{cases} z_{i+1} = z_i - d_i \gamma_{m,i} \\ z_n = z_0 - \sum_{i=0}^{n-1} d_i \gamma_{m,i} \end{cases} \quad (\text{C.5})$$

onde z_n é igual a diferença entre o ângulo inicial z_0 e o ângulo total de rotação acumulado nas n iterações.

C.2 Modos de Operação - d_i

Assim como o sistema de coordenadas, a escolha do modo de operação permite que se chegue a resultados específicos. Esta escolha é feita substituindo d_i por uma determinada direção de cada rotação [3]. A variável d_i pode assumir os valores $\{1, -1\}$, e quando o produto de d_i e de x_i for

positivo na Equação (C.1), o sentido da rotação é anti-horário. Por sua vez, quando o produto destas variáveis for negativo, o sentido da rotação é horário. Os modos de operações principais são os modos de rotação e o de vetorização.

C.2.1 Modo de Rotação - Z-Reduction

Caracteriza-se por ir reduzindo o valor da variável z a cada iteração convergindo a variável para zero. Quando isto acontece, o ângulo total de rotação acumulado é exatamente igual ao ângulo inicial z_0 escolhido.

Exemplo de modo de rotação:

Um vetor $\vec{v} = (x, y)$ de entrada é atribuído um ângulo inicial de rotação $z_0 = \varphi$. Com isto, tem $x_0 = x, y_0 = y$ e $z_0 = \varphi$, onde:

$$\left\{ \begin{array}{l} z_n = \varphi - \sum_{i=0}^{n-1} d_i \gamma_{m,i} \end{array} \right. \quad (\text{C.6})$$

Quando $z_n = 0, \varphi = \phi = \sum_{i=0}^{n-1} d_i \gamma_{m,i}$, ou seja, o ângulo total acumulado de rotação é igual a φ . Para que isto aconteça, o esquema de controle $d_i = \text{sign}(z_i)$ é usado, obtendo-se:

$$\left\{ \begin{array}{l} x_{i+1} = x_i - \text{sign}(z_i) y_i \rho_{m,i} m \\ y_{i+1} = y_i + \text{sign}(z_i) x_i \rho_{m,i} \\ z_{i+1} = z_i - \text{sign}(z_i) \gamma_{m,i} \end{array} \right. \quad (\text{C.7})$$

Na Figura C.2 é mostrada a trajetória para o modo de rotação no sistema de coordenadas circular. É evidente a expansão não uniforme a cada iteração, que pode ser corrigida com o fator de correção de escala.

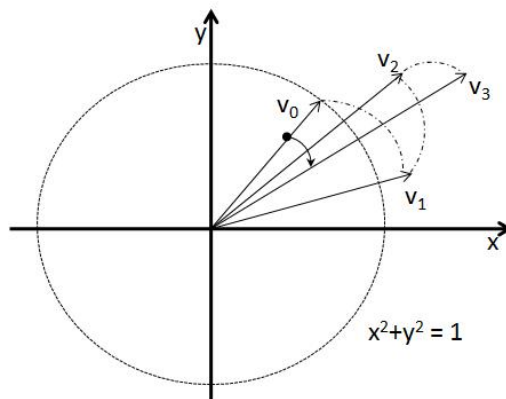


Figura C.2: Trajetórias das rotações - modo de rotação.

C.2.2 Modo de Vetorização - *Y-Reduction*

Tem como finalidade rotacionar um vetor de entrada $\vec{v} = (x, y)$ ao redor do eixo x . Isto é feito através de um esquema de controle capaz de forçar y_0 para zero durante as iterações. A rotação ao redor do eixo x pode ser tanto no eixo positivo ($x_0 \geq 0$) quanto no eixo negativo ($x_0 \leq 0$), dependendo do sinal de x_0 , e quando $y_n = 0$, y_n contém o ângulo total de rotação φ depois de n iterações.

Exemplo de modo de vetorização:

Um vetor $\vec{v} = (x, y)$ de entrada é atribuído um ângulo inicial de rotação $z_0 = \varphi$. Com isto, tem $x_0 = x$, $y_0 = y$ e $z_0 = \varphi$, onde:

$$\left\{ \begin{array}{l} z_n = -\varphi - \sum_{i=0}^{n-1} d_i \gamma_{m,i} \end{array} \right. \quad (\text{C.8})$$

O esquema de controle no modo *Y-reduction* é $d_i = -\text{sign}(x_i)\text{sign}(y_i)$. Substituindo a variável d_i na Equação (C.1), tem-se:

$$\left\{ \begin{array}{l} x_{i+1} = x_i + \text{sign}(x_i)\text{sign}(y_i) y_i \rho_{m,i} m \\ y_{i+1} = y_i - \text{sign}(x_i)\text{sign}(y_i) x_i \rho_{m,i} \\ z_{i+1} = z_i + \text{sign}(x_i)\text{sign}(y_i) \gamma_{m,i} \end{array} \right. \quad (\text{C.9})$$

Ao final das iterações, $x_i = K_n \text{sign}(x_0) \sqrt{x^2 + my^2}$. Na Figura C.3 pode-se observar a trajetória para o modo *Y-Reduction* no sistema de coordenadas circular de um vetor $\vec{v} = (x, y)$.

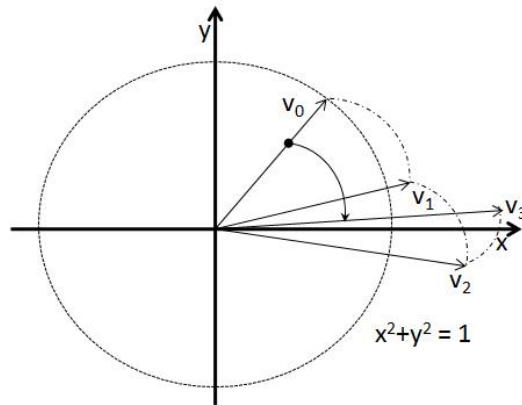


Figura C.3: Trajetórias das rotações - modo de vetorização.

C.3 Convergência do Algoritmo - γ

Sendo o número de iterações igual a n rotações de ângulos fixos com direção variável, o ângulo desejado de rotação A_0 pode apresentar um erro de arredondamento $\Delta\phi$ dado por:

$$\Delta\phi = A_0 - \sum_{i=0}^{n-1} d_i \gamma_{m,i} \quad (\text{C.10})$$

O erro de arredondamento $\Delta\phi$ não inclui erro de quantização finita dos ângulos de rotação $\gamma_{m,i}$. Pode-se definir dois critérios de convergência da Equação (C.10):

1. O ângulo escolhido de rotação deve satisfazer à condição:

$$\gamma_{m,i} - \sum_{j=i+1}^{n-1} \gamma_{m,j} \leq \gamma_{m,n-1} \quad (\text{C.11})$$

se o ângulo desejado para rotação foi atingido, mas ainda restam $n - i$ iterações ($A_0 = 0$), significa que na próxima iteração o vetor bidimensional sofrerá uma rotação $\pm\gamma_{m,i}$. Neste caso, o somatório dos ângulos restantes de rotação $\sum_{j=i+1}^{n-1} \gamma_{m,j}$ seria suficiente para fazer com que o ângulo de rotação final A_n , após a última iteração n , seja zero com acurácia de $\gamma_{m,j}$.

2. O ângulo de rotação A_0 não deve exceder à região de convergência da determinada iteração, ou seja, a soma de todos os ângulos restantes mais o ângulo final $\gamma_{m,n-1}$:

$$|A_0| \leq \sum_{i=0}^{n-1} \gamma_{m,i} + \gamma_{m,n-1} \quad (\text{C.12})$$

Exemplo de convergência do algoritmo:

A seqüência de deslocamento convergente a cada rotação de um ângulo fixo, dado por:

$$\gamma_{m,i} = \left(\frac{1}{\sqrt{m}}\right) \tan^{-1}(\sqrt{m}b_{m,i}) \quad (\text{C.13})$$

sendo $i \in \{0, \dots, n - 1\}$

Uma mesma seqüência de deslocamento nem sempre converge em diferentes sistemas de coordenadas, o que possibilita a atuação do algoritmo em diferentes regiões de convergência, com fatores de correção de escala diferentes. Assim que for escolhida a seqüência, independente do modo de operação, pode-se calcular antecipadamente os ângulos fixos $\gamma_{m,i}$ e o fator de correção de escala.

A seqüência escolhida independe do modo de operação, pode-se calcular antecipadamente os ângulos fixos $\gamma_{m,i}$ e o fator de correção de escala. Para os sistemas de coordenadas circular e linear, é intuitiva a escolha do conjunto dos números naturais para $b_{m,i}$, pois a seqüência seria inteira e não

decrecente, fazendo com que $\rho_{m,i} = d^{-b_{m,i}}$ sempre convirja. Já quando o sistema de coordenadas for hiperbólico, a seqüência anterior convergirá se repetirmos algumas iterações para alcançar o resultado desejado.

Esta idéia foi desenvolvida por Walther e consiste em montar a seqüência $b_{m,i}$ com o conjunto de naturais repetindo as iterações $3k + 1$, sendo k igual às iterações $\{1, 2, \dots, 3\}$. A tabela a seguir mostra as seqüências de deslocamento para cada sistema de coordenadas.

m	$b_{m,i}$	$ A_0 $	$K_m(n \rightarrow \infty)$
1	0,1,2,...,i,...	$\approx 1,74$	$\approx 1,6468$
0	1,2,...,i,...	1,00	1,0000
-1	1,2,3,4,5,...	$\approx 1,13$	$\approx 1,8282$

Tabela C.1: Seqüência de deslocamento do CORDIC.

C.4 Funções CORDIC

Para um melhor entendimento, a Tabela C.1 mostra a permutação das combinações acima e as respectivas funções obtidas em cada modo de operação.

Exemplo da utilização da Tabela C.1 é a implementação da função seno CORDIC:

A função seno CORDIC pode ser obtida no sistema de coordenadas circular, modo *Z-Reduction* com o vetor de entrada sendo,

$$\begin{cases} x_0 = 1/K_1(n) \\ y_0 = 0 \\ z_0 = \phi \end{cases} \quad (\text{C.14})$$

Supondo $\phi = 68^\circ$, com 5 bits de precisão, temos a seqüência de rotações a seguir apresentada na Tabela C.2.

A Figura C.4 mostra a trajetória das rotações CORDIC no cálculo do seno de 68° . A variável z , iniciada com $\phi = 68^\circ$, converge para o valor 0, ou seja, atinge o ângulo desejado. O valor do seno pode ser visto na variável y , que está ilustrada sem a aplicação do fator de correção de escala.

m	Mode de Operação	Entrada	Saída	Funções
1	rotação	$x_0 = x$ $y_0 = y$ $z_0 = \phi$ $x_0 = 1/K_1(n)$ $y_0 = 0$ $z_0 = \phi$	$x_n = K_1(n)(x \cos \phi - y \text{sen} \phi)$ $y_n = K_1(n)(x \cos \phi + y \text{sen} \phi)$ $z_n = 0$ $x_n = \cos \phi$ $y_n = \text{sen} \phi$ $z_n = 0$	tan sen cos csc sec
1	vetorização	$x_0 = x$ $y_0 = y$ $z_0 = \phi$	$x_n = K_1(n) \text{sign}(x_0) \sqrt{x^2 + y^2}$ $y_n = 0$ $z_n = \phi + \tan^{-1} y/x$	arctan
0	rotação	$x_0 = x$ $y_0 = y$ $z_0 = z$	$x_n = x$ $y_n = y + xz$ $z_n = 0$	multiplicação soma
0	vetorização	$x_0 = x$ $y_0 = y$ $z_0 = z$	$x_n = x$ $y_n = 0$ $z_n = z + y/x$	divisão
-1	rotação	$x_0 = x$ $y_0 = y$ $z_0 = \phi$ $x_0 = 1/K_{-1}(n)$ $y_0 = 0$ $z_0 = \phi$	$x_n = K_{-1}(n)(x \cosh \phi - y \text{senh} \phi)$ $y_n = K_{-1}(n)(x \cosh \phi + y \text{senh} \phi)$ $z_n = 0$ $x_n = \cosh \phi$ $y_n = \text{senh} \phi$ $z_n = 0$	tanh senh cosh exp
-1	vetorização	$x_0 = x$ $y_0 = y$ $z_0 = \phi$	$x_n = K_{-1}(n) \text{sign}(x_0) \sqrt{x^2 + y^2}$ $y_n = 0$ $z_n = \phi + \tan^{-1} y/x$	arctanh sqrt ln

Tabela C.2: Funções calculadas pelo CORDIC [3].

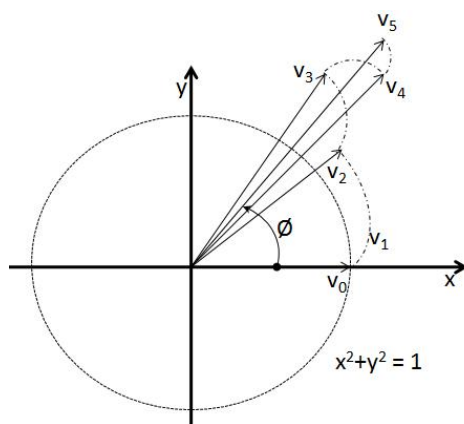


Figura C.4: Trajetórias das rotações CORDIC no cálculo do seno de 68°.

Iteração	Entrada	Saída	Parâmetros CORDIC
1	$x_0 = x$ $y_0 = y$ $z_0 = 68^\circ$	$x_1 = 0,7071$ $y_1 = 0,7071$ $z_1 = 23^\circ$	$\alpha_1 = 45^\circ$ $K_1 = 1,4142$ $m_1 = +1$
2	$x_1 = 0,7071$ $y_1 = 0,7071$ $z_1 = 23^\circ$	$x_2 = 0,3162$ $y_2 = 0,9487$ $z_2 = -3,5^\circ$	$\alpha_2 = 26,5^\circ$ $K_2 = 1,1180$ $m_2 = +1$
3	$x_2 = 0,3162$ $y_2 = 0,9487$ $z_2 = -3,5^\circ$	$x_3 = 0,5369$ $y_3 = 0,8437$ $z_3 = 10,5^\circ$	$\alpha_3 = 14^\circ$ $K_3 = 1,0308$ $m_3 = -1$
4	$x_3 = 0,5361$ $y_3 = 0,8431$ $z_3 = 10,3^\circ$	$x_4 = 0,4281$ $y_4 = 0,9037$ $z_4 = 3,3^\circ$	$\alpha_4 = 7^\circ$ $K_4 = 1,0078$ $m_4 = +1$
5	$x_4 = 0,4281$ $y_4 = 0,9031$ $z_4 = 3,3^\circ$	$x_5 = 0,3709$ $y_5 = 0,9287$ $z_5 = -0,2^\circ$	$\alpha_5 = 3,5^\circ$ $K_5 = 1,0020$ $m_5 = +1$

Tabela C.3: Exemplo das iterações CORDIC para o cálculo do seno de 68° [3].