Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Departamento de Engenharia Elétrica

TRABALHO DE CONCLUSÃO DE CURSO

Simulação em FPGA de uma Máquina Síncrona a Ímã Permanente em Tempo Real Utilizando Ponto Flutuante

VICTOR DE SOUSA CAVALCANTE

Campina Grande

Junho de 2013

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

TRABALHO DE CONCLUSÃO DE CURSO

Simulação em FPGA de uma Máquina Síncrona a Ímã Permanente em Tempo Real Utilizando Ponto Flutuante

Aluno

Victor de Sousa Cavalcante

Orientador

Alexandre Cunha Oliveira

Campina Grande

Junho de 2013

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

TRABALHO DE CONCLUSÃO DE CURSO

Simulação em FPGA de uma Máquina Síncrona a Ímã Permanente em Tempo Real Utilizando Ponto Flutuante

Trabalho de conclusão de curso apresentado ao Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande em cumprimento às exigências para obtenção do grau de Bacharel em Engenharia Elétrica

ALUNO: _____

Victor de Sousa Cavalcante

ORIENTADOR: _____

Alexandre Cunha Oliveira

Campina Grande

Junho de 2013

AGRADECIMENTOS

Primeiramente eu gostaria de agradecer a meus familiares que sempre conversavam comigo durante o meu curso e principalmente no meu tempo de intercâmbio apesar da distância e do fuso-horário. Especialmente à minha irmã Eugênia Cavalcante e aos meus pais, Antonia Cavalcante e José Ivan Cavalcante.

Igualmente, eu sou muito grato aos meus grandes amigos Arthur Monteiro, João Viniccius Gomes, Matheus Telles, Rafael Ângelo Vieira e Túlio Vidal que me acompanharam desde o início do curso compartilhando momentos bons e ruins.

Ainda, agradeço a minha namorada Tiziana Mombelli que eu tanto amo e que apesar de morar a milhares de quilômetros de distância ela é a melhor companheira que eu poderia ter, me acalmando, ajudando e amando nas horas mais difíceis de maneira incomparável.

Por fim, e não menos importante, gostaria de agradecer a orientação do professor Alexandre neste projeto. Sua orientação foi de grande ajuda ao decorrer deste estudo. Com certeza não poderia ter encontrado uma orientação melhor.

Dedico este trabalho à minha mãe e ao meu pai que me apoiaram durante esta longa caminhada. A Tizi que tem me proporcionado momentos felizes. Aos meus grandes amigos que sem eles eu não chegaria até aqui.

RESUMO

Tendo em vista a tendência crescente e dominante das simulações em tempo real e do avanço dos dispositivos de *hardware* reconfiguráveis, este estudo visa um aprimoramento da utilização de dispositivos FPGA na simulação em tempo real de plantas genéricas. A planta de teste utilizada neste projeto foi uma máquina síncrona a ímã permanente.

A precisão e a reutilização destes simuladores é um ponto importante que merece aprofundamento e que apresenta algumas dificuldades quando são implementados utilizando aritmética de ponto fixo. O formato de ponto flutuante já é bastante utilizado atualmente na computação. Contudo, em *hardware* ainda é um objeto complexo.

Neste projeto foram testadas quatro FPU diferentes com diversas latências e tamanhos. Elas utilizam tanto a precisão simples como a dupla de acordo com o padrão IEEE 754. Com estas FPU, foram projetados vários simuladores em tempo real para a planta em estudo. Os resultados foram analisados no computador em comparação a um modelo de referência mais preciso calculado em um *software* no computador.

O conjunto de avaliações realizadas quanto à implememtação das FPU, bem como com relação às formas como os simuladores foram projetados são compiladas em uma série de conclusões apresentadas no fim deste relatório, em conjunto com sugestões de trabalhos futuros que podem ser desenvolvidos tomando este projeto como base.

Palavras-Chaves: Hardware-in-the-loop; FPGA; Ponto flutuante; SystemVerilog; Unidade de Ponto Flutuante; Verilog.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

A

A/D – *Analog/Digital*

ASIC – *Application-Specific Integrated Circuit*

B

BDCM – *Brushless DC Machines*

D

D/A – *Digital/Analog*

DSP – *Digital Signal Processing*

E

EL – Elementos Lógicos

F

FPGA – *Field-Programmable Gate Array*

FP – *Floating Point*

FPU – *Floating Point Unit*

G

GPP – *General Purpose Processor*

H

HDL – *Hardware Description Language*

HILS – *Hardware in the Loop Simulation*

I

IC – *Integrated Circuit*

IP – *Intellectual Property*

M

MAC – *Multiply-Accumulate*

N

NaN – *Not a Number*

P

PC – *Personal Computer*

PLL – *Phase Locked Loop*

PMSM – *Permanent Magnet Synchronous Machines*

PWM – *Pulse-Width Modulation*

R

RTOS – *Real-Time Operating System*

S

SoC – *System-on-Chip*

V

VHDL – *VHSIC Hardware Description Language*

VHSIC – *Very-High Speed Integrated Circuit*

SUMÁRIO

1. Introdução

O uso de dispositivos eletrônicos chamados *Field-Programmable Gate Array* (FPGA) na realização de simulações em tempo real tem uma tendência crescente e dominante, principalmente no contexto do *Hardware in the Loop Simulation* (HILS) [2]. A Figura 1 ilustra uma configuração típica do HILS. No canto inferior esquerdo da figura, está representado um simulador em tempo real da planta que é a base de cálculo do modelo matemático do sistema simulado e pode ser um computador com vários núcleos de processamento. A planta é conectada ao simulador através de uma unidade de controle eletrônica e física composta por filtros, fontes de alimentação, isolamento, etc. A capacidade de processamento do simulador é estendida pelo uso de placas com FPGA que são acessadas pelo computador através de conexões de alta velocidade.

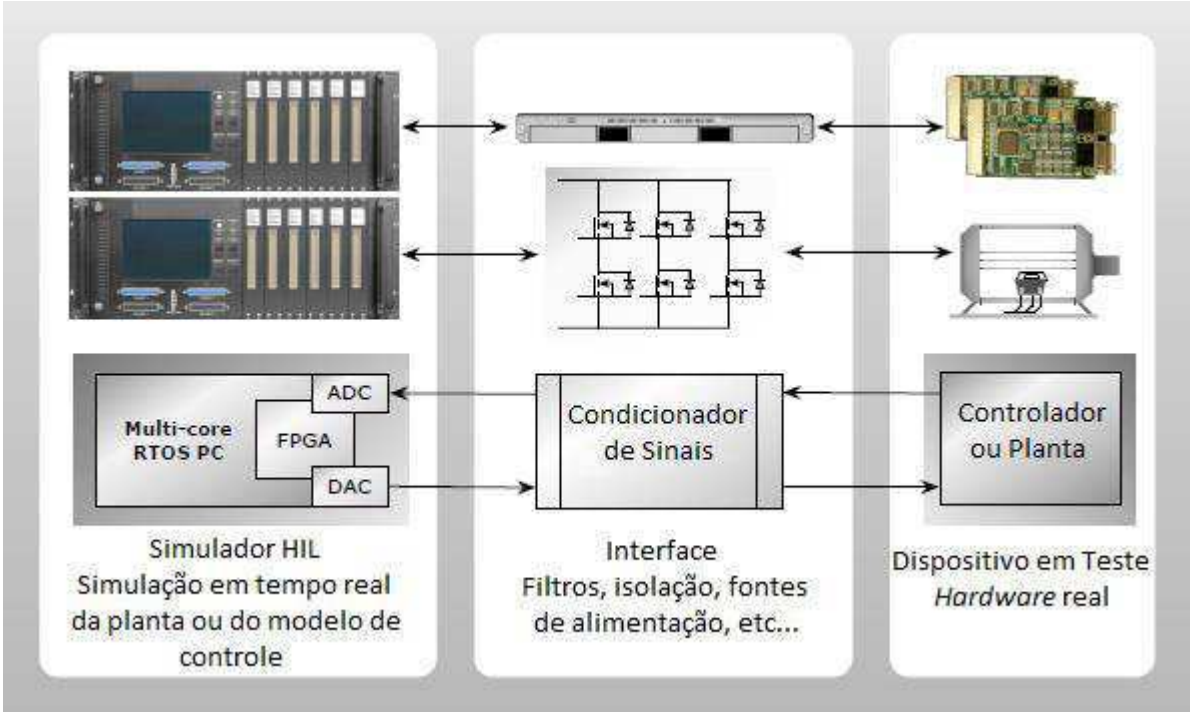

Figura 1: Configuração típica do HILS [2]

Estas placas com FPGA ligadas ao simulador têm vários objetivos. Primeiramente, elas ajudam na interface que atua no *hardware* testado, fazendo as conversões dos sinais analógicos para digitais e vice-versa. Segundo, elas servem como fontes de estímulos, gerando sinais senoidais, PWM, etc.. Por último, elas podem embarcar modelos matemáticos das plantas a serem simuladas, ou parte

destes modelos, que apresentem requisitos temporais críticos (passo de cálculo igual ou inferior a 1 μs). O trabalho desenvolvido e que será apresentado neste relatório teve como foco principal o último caso, conhecido como simulação *on-chip* ou simulação embarcada.

Um conjunto de fatores motiva o interesse por simulação embarcada. Primeiramente, a precisão e a fidelidade da simulação em tempo real de sistemas de potência e eletromecânicos, que necessitam de passos de cálculo menores do que um microssegundo. Processadores modernos de uso geral (GPP) não conseguem passo de cálculo menor do que 2 a 20 microssegundos, para sistemas de simulação de tamanhos moderados. Portanto, o uso de GPP é limitado a uma pequena quantidade de aplicações [2]. Segundo, as conexões de alta velocidade usadas no acesso às placas dos conversores D/A possuem uma latência intrínseca de 2 a 5 microssegundos. Essa latência também limita a quantidade de sistemas que podem ser simulados apenas com GPP. Finalmente, os dispositivos FPGA apresentaram um impressionante crescimento na sua capacidade interna nos últimos 5 a 10 anos, com consequente reflexo na complexidade de dispositivos digitais sintetizáveis, podendo implementar dispositivos com elevada capacidade de processamento. Hoje, simulações embarcadas não são apenas um objetivo alcançável, são uma realidade provada por ser efetiva em uma quantidade numerosa de aplicações [1], [2], [3].

Entretanto, apesar de todos os benefícios anteriormente citados, muitos desafios ainda precisam ser enfrentados para que a adoção de FPGA seja mais difundida em aplicações voltadas para simulação de sistemas em tempo real. O predomínio do formato em ponto fixo utilizado nos projetos em FPGA e a complexidade na utilização do ponto flutuante são importantes desafios a serem enfrentados. A utilização de ponto fixo implica em um trabalho suplementar para fazer um pré-processamento dos dados e é um processo diretamente dependente do tipo de aplicação do sistema. Já o sistema que utilize o ponto flutuante pode ser reutilizado para diferentes aplicações, no entanto, apresenta uma maior complexidade de implementação devido a utilização de uma Unidade Aritmética de Ponto Flutuante que deve ser sintetizada em conjunto com o modelo matemático da planta. Espera-se que uma futura popularidade da aritmética de ponto flutuante no domínio da computação reconfigurável ajude a popularizar o uso da FPGA no

âmbito do HIL. Contudo, devido ao circuito complexo que ela detém, os operadores de ponto flutuante precisam ser utilizados com bastante cuidado para alcançar um ótimo desempenho com pequenos passos de cálculo.

Com esta temática em vista, este trabalho foi focado no problema do uso de diferentes unidades de ponto flutuante (FPU) em FPGA na resolução de diversas equações em tempo real. Como estudo de caso, foram utilizadas as equações matemáticas do modelo discreto de uma máquina síncrona a imã permanente [4]. A partir destas equações, realizou-se a simulação em tempo real deste modelo em FPGA. A sessão 3 descreve detalhadamente como se deu o processo de desenvolvimento deste sistema.

Na sequência, na sessão 2, é feita uma explanação de alguns conceitos básicos e ferramentas utilizadas neste projeto. Na sessão 3 é explicado todo o projeto desenvolvido neste trabalho, assim como um resumo e análise dos resultados obtidos. Por fim, na sessão 4 é apresentada uma conclusão a respeito do que foi desenvolvido e apresentadas sugestões de trabalhos futuros, tomando por base o presente projeto.

1.1 Objetivos

O trabalho realizado teve como principal objetivo avaliar um conjunto de implementações de unidades aritméticas de ponto flutuante em FPGA, aplicadas a simulação tempo real de modelo matemático de uma planta física (máquina a Ímã Permanente operando como motor). O modelo matemático utilizado para exemplificar o uso deste sistema foi o modelo *dq* de uma máquina síncrona a ímã permanente [4]. Este modelo será explicado brevemente nas próximas sessões. Além da vantagem da precisão nos cálculos matemáticos quando comparada a obtida com o uso de ponto fixo, um sistema que utiliza ponto flutuante também tem a vantagem de apresentar uma portabilidade entre modelos matemáticos e assim, poder ser reutilizado independentemente da ordem de grandeza das variáveis do sistema.

Outro objetivo do estudo realizado esteve relacionado à observação de parâmetros como: latência do sistema, área ocupada da FPGA e frequência máxima

de operação, face às modificações na forma de sintetização do modelo matemático da planta sendo simulada.

1.2 Cronograma

Inicialmente, este trabalho de conclusão de curso foi dividido em diversas tarefas. As principais foram:

1. Revisão sobre representação numérica em ponto flutuante e da norma IEEE 754 [5]; Revisão bibliográfica sobre simulação em tempo real utilizando FPGA;
2. Identificação nos trabalhos avaliados de versões de FPU que satisfizessem os requisitos do projeto;
3. Montagem de um ambiente de teste para validar as FPU escolhidas e aprender seu funcionamento;
4. Revisão sobre máquinas síncronas à imã permanente;
5. Projeto e teste de um acumulador multiplicativo (MAC);
6. Avaliação do método numérico escolhido para resolver o modelo matemático e aplicação nas equações do problema;
7. Projeto e teste do simulador;
8. Desenvolvimento da interface para aquisição dos dados;
9. Descrição e teste do simulador variando a quantidade de unidades de ponto flutuante instanciadas e o algoritmo de como os cálculos são realizados;
10. Pesquisa de versões de FPU a serem utilizadas e testadas no projeto;
11. Implementação do simulador utilizando os vários modelos de FPU avaliadas em diferentes frequências de operação;
12. Comparação e análise dos resultados obtidos;
13. Conclusões sobre o trabalho e escrita do relatório final.

Este trabalho de conclusão de curso em Engenharia Elétrica foi realizado durante 15 semanas dos dias 25 de Fevereiro ao dia 7 de Junho. A Tabela 1 apresenta o cronograma seguido durante este trabalho com respeito às atividades anteriormente citadas.

Tabela 1: Cronograma realizado no trabalho de conclusão de curso

Semanas/ Tarefas	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															
11															
12															
13															

2. Conceitos Básicos

2.1 Visão geral do estado da arte

A modelagem das máquinas de indução é um exemplo do que as simulações em tempo real embarcadas podem fazer. Para isto, o modelo matemático precisa ser calculado com um passo de cálculo inferior a um microssegundo para comportar o chaveamento de alta frequência da fonte PWM que alimenta a máquina. Dentre os vários simuladores embarcados em tempo real que são conhecidos na literatura, a representação em espaço de estados é a mais citada e a mais usada, podendo ser considerado o método de escolha para modelagem de sistemas no contexto de simulação em tempo real. A Tabela 2 mostra uma seleção de arquiteturas extraídas de [11], [12], [13], [14], [15], [16] e [2], que utilizam modelos de espaço de estados com uma aritmética tanto de ponto fixo como de ponto flutuante. Os autores identificam os respectivos passos de cálculo e arquiteturas utilizadas, embora algumas não sejam claramente descritas. A escolha do formato numérico é um fator importante na complexidade do *hardware* que fará a solução de um modelo em espaço de estados. Como pode ser visto na Tabela 2, faz-se uma comparação apenas entre os formatos de ponto fixo e ponto flutuante, já que estes são os mais utilizados [2]. Na sessão 2.2 será apresentada uma breve explanação de como é feita a representação de números em ponto flutuante de acordo com a norma do IEEE [5].

Tabela 2: Comparação entre as simulações embarcadas encontradas nas referências

Referência	[11]	[12]	[13]	[14]	[15]	[16]	[2]
Formato	Fixo	Fixo	Fixo	Flutuante	Flutuante	Flutuante	Flutuante
Arquitetura	Não divulgada	Parcialmente divulgada	Totalmente divulgada	Não divulgada	Parcialmente divulgada	Totalmente divulgada	Totalmente divulgada
Passo de cálculo	0,3 µs	10 µs	0,1 µs	0,6 µs	0,2 µs	0,8 µs	1 µs

2.2 Formatos numéricos

2.2.1 Ponto fixo

O formato de ponto fixo é uma representação de números reais que consiste em um conjunto finito de valores equidistantes representados por uma quantidade fixa de bits antes e depois do ponto da vírgula. O formato de ponto fixo tem a

vantagem da disponibilidade de operadores de aritmética básicos na FPGA, como por exemplo, somadores e multiplicadores. Os operadores básicos são facilmente projetados em *hardware* e caracterizados por alta frequência de *clock* e baixa latência. Entretanto, o formato de ponto fixo tem uma faixa de valores restrita para cada aplicação, já que a quantidade de bits representando a parte fracionária dos números é predefinida no processo de normalização e não pode ser alterado. A aproximação por unidade (p.u.) é uma opção válida para operadores embarcados de ponto fixo [11], mas ainda requer uma fase de especificação da resolução e faixa de todas as quantidades físicas normalizadas.

Além de ter uma baixa precisão para diversas aplicações, esta representação numérica não é portável de uma aplicação para outra já que precisa de uma especificação diferente e assim, uma definição diferenciada de bits representando a parte fracionária dos números representados para cada sistema diferente.

A Figura 2 mostra como esta representação numérica é implementada:

Figura 2: Representação numérica por ponto fixo

2.2.2 Ponto flutuante

O formato de ponto flutuante é bastante conhecido na prática de aritmética digital e é padronizado pelo IEEE através da norma IEEE 754 [5]. A norma define que um número real *x* pode ser representado no formato de ponto flutuante pelo conjunto (*s*; *e*; *m*), tal que:

$$x = (-1)^s 2^e m$$

Onde *s* é o bit de sinal, *e* é o expoente e *m* a mantissa. O valor de *m* é restrito à faixa [1; 2) para assegurar a unicidade da representação de *x*. A representação em ponto flutuante também admite casos especiais, como por exemplo, zero, infinito e NaN. Existem diferentes tipos de representação no formato ponto flutuante.

A Figura 3 ilustra estes tipos e eles serão escolhidos de acordo com a aplicação. A precisão simples (*single precision*) utiliza 1 bit para sinal, 8 bits para o expoente e os 24 restantes para a mantissa. A precisão dupla (*double precision*) utiliza 1 bit para sinal, 11 para o expoente e os 53 restantes para a mantissa. Posteriormente foi definido o tipo estendido, que utiliza 1 bit para o sinal, 15 para o expoente e 64 para a mantissa. Neste trabalho serão discutidas apenas as representações em precisão simples e dupla por serem as mais utilizadas [2].

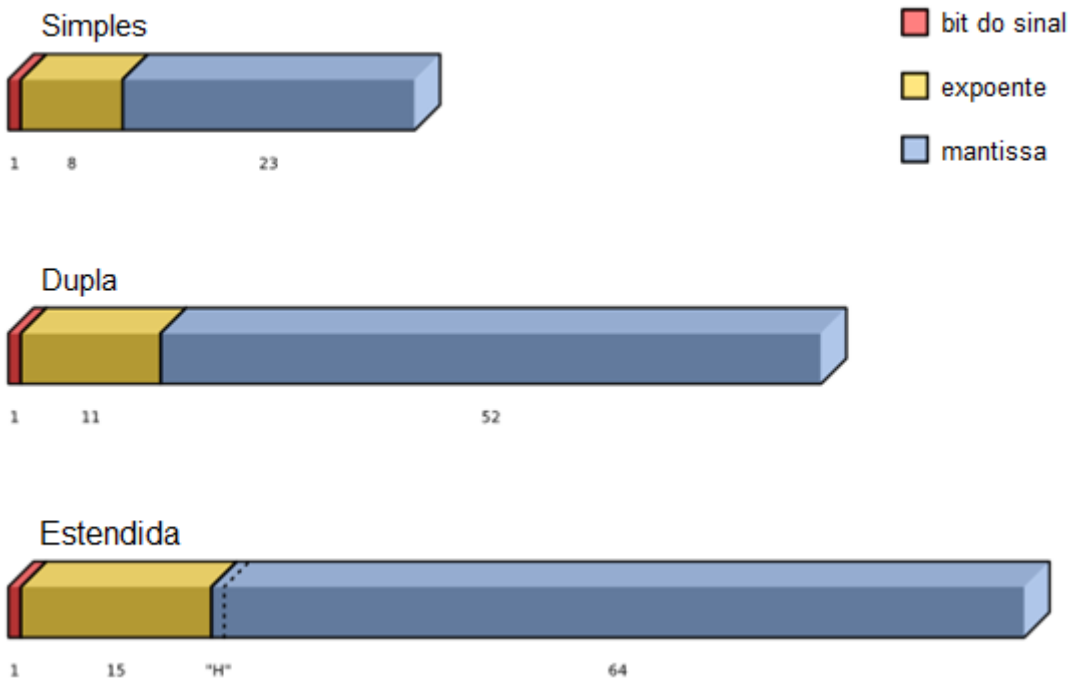

Figura 3: Tipos do formato ponto flutuante [5]

Na mesma norma são definidos os casos especiais citados anteriormente para a precisão simples e dupla. A Tabela 3 resume estes cinco casos especiais [5].

Tabela 3: Comparação entre as simulações embarcadas encontradas nas referências

	Sinal (s)	Expoente (e)	Mantissa (m)	Valor (x)
Zero Positivo	0	000000... 000	000000... 000	+0
Zero Negativo	1	000000... 000	000000... 000	-0
Infinito Positivo	0	111111... 111	000000... 000	+∞
Infinito Negativo	1	111111... 111	000000... 000	-∞
NaN	0 ou 1	111111... 111	≠ 00000... 000	NaN

A vantagem de se utilizar ponto flutuante para propósitos de modelagem é a correspondência direta de um para um de todas as quantidades físicas de um modelo sendo simulado em tempo real ou não. Entretanto, os operadores de ponto flutuante são mais complexos do que os de ponto fixo. Os operadores de ponto flutuante ocupam uma maior área e têm uma maior latência do que os operadores de ponto fixo.

Com o crescimento contínuo em tamanho e capacidade de acomodar a descrição de dispositivos de processamento com elevado poder de processamento dos dispositivos reconfiguráveis, as FPGA modernas, como por exemplo, Virtex7 da Xilinx ou Stratix V da Altera são atualmente indicadas para o projeto de arquiteturas complexas de processamento digital de sinais (DSP) utilizando ponto flutuante com precisão simples ou dupla. Dependendo da aplicação, até mesmo as FPGA mais antigas como Cyclone II da Altera podem ser suficientes para satisfazer os requisitos desejados do sistema.

Com o intuito de ser eficiente, o projeto de *hardware* de algoritmos DSP seguem um método típico: as dependências aritméticas de dados são expressas em um gráfico de fluxo de dados de formato equivalente, para conseguir estabelecer um algoritmo que use um paralelismo em potencial. Com isto, o gráfico é mapeado para uma arquitetura de *hardware* ótima, fazendo o balanço dos critérios de simulação, como por exemplo, tamanho do modelo, velocidade e passo de cálculo, em oposição com os recursos disponíveis. A última fase é crucial e requer um bom conhecimento de técnicas de projeto em FPGA para algoritmos DSP. Consequentemente, os números reais em ponto flutuante ajudam a resolver o problema da faixa dinâmica e permitem o projeto de algoritmos DSP complexos, mas eles também trazem uma desvantagem na velocidade do projeto final.

O acúmulo de valores em ponto flutuante é um grande e crucial desafio que precisa ser resolvido neste projeto para resolver integrações numéricas no intuito de encontrar as soluções do espaço de estados.

2.3 *Field-Programmable Gate Array* (FPGA)

Uma FPGA é um circuito integrado projetado para ser configurado pelo consumidor ou por um projetista após a sua compra. As FPGA contemporâneas têm

uma grande quantidade de recursos como portas lógicas, blocos de memórias RAM e circuitos *Phase Locked Loop* (PLL), dentre outros, para resolver desde problemas simples aos mais complexos.

Uma FPGA pode ser usada para desempenhar qualquer função lógica que um ASIC digital poderia realizar. As habilidades de atualizar as funcionalidades após a compra do dispositivo, reconfiguração parcial de uma parte do projeto e o baixo custo relativo ao projeto de um ASIC, oferecem vantagens em muitas aplicações nos projetos que utilizam FPGA.

A Figura 4 ilustra os componentes de uma FPGA que podem ser classificados em três principais blocos: blocos lógicos, blocos de entrada/saída e as interconexões programáveis. Os blocos lógicos podem ser configurados para desempenhar desde as funções mais simples como NAND e NOR até as funções combinatórias mais complexas. Na maioria das FPGA, estes elementos também incluem memórias que podem ser desde simples *flip-flops* a blocos mais completos de memórias. As interconexões programáveis são blocos que conectam ou desconectam as trilhas dentro da FPGA de acordo com o circuito que for especificado. Por fim, os blocos de entrada e saída são aqueles que fazem a ligação dos sinais internos e externos ao dispositivo.

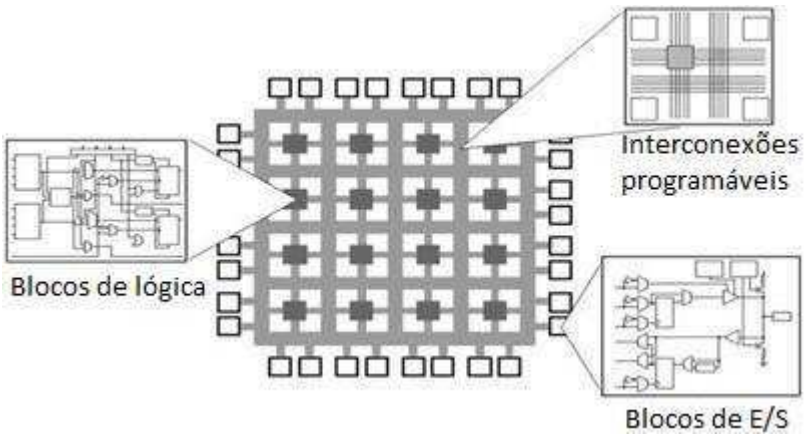

Figura 4: Estrutura interna de uma FPGA

Algumas FPGA também possuem elementos analógicos em adição aos digitais. Algumas FPGA de sinais mistos têm conversores analógico/digital (ADC) e digital/analógico (DAC) integrados. Estes conversores são condicionadores de sinais analógicos que permitem que elas funcionem como um SoC. Outro exemplo de circuito analógico que pode ser encontrado neste dispositivo são os circuitos PLL.

Devido à possibilidade de reconfigurar todo o circuito e de criar circuitos dedicados e bastante complexos de forma relativamente fácil, que realizem os cálculos desejados a um passo de cálculo pequeno e praticamente impossível para os processadores de propósito geral, foi escolhido utilizar um dispositivo FPGA neste projeto para implementar o simulador em tempo real.

2.3.1 Kit de desenvolvimento DE2

Inicialmente foi utilizado o kit de desenvolvimento DE2 fabricado pela TerasIC. Ela possui uma FPGA Cyclone II da Altera que possui 33.216 elementos lógicos, 483.840 bits de memória RAM, 4 PLL e 35 multiplicadores internos.

Esta placa possui diversos periféricos o que torna esta placa uma ótima opção para o desenvolvimento de novos projetos assim como para fins educacionais. A Figura 5 ilustra a placa em questão.

O kit de desenvolvimento DE2 possui as seguintes características:

- FPGA Altera Cyclone II 2C35;
- Dispositivo de configuração serial da Altera (EPCS16) para Cyclone II 2C35;
- USB Blaster embutido na placa para programar a FPGA e usar como controle através de uma API;
- Modo JTAG e modo AS são suportados;
- 8Mbyte (1M x 4 x 16) SDRAM;
- 512K byte(256K X16) SRAM;
- Memória Flash 4Mbyte (atualizável para 4Mbyte);
- Soquete para cartão SD;
- 4 botões seletores;
- 18 chaves bipolares;
- 9 LED verdes;
- 18 LED vermelhos;
- Módulo LCD 16 x 2;
- Oscilador de 50 MHz e 27 MHz (do decodificador de TV) como fontes de *clock*;
- CODEC de áudio de 24-bit com qualidade de CD, entrada, saída e entrada para microfone;

- Conversor DA VGA (DAC triplo de alta velocidade de 10-bit) com conector de saída VGA;
- Decodificador de TV (NTSC/PAL) e entrada para TV;
- Controlador Ethernet 10/100 com soquete;
- Controlador USB Hospedeiro/Escravo com conectores do tipo A e B;
- Transmissor RS-232 e conector de 9 pinos;
- Conector PS/2 para mouse e teclado;
- Transmissor IrDA;
- Dois barramentos de expansão com 40 pinos.

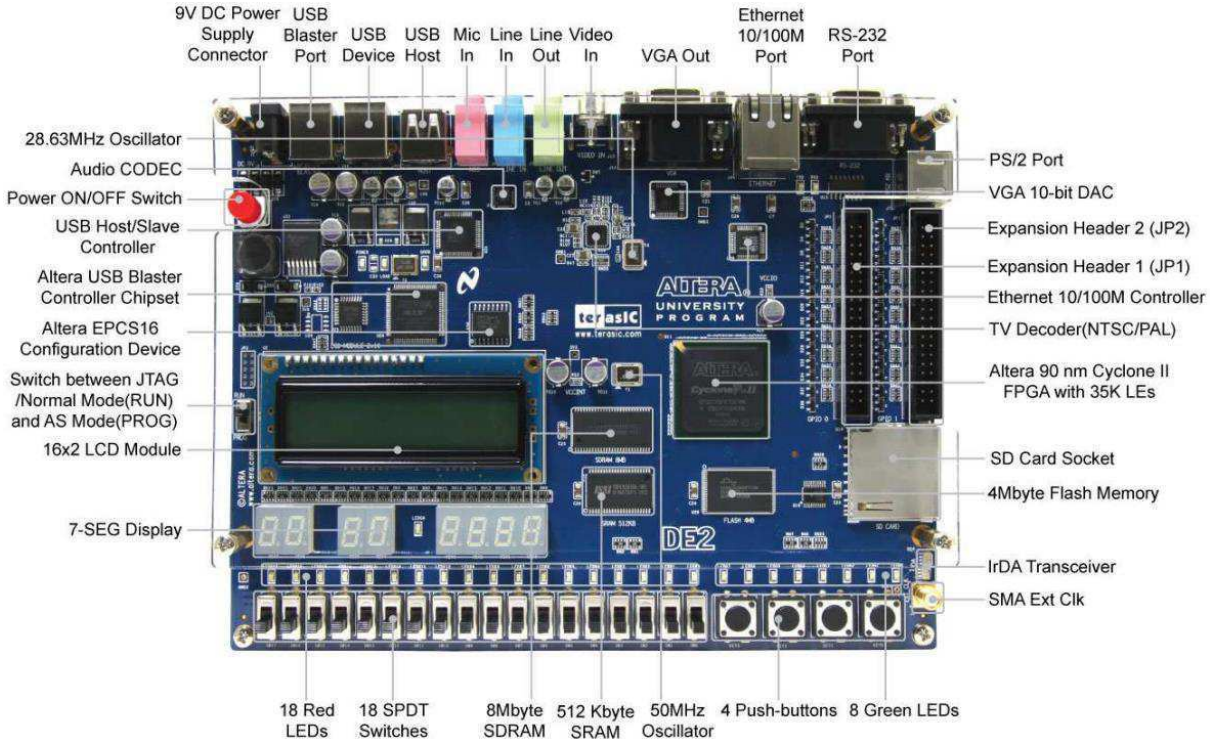

Figura 5: Descrição dos periféricos encontrados no kit de desenvolvimento DE2

2.3.2 Kit de desenvolvimento DE2-115

No decorrer do desenvolvimento do projeto, como será mostrado mais adiante, foi preciso utilizar uma FPGA com maior capacidade de memória e com um maior tamanho. Por isto, foi adotado o kit de desenvolvimento DE2-115 também fabricado pela TerasIC. Esta placa possui uma FPGA Cyclone IV E da Altera que possui 114.480 elementos lógicos, 3.981.312 bits de memória RAM, 4 PLL e 532 multiplicadores internos.

Esta placa possui diversos periféricos assim como a sua versão mais antiga (DE2), o que a torna também uma ótima opção tanto para fins educacionais como no desenvolvimento de novos projetos. A Figura 6 ilustra a placa em questão.

O kit de desenvolvimento DE2 possui as seguintes características:

- FPGA Altera Cyclone IV EP4CE115;
- Dispositivo de configuração serial da Altera (EPCS64) para Cyclone IV EP4C;
- USB Blaster embutido na placa para programar a FPGA e usar como controle através de uma API;
- Modo JTAG e modo AS são suportados;
- 128Mbyte (1M x 4 x 16) SDRAM;
- 2Mbyte(256K X16) SRAM;
- Memória Flash 8Mbyte;
- 32Kbit EEPROM
- Soquete para cartão SD;
- 4 botões seletores;
- 18 chaves bipolares;
- 9 LED verdes;
- 18 LED vermelhos;
- 8 displays de 7-segmentos;
- Módulo LCD 16 x 2;
- Três entradas para osciladores de 50MHz;
- Conectores SMA (entrada/saída de *clock* externo);
- Soquete para cartão SD;
    - Fornece acesso ao cartão SD pelo modo SPI e modo SD 4-bit;
- Duas portas Gigabit Ethernet;
    - 10/100/1000 Gigabit Ethernet integrado;
    - Suporta *IP cores* com *Ethernet* industrial;
- 172 pinos de alta velocidade para cartão Mezzanine (HSMC);
    - Padrões de E/S configuráveis (níveis de tensão: 3.3/2.5/1.8/1.5V);
- USB tipo A e B;
    - Fornece controladores para dispositivos hospedeiros e escravos com o USB 2.0;

  - Suporta transferência de dados a velocidade máxima e mínima;
  - Driver PC disponível;
- Porta com 40 pinos de expansão;
  - Padrões de E/S configuráveis (níveis de tensão: 3.3/2.5/1.8/1.5V);
- Conector de saída VGA;
  - Conversor DA VGA (DAC triplo de alta velocidade);
- Conector serial DB-9;
  - Porta RS232 com controle de fluxo;
- Conector PS/2;
  - Conector PS/2 para mouse ou teclado;
- Controle remote;
  - Modulo receptor infra-vermelho;
- Conector de entrada para TV;
  - Decodificador de TV (NTSC/PAL/SECAM);
- Alimentação;
  - Entrada CC do computador;
  - Regulador chaveado e abaixador LM3150MH.

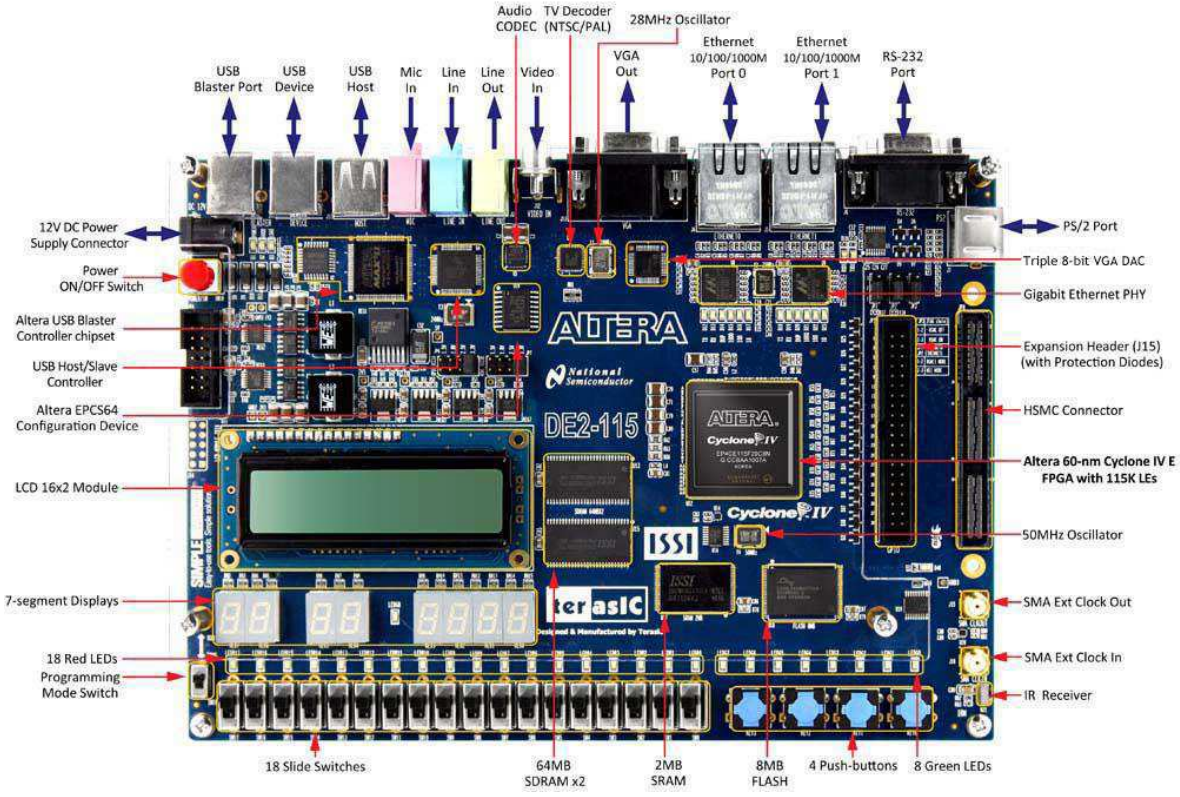

Figura 6: Descrição dos periféricos encontrados no kit de desenvolvimento DE2-115

2.4 Linguagem de descrição de *hardware* (HDL)

A Linguagem de Descrição de *Hardware* é um tipo de linguagem usada para descrever a estrutura, projeto e operação de circuitos eletrônicos, mais comumente, circuitos lógicos digitais.

Um *hardware* descrito por uma linguagem HDL possibilita uma maneira precisa e formal de projeto de um circuito eletrônico que permite a análise automatizada, simulação, e testes simulados do dispositivo. Também é possível através da compilação de uma HDL, verificar características em um nível de especificação mais baixo. Neste nível é possível verificar detalhes físicos e eletrônicos do circuito, como por exemplo, análises críticas de tempo e detalhes das máscaras utilizadas para criar um circuito integrado (IC).

A principal diferença entre as linguagens comuns de programação e as HDL é o fato de que a última inclui explicitamente uma noção de tempo em sua descrição. Alguns exemplos de HDL são: *Verilog*, *SystemVerilog* e VHDL. Por ser uma linguagem mais recente, o nível de descrição de *hardware* do *SystemVerilog* é mais alto do que as outras duas.

A sua configuração é geralmente especificada por uma linguagem de descrição de *hardware*, as mesmas que são utilizadas no desenvolvimento de um circuito integrado de aplicação específica (ASIC). Diagramas de circuitos eram comumente utilizados para especificar esta configuração, assim como eram utilizados também para ASIC, mas isto é raro atualmente.

Apesar de serem reaproveitadas neste projeto algumas partes de códigos descritas em VHDL e *Verilog* oriundas das FPU adquiridas, a sua grande maioria foi descrita em *SystemVerilog*. Algumas pequenas partes também foram descritas com *Verilog*.

2.5 Ferramentas de *software* utilizadas

Foram utilizadas várias ferramentas de *software* para desenvolver o projeto proposto. A primeira delas foi o simulador da Mentor Graphics adicionado à ferramenta da Altera com o nome de ModelSim Altera Starter Edition 10.1b. Com este *software* é possível simular códigos descritos em uma linguagem HDL, sejam

eles sintetizáveis ou não. Isto facilita bastante o desenvolvimento de um projeto em FPGA, pois é possível visualizar todo o comportamento interno do sistema em diferentes formatos em resposta aos estímulos aplicados. Outra razão para o uso desta ferramenta é o tempo de compilação que, no caso deste projeto, é quase instantâneo já que não é preciso fazer nenhuma etapa demorada como, por exemplo, o posicionamento e roteamento de elementos lógicos ou uma síntese como no caso da FPGA. Em contrapartida, uma desvantagem de simular ao invés de testar em FPGA é o tempo de simulação que aumenta proporcionalmente com a complexidade do projeto. A Figura 7 ilustra a interface gráfica desta ferramenta no decorrer de uma simulação deste projeto.

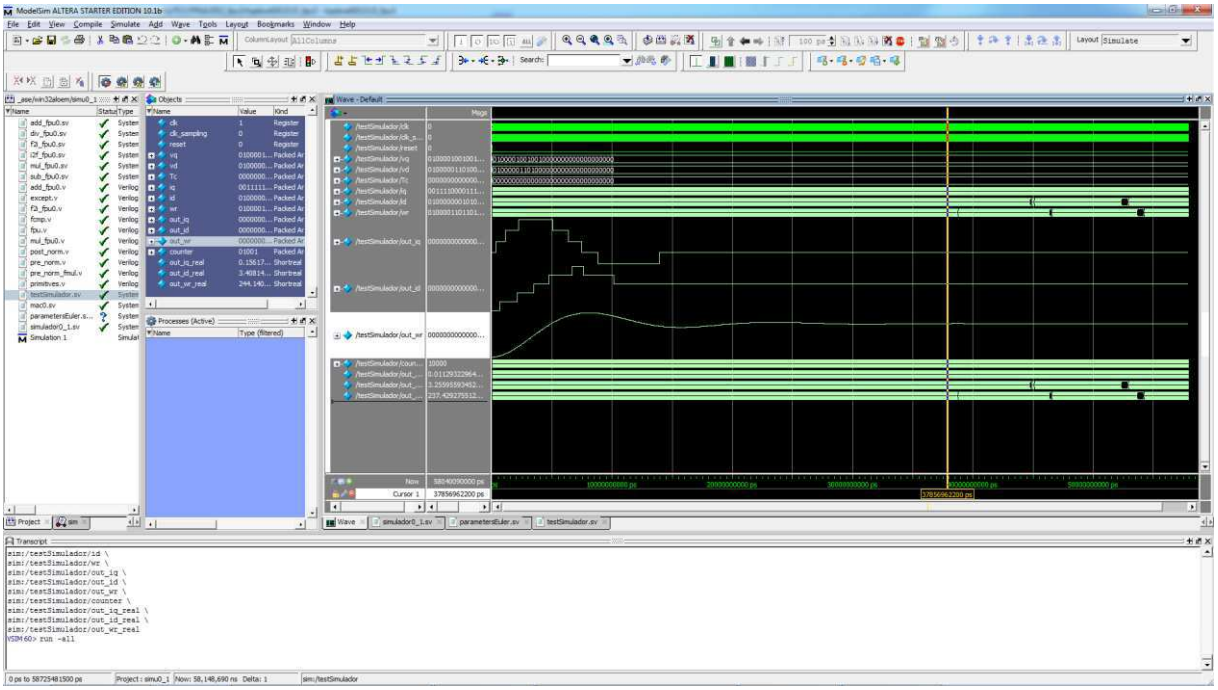

Figura 7: Interface gráfica do ModelSim Altera Starter Edition 10.1b

Outra ferramenta utilizada neste projeto foi o MATLAB R2013a de versão 8.1.0.604 da Mathworks. Este *software* foi utilizado principalmente para fazer uma análise detalhada dos dados obtidos nas simulações em FPGA, no ModelSim e Simulink. A Figura 8 ilustra a interface gráfica do MATLAB no decorrer de uma comparação de resultados adquiridos neste projeto.

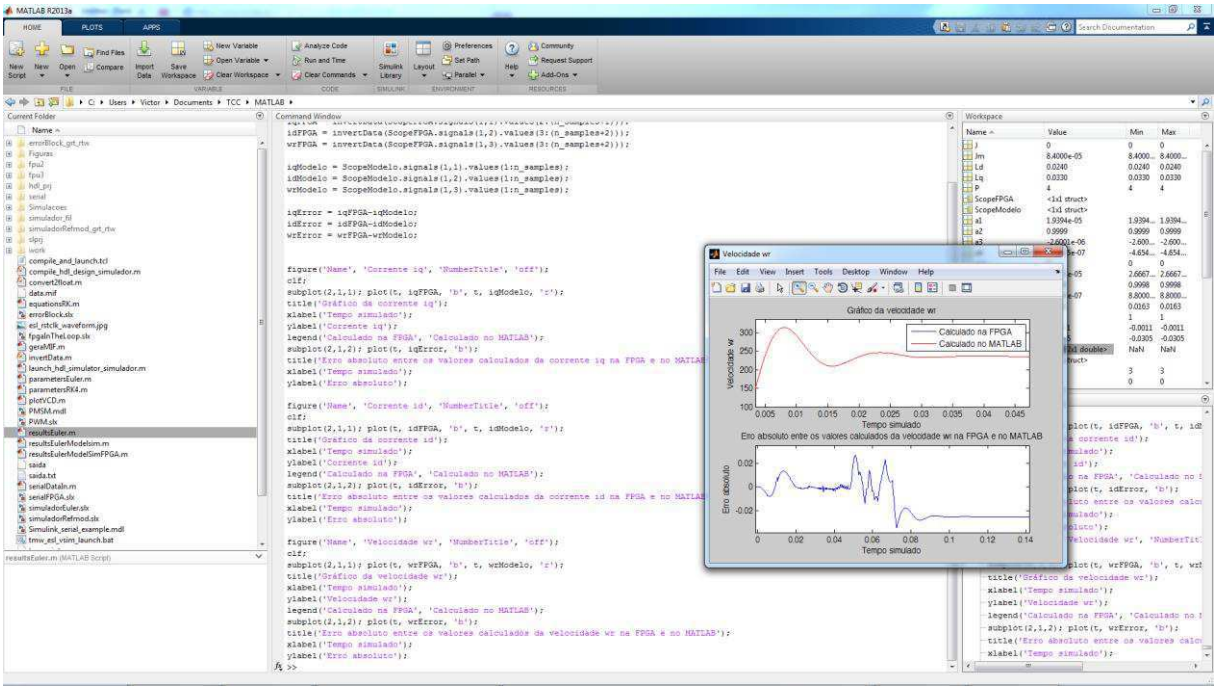

Figura 8: Interface gráfica do MATLAB R2013a (8.1.0.604)

Para embarcar o sistema na FPGA é preciso realizar as etapas de síntese, elaboração, posicionamento, roteamento e análise de tempo do código HDL. Sendo a Altera a fabricante das FPGA citadas na sessão 2.3, foi necessário utilizar o *software* Quartus II 64-bits Web-Edition de versão 13.0.0 da Altera. A Figura 9 ilustra a interface gráfica desta ferramenta no decorrer de uma compilação deste projeto.

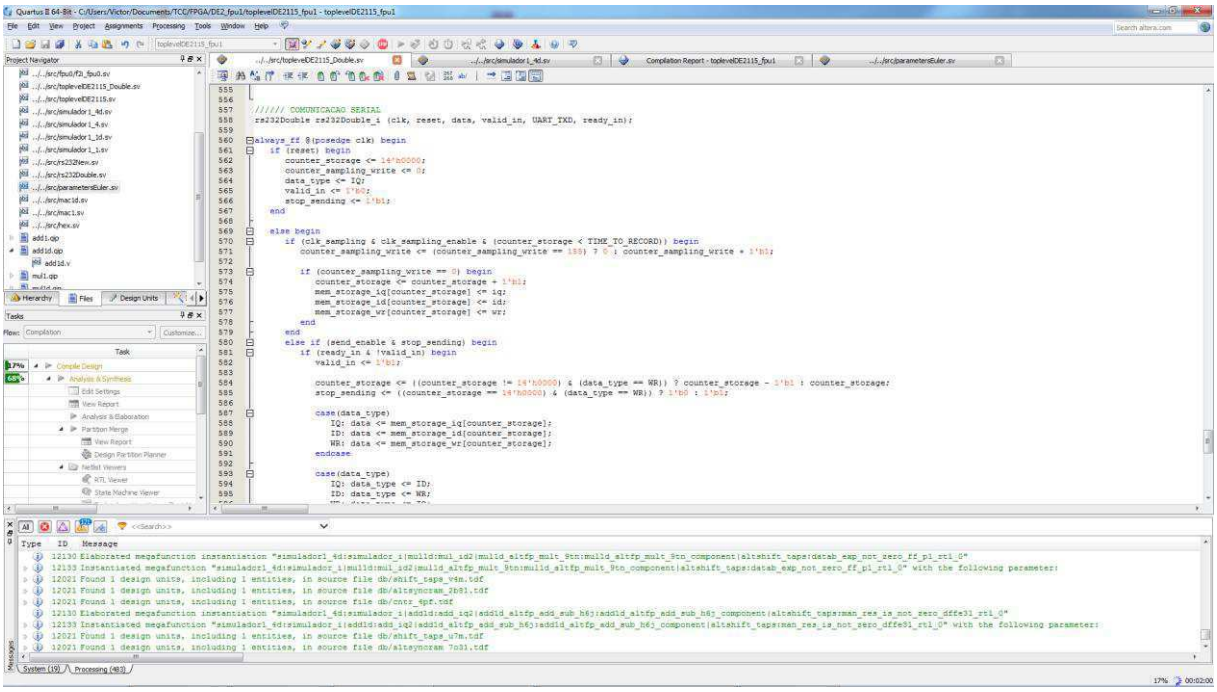

Figura 9: Interface gráfica do Quartus II 64-bits Version 13.0.0 Web Edition

Para depurar o sistema na FPGA, foi utilizado o SignalTap II Logic Analyzer também da Altera. Este *software* permite analisar os sinais internos do projeto na FPGA ao mesmo tempo em que ele está sendo executado. Ele funciona como um analisador lógico realizando esta comunicação através da interface JTAG que o kit de desenvolvimento e a FPGA possuem. A Figura 10 ilustra a interface gráfica desta ferramenta em meio a uma das simulações feitas neste projeto.

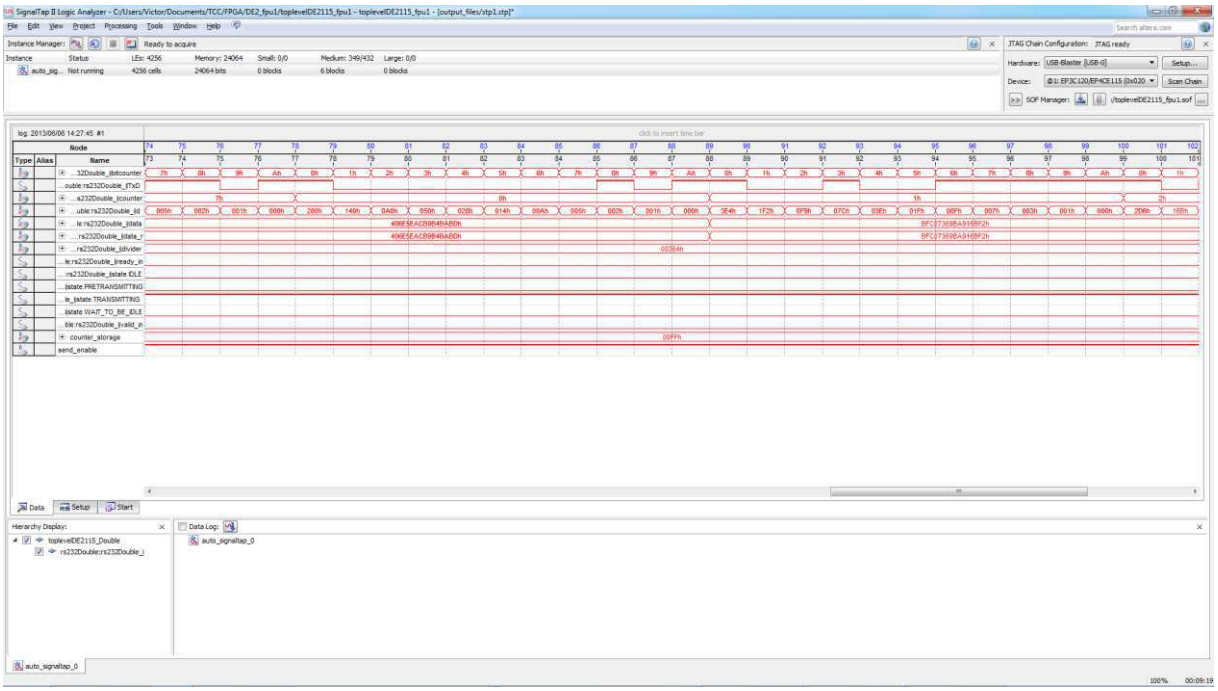

Figura 10: Interface gráfica do SignalTap II Logic Analyzer

Por último, foi utilizada a ferramenta da Mathworks que vem anexada ao MATLAB conhecida por Simulink com a versão 8.1 R2013a. Esta ferramenta proporcionou uma simulação adequada do modelo matemático proposto que será discutido mais adiante. A Figura 11 ilustra a interface gráfica deste *software* em meio a simulação das equações matemáticas deste projeto.

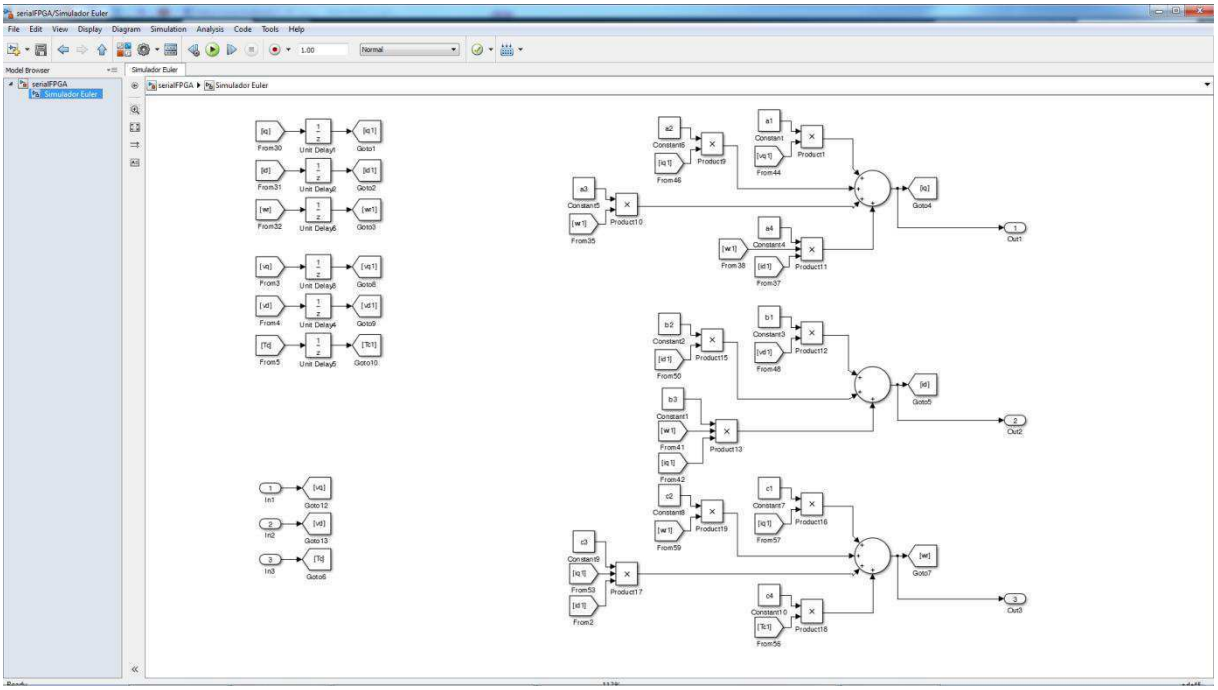

Figura 11: Interface gráfica do Simulink Versão 8.1 R2013a

3. Trabalho Realizado

Esta sessão mostra todo o trabalho realizado neste projeto assim como os resultados adquiridos ao final deste estudo. A sessão 3.1 introduz os objetivos do projeto e os passos a serem seguidos para alcança-los. Na sessão 3.2 é feita uma breve introdução a respeito do modelo matemático da máquina síncrona a imã permanente e na sessão 3.3 estas equações são discretizadas. A sessão 3.4 mostra as unidades de ponto flutuante utilizadas no projeto dos simuladores que foram elaborados, explicados e comparados na sessão 3.5. A sessão 3.6 mostra como se passou a aquisição dos dados da FPGA para um computador. Por fim, foi feita uma análise dos dados adquiridos no MATLAB e mostrados os resultados na sessão 3.7.

3.1 Introdução

Este projeto tem como principal objetivo utilizar uma unidade de ponto flutuante para resolver equações em tempo real. Para poder validar este projeto, foi preciso definir as equações que seriam testadas. Para isto, foi escolhida uma máquina síncrona a ímã permanente como planta do sistema. A sessão 3.2 explica brevemente o funcionamento e as equações no tempo contínuo deste sistema, enquanto que na sessão 3.3, as mesmas equações são definidas no tempo discreto. Em posse destas equações discretizadas, é possível embarcá-las em um dispositivo eletrônico para realizar o processamento necessário e calcular o resultado destas variáveis em tempo real.

3.2 Máquina síncrona a ímã permanente

A substituição do enrolamento de campo do rotor da máquina síncrona por um ímã permanente deu origem as máquinas síncronas a ímã permanente. Ela possui uma melhor relação na eficiência por volume, o que torna adequado seu uso em sistemas onde o espaço ocupado é crucial. Algumas aplicações comuns para este tipo de máquina são [4]:

- Aplicações com velocidade constante;
- Aplicações para operação como servomotor;
- Aplicações com velocidade variável.

Em relação à forma da força contra eletromotriz, a máquina síncrona pode ser classificada em: máquinas com força contra eletromotriz trapezoidal (*Brushless DC Machines* – BDCM) e máquinas com força contra eletromotriz senoidal (Permanent Magnet Synchronous Machines – PMSM). A modelagem matemática aqui mostrada é válida para a máquina síncrona PMSM.

É bastante comum no estudo das máquinas trifásicas a utilização de transformações de variáveis que permitam obter relações mais simples do que aquelas existentes entre as variáveis de fase. Uma forma conveniente para análise do comportamento da máquina PMSM é a representação das tensões, correntes e fluxos, em um referencial síncrono girante, conhecido por referencial *dq*, que está acoplado ao rotor da máquina [4].

Como este trabalho de conclusão de curso objetiva o uso de ponto flutuante em FPGA para simulações em tempo real, não será feita uma explanação detalhada da modelagem matemática desta máquina. Um maior detalhamento deste equacionamento e uma explicação melhor sobre o funcionamento da máquina pode ser encontrado na referência bibliográfica [4].

De forma resumida, a máquina síncrona PMSM possui o modelo matemático no tempo contínuo descrito pelas equações 1-4.

$$\begin{bmatrix} v_{sd} \\ v_{sq} \end{bmatrix} = \begin{bmatrix} r_s & -\omega_r L_{sq} \\ \omega_r L_{sd} & r_s \end{bmatrix} \begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix} + \begin{bmatrix} L_{sd} & 0 \\ 0 & L_{sq} \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix} + \omega_r \begin{bmatrix} 0 \\ \lambda_{pm} \end{bmatrix} \quad (1)$$

$$(J + J_m) \frac{d\omega_m}{dt} = T_t - T_c - f_w \omega_m \quad (2)$$

$$T_t = P[\lambda_{pm} i_{sq} + (L_{sd} - L_{sq}) i_{sd} i_{sq}] \quad (3)$$

$$\omega_r = P\omega_m \quad (4)$$

onde:

- v_{sd} é a tensão de eixo direto (d) dos enrolamentos do estator;
- v_{sq} é a tensão de eixo em quadratura (q) dos enrolamentos do estator;
- r_s é a resistência dos enrolamentos do estator;
- ω_r é a velocidade elétrica do rotor;
- ω_m é a frequência angular mecânica do rotor;
- L_{sd} é a indutância de eixo direto (d);

- L_{sq} é a indutância de eixo em quadratura (q);
- i_{sq} é a corrente de eixo direto (d) dos enrolamentos do estator;
- i_{sd} é a corrente de eixo em quadratura (q) dos enrolamentos do estator;
- λ_{pm} é igual a $\sqrt{\frac{3}{2}}$ vezes o valor de pico do fluxo produzido pelo ímã que enlaça os enrolamentos do estator;
- J é o momento de inércia do rotor da máquina;
- J_m é o momento de inércia da carga;
- T_t é o torque total desenvolvido pela máquina;
- T_c é o torque da carga;
- P é o número de polos da máquina;
- f_w é o coeficiente de atrito viscoso da máquina.

3.3 Equações no tempo discreto

Para poder realizar uma simulação em FPGA das equações 1-4, é preciso transcrevê-las do tempo contínuo para o tempo discreto através de algum método de discretização.

Neste projeto foi utilizado um método simples de discretização e bastante conhecido na literatura por Método de Euler ou *"Forward Difference"*. Este método se baseia em fazer uma aproximação no mapeamento entre o plano-z e o plano-s através de uma expansão em série truncada, ou seja:

$$z = e^{sh} \approx 1 + sh \tag{5}$$

onde h é o passo de cálculo da discretização. Manipulando a equação 5, obtém-se:

$$s = \frac{z-1}{h} \tag{6}$$

Esta aproximação também pode ser obtida ao representar a aproximação de uma derivada no tempo contínuo da seguinte forma:

$$\frac{dx(t)}{dt} \approx \frac{x(t)-x(t-h)}{h} = \frac{z-1}{h}x(t) \tag{7}$$

Considerando a transformada de Laplace da equação 7, obtém-se o resultado já apresentado anteriormente na equação 6.

Sendo assim, para realizar uma discretização através do Método de Euler, basta realizar a substituição apresentada pela equação 7. O primeiro passo para transformar as equações 1-4 em equações diferença é reescrevê-las da seguinte forma:

$$\frac{di_{sq}}{dt} = -\frac{\omega_r L_{sd}}{L_{sq}} i_{sd} - \frac{r_s}{L_{sq}} i_{sq} - \frac{\omega_r \lambda_{pm}}{L_{sq}} + \frac{v_{sq}}{L_{sq}} \tag{8}$$

$$\frac{di_{sd}}{dt} = -\frac{r_s}{L_{sd}} i_{sd} + \frac{\omega_r L_{sq}}{L_{sd}} i_{sq} + \frac{v_{sd}}{L_{sd}} \tag{9}$$

$$\frac{d\omega_r}{dt} = \frac{P^2 \lambda_{pm} i_{sq}}{(J+J_m)} + \frac{P^2 (L_{sd} - L_{sq}) i_{sd} i_{sq}}{(J+J_m)} - \frac{P T_c}{(J+J_m)} - \frac{f_w \omega_r}{(J+J_m)} \tag{10}$$

Fazendo a substituição da aproximação apresentada na equação 7 nas equações 8-10 e considerando o passo de cálculo *"h"*, as equações 1-4 apresentadas no tempo contínuo podem ser transformadas para as equações 11-13 no tempo discreto.

$$i_{sq}[k] = \frac{h}{L_{sq}} v_{sq}[k-1] + \left(1 - \frac{r_s h}{L_{sq}}\right) i_{sq}[k-1] - \frac{h \lambda_{pm}}{L_{sq}} \omega_r[k-1] - h \frac{L_{sd}}{L_{sq}} \omega_r[k-1] i_{sd}[k-1] \tag{11}$$

$$i_{sd}[k] = \frac{h}{L_{sd}} v_{sd}[k-1] + \left(1 - \frac{r_s h}{L_{sd}}\right) i_{sd}[k-1] + h \frac{L_{sq}}{L_{sd}} \omega_r[k-1] i_{sq}[k-1] \tag{12}$$

$$\omega_r[k] = \frac{hP^2 \lambda_{pm}}{(J+J_m)} i_{sq}[k-1] + \left(1 - \frac{f_w h}{(J+J_m)}\right) \omega_r[k-1] + \frac{hP^2}{(J+J_m)} \left(L_{sd} - L_{sq}\right) i_{sd}[k-1] i_{sq}[k-1] - \frac{hP}{(J+J_m)} T_c[k-1] \tag{13}$$

As únicas variáveis nestas três equações são i_{sq}, i_{sd} e ω_r. Sendo assim, podemos separar as constantes e reescrever as equações 11-13 da seguinte forma:

$$i_{sq}[k] = a_1 v_{sq}[k-1] + a_2 i_{sq}[k-1] + a_3 \omega_r[k-1] + a_4 \omega_r[k-1] i_{sd}[k-1] \tag{14}$$

$$i_{sd}[k] = b_1 v_{sd}[k-1] + b_2 i_{sd}[k-1] + b_3 \omega_r[k-1] i_{sq}[k-1] \tag{15}$$

$$\omega_r[k] = c_1 i_{sq}[k-1] + c_2 \omega_r[k-1] + c_3 i_{sd}[k-1] i_{sq}[k-1] + c_4 T_c[k-1] \tag{16}$$

onde:

$$a_1 = \frac{h}{L_{sq}} \qquad a_2 = \left(1 - \frac{r_s h}{L_{sq}}\right) \qquad a_3 = -\frac{h \lambda_{pm}}{L_{sq}} \qquad a_4 = -h \frac{L_{sd}}{L_{sq}}$$

$$b_1 = \frac{h}{L_{sd}} \qquad b_2 = \left(1 - \frac{r_s h}{L_{sd}}\right) \qquad b_3 = h\frac{L_{sq}}{L_{sd}}$$

$$c_1 = \frac{hP^2\lambda_{pm}}{(J+J_m)} \quad c_2 = \left(1 - \frac{f_w h}{(J+J_m)}\right) \quad c_3 = \frac{hP^2}{(J+J_m)}\left(L_{sd} - L_{sq}\right) \quad c_4 = -\frac{hP}{(J+J_m)}$$

Foram utilizadas as características da máquina da WEG SWA 40-16,6-30 para ser simulada. Sendo assim, as constantes da máquina são descritas na Tabela 4.

Nos cálculos aqui realizados, foi utilizado um f_w igual a zero. Conhecendo o fator V/krpm e a quantidade de pares de polos da máquina, ambos mostrados na Tabela 4, é possível calcular o λ_{pm} como sendo igual a 0,13407 Wb.

Ao substituir estas grandezas nas equações das constantes a_1, a_2, a_3, a_4, b_1, b_2, b_3, c_1, c_2, c_3 e c_4 citadas anteriormente, obtêm-se os valores apresentados na Tabela 5.

Tabela 4: Constantes da máquina WEG SWA 40-16,6-30

Grandeza	Valor
Potência Nominal	0,45 kW
r_s	6,187 Ω
L_{sq}	0,033 H
L_{sd}	0,024 H
V/krpm	56,16
Velocidade	3000 RPM
Corrente Nominal	2 A
Inércia	0,000084 kg/m^2
Pares de polos	4

A fim de obter resultados mais precisos nos cálculos das três variáveis em questão, é necessário utilizar um passo de cálculo menor ou igual a 1 µs. Portanto, para a maioria dos casos neste projeto, foi utilizado um passo de cálculo igual a 0,64 µs e em algumas exceções foi utilizado o passo de cálculo de 1 µs.

Tabela 5: Constantes utilizadas no modelo matemático

Constante	Resultado *(h = 0,64 µs)*	Resultado *(h = 1 µs)*
a_1	1,939393939393940 x 10^{-5}	3,030303030303030 x 10^{-5}
a_2	0,999880009696970	0,999812515151515
a_3	-2,600145454545454 x 10^{-6}	-4,062727272727272 x 10^{-6}
a_4	-4,654545454545455 x 10^{-7}	-7,272727272727272 x 10^{-7}
b_1	2,666666666666667 x 10^{-5}	4,166666666666667 x 10^{-5}
b_2	0,999835013333333	0,999742208333333
b_3	8,799999999999999 x 10^{-7}	1,375000000000000 x 10^{-6}
c_1	0,016343771428571	0,025537142857143
c_2	1	1
c_3	-0,001097142857143	-0,001714285714286
c_4	-0,030476190476190	-0,047619047619048

Em posse das constantes apresentadas na Tabela 5 e das equações diferença 11-13, é possível simular a máquina PMSM em questão em FPGA.

Para testar o Método de Euler aplicado a este sistema, foram implementadas as equações no modelo contínuo e no modelo discreto no Simulink. Após fazer a aquisição dos resultados da simulação foi feita uma comparação para obter o erro absoluto no MATLAB. Os resultados desta simulação podem ser analisados na Figura 12, Figura 13 e Figura 14. Foi aplicada uma tensão v_q constante em 50 V, v_d constante em 20 V e um torque de carga T_C nulo.

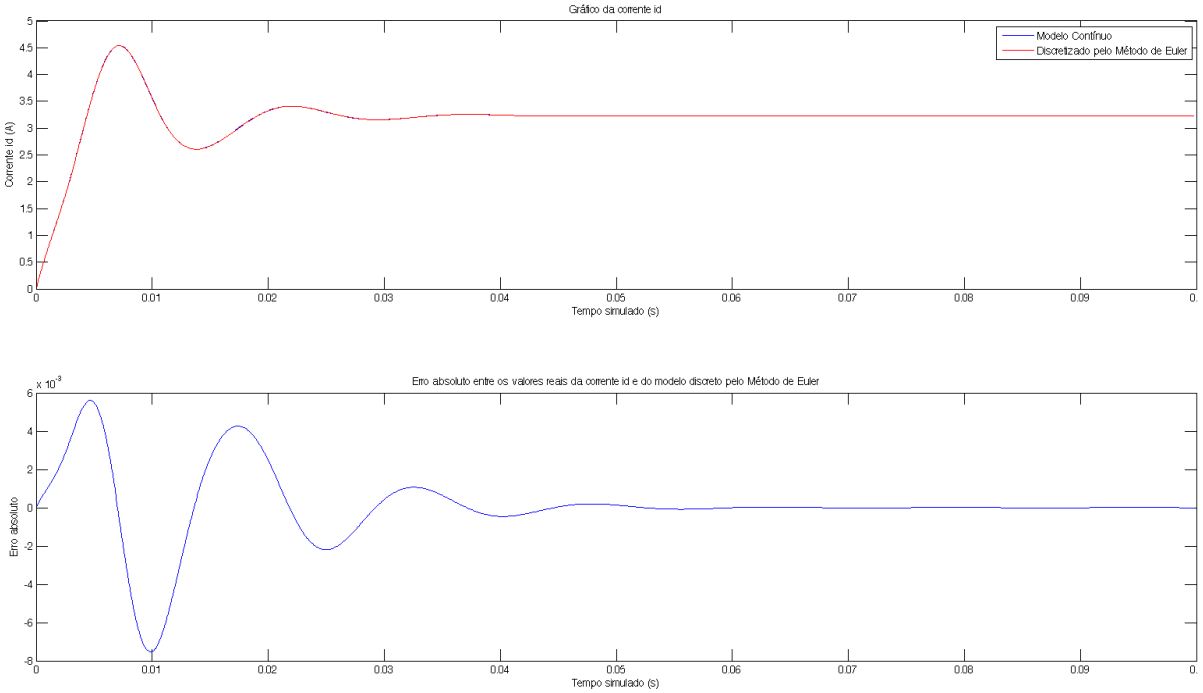

Figura 12: Comparação da corrente i_q do modelo contínuo e discreto no tempo

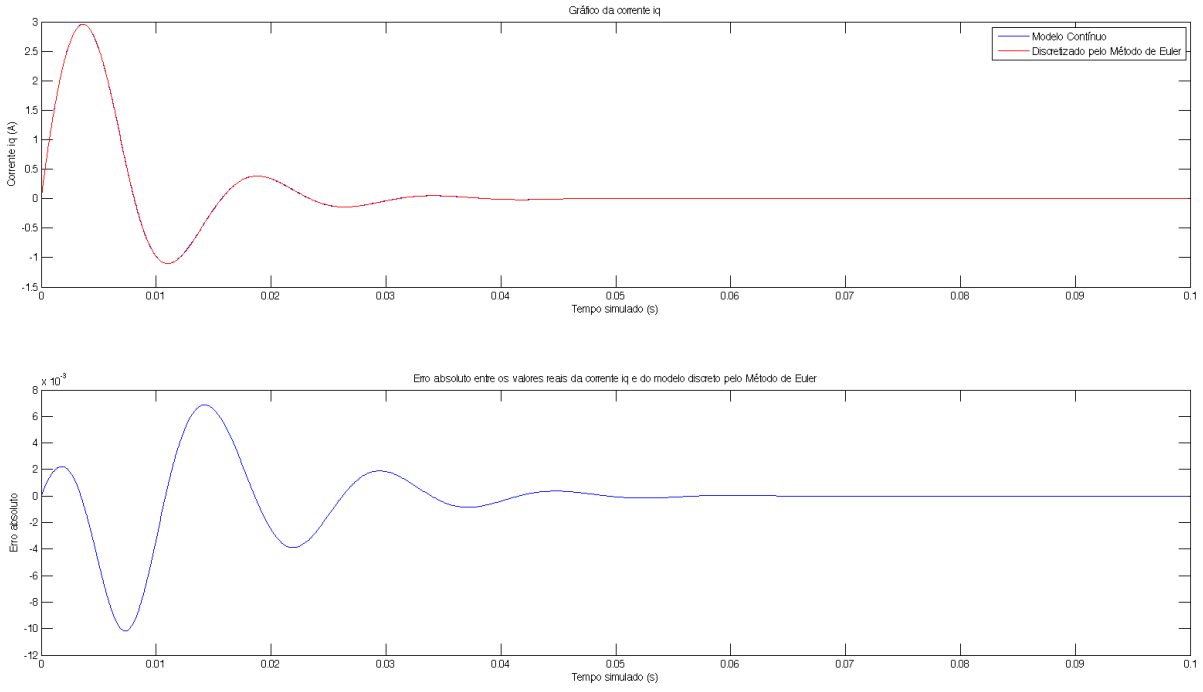

Figura 13: Comparação da corrente i_d do modelo contínuo e discreto no tempo

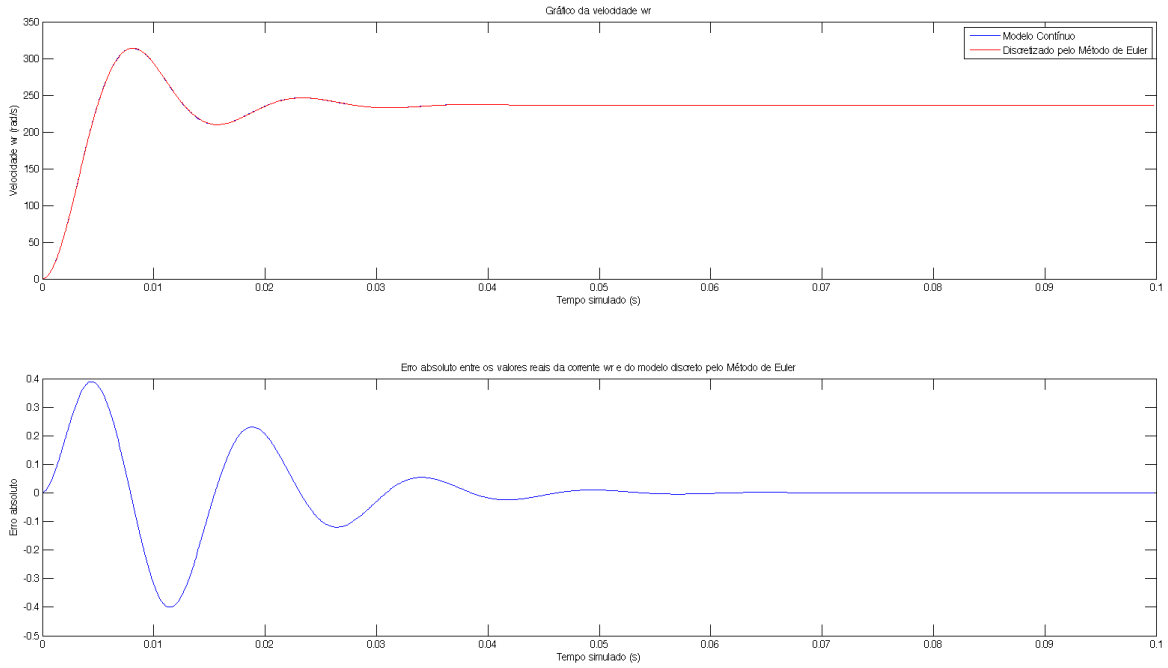

Figura 14: Comparação da velocidade angular ω_r do modelo contínuo e discreto no tempo

3.4 Unidades de ponto flutuante

No caso da representação numérica em ponto fixo, é possível facilmente realizar cálculos de forma fácil, rápida e sem a necessidade de muitos recursos. Contudo, para efetuar cálculos em ponto flutuante na FPGA, é necessário ter uma FPU que realize as operações básicas nesta representação numérica.

Foi feita uma busca por diversas FPU, livres ou comerciais, com diferentes características. Com isto, foram encontradas quatro FPU diferentes e assim foi realizado o mesmo trabalho e simulações para cada uma delas. Com isto, foi possível comparar o comportamento e algumas especificidades como latência e área da FPGA utilizada quando se utiliza cada uma delas. Foram encontradas três FPU de precisão simples e duas de precisão dupla.

Esta subseção 3.4 foca na explanação básica de cada uma das FPU utilizadas.

3.4.1 Primeira FPU

A primeira FPU foi encontrada no site da comunidade OpenCores (www.opencores.org) que é uma comunidade de código aberto para *hardware* onde

é possível encontrar em seu site diversos projetos de *hardware* livres ou *IP cores* como são conhecidos.

Esta FPU esta escrita em *Verilog*. O seu código é aberto e livre. Sendo assim, é possível ver e alterar o dispositivo. A Figura 15 apresenta a arquitetura desta FPU [6].

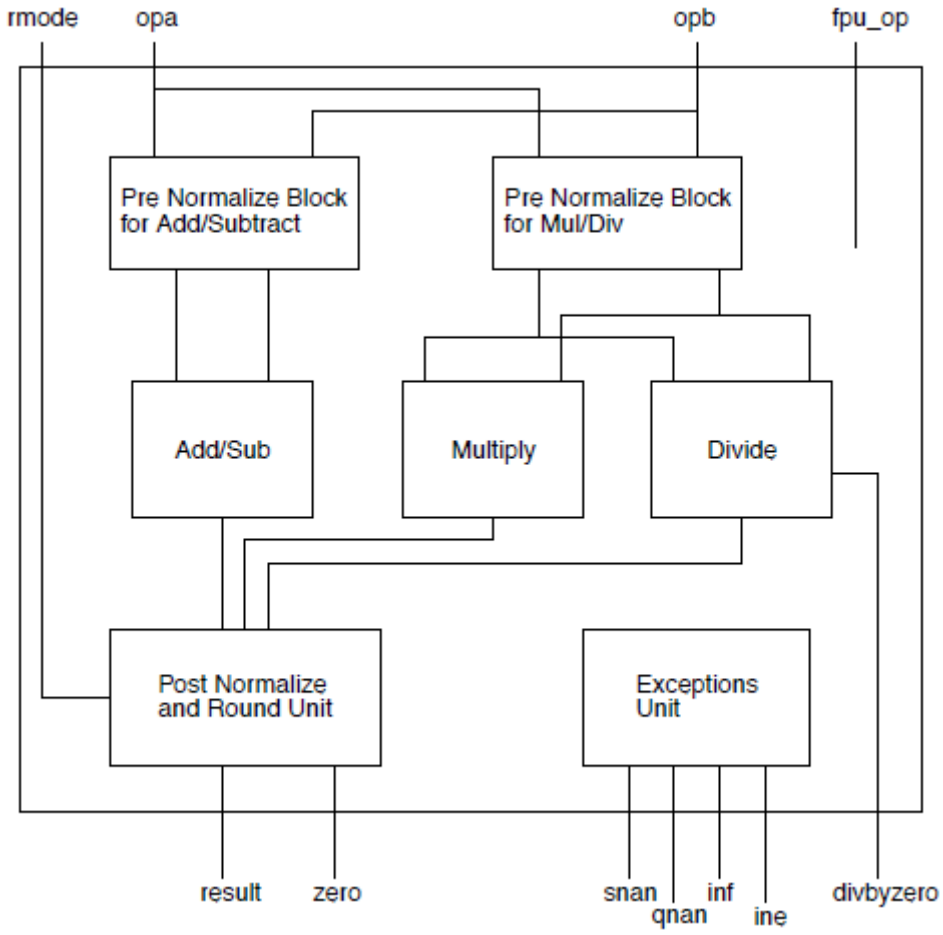

Figura 15: Arquitetura da primeira FPU

Esta FPU utiliza uma precisão de ponto flutuante simples. Este sistema é bastante simples de ser utilizado já que não possui nenhum protocolo de entrada ou saída. A Tabela 6 mostra as entradas e saídas deste *IP core*, a largura de cada sinal, direção e sua descrição.

Tabela 6: Entradas e saídas da primeira FPU [6]

Sinal	Largura	Direção	Descrição
clk	1	Entrada	*Clock* do sistema
rmode	2	Entrada	Modo de arredondamento
fpu_op	3	Entrada	Seleciona a operação
opa, opb	32	Entrada	Operandos A e B
out	32	Saída	Resultado da operação
snan	1	Saída	Sinaliza se algum operando é sNaN
qnan	1	Saída	Sinaliza se a saída é qNaN
inf	1	Saída	Sinaliza se a saída é INF
ine	1	Saída	Sinaliza se a saída é inexata
overflow	1	Saída	Sinaliza se ocorrer um *overflow*
underflow	1	Saída	Sinaliza se ocorrer um *underflow*
div_by_zero	1	Saída	Sinaliza se ocorrer uma divisão por zero
zero	1	Saída	Sinaliza se a saída for zero

Outro fator que torna esta FPU importante é a sua baixa latência para qualquer uma das operações em relação às outras FPU. A quantidade de recursos utilizados da FPGA é considerada normal quando comparado com os recursos utilizados pelas outras unidades de ponto flutuante. A frequência máxima aceitável deste *IP core* é uma desvantagem já que esta abaixo da média das outras FPU utilizadas neste projeto.

A Tabela 7 faz uma comparação para cada uma das operações entre três importantes fatores a serem analisados: recursos da FPGA utilizados, latência e frequência máxima. São apresentadas duas frequências máximas, pois a primeira foi calculada para o pior caso de um modelo lento com tensão 1200mV a 85°C e a segunda a 0°C. Estes cálculos da frequência máxima foram realizados pela ferramenta TimeQuest Timing Analyzer da Altera. A estimativa dos recursos utilizados foi feita com base na FPGA Cyclone IV EP4CE115F29C7 que se encontra no kit de desenvolvimento DE2-115 anteriormente citado pelo software *Quartus II 64-bits Web-Edition*. Os recursos utilizados aqui analisados foram o total de elementos lógicos (EL) e o número de multiplicadores de 9 bits embarcados (Mult.).

Tabela 7: Recursos da FPGA utilizados, latência e frequência máxima de cada operação da primeira FPU com precisão simples.

Operação	Recursos utilizados da FPGA	Latência	Frequência Máxima *(MHz)*
Soma	895 EL	4	48,82 / 53,18
Subtração	893 EL	4	46,04 / 50,1
Multiplicação	1.124 EL 7 Mult.	4	35,29 / 38,62
Divisão	5.329 EL	4	4,88 / 5,41
Conversão: Inteiro/Ponto flutuante	730 EL	4	63,22 / 69,0
Conversão: Ponto flutuante/Inteiro	788 EL	4	66,79 / 73,99

Outra grande vantagem desta FPU é o fato dela ter um *pipeline* interno. Com isto, não é necessário esperar uma latência de quatro ciclos de *clock* para poder aplicar outros operandos. Este bloco entrega resultados com a latência mencionada a cada ciclo de *clock*. Isto torna os cálculos mais rápidos caso sejam bem organizados. Na sessão 3.5 serão mostradas diferentes formas de organizar estes cálculos de forma a aproveitar este *pipeline*.

Esta FPU também apresenta uma especificidade que as outras não têm. Ela possui um bloco separado que permite fazer operações de comparação entre dois operandos em representação de ponto flutuante como, por exemplo, “maior que”, “menor que”, igualdade e etc. O Apêndice A mostra as entradas e saídas deste bloco comparador de operandos em ponto flutuante.

Uma especificação mais detalhada deste *IP core* pode ser encontrada na referência bibliográfica [6] que é a especificação desta FPU.

3.4.2 Segunda FPU

Esta FPU não é aberta e pertence a Altera. É possível utilizar esta unidade de ponto flutuante no *software* Quartus II através da ferramenta *MegaWizard Plugi-in Manager*. Esta ferramenta é um gerador passo-a-passo de *IP cores* pré-definidos.

Como esta ferramenta gera apenas um bloco, ela não realiza a geração de uma unidade de ponto flutuante completa. Ela realiza apenas a geração de blocos com ponto flutuante que realizam uma única operação, seja ela básica ou mais complexa.

A grande vantagem desta FPU é a flexibilidade em suas características. É possível escolher a latência dentre as opções existentes. Também é possível escolher quais as saídas adicionais que este bloco possuirá. Dentre outras opções, também é possível escolher a precisão da representação em ponto flutuante: simples, dupla ou simples-estendida.

No momento em que a função é gerada pela ferramenta, é perguntado se o usuário deseja obter uma interface descrita em *Verilog* ou VHDL, por exemplo. Vale salientar que apenas a interface é gerada. O bloco real é fechado e não pode ser visto e nem alterado a não ser pela própria fabricante. Sendo assim, não é possível mostrar a arquitetura desta unidade.

Esta FPU, assim como a primeira, não possui nenhum protocolo de entrada e nem de saída. Portanto, a comunicação e controle da operação deste bloco são bastante fáceis. Os sinais de sua interface são bastante similares aos da primeira. A Tabela 8 mostra as entradas e saídas da segunda FPU.

Tabela 8: Entradas e saídas da segunda FPU [17]

Sinal	Largura	Direção	Descrição
clk	1	Entrada	*Clock* do sistema
opa, opb	32 / 64	Entrada	Operandos A e B
out	32 / 64	Saída	Resultado da operação
nan	1	Saída	Sinaliza se a saída é NaN
overflow	1	Saída	Sinaliza se ocorrer um *overflow*
underflow	1	Saída	Sinaliza se ocorrer um *underflow*
div_by_zero	1	Saída	Sinaliza se ocorrer uma divisão por zero
zero	1	Saída	Sinaliza se a saída for zero

Outro fator que se destaca nesta FPU são os recursos utilizados na FPGA. Como ela foi projetada pela própria Altera, foi possível fazer um bom uso da área do dispositivo.

É possível definir a latência no momento da geração do *IP core* dentre um conjunto de valores já pré-definidos. Estes valores são maiores do que os da primeira FPU, embora sejam bem menores do que às outras, o que torna esta uma boa opção quando a latência é crucial para o projeto.

A frequência máxima aceitável é relativamente alta e é maior do que a frequência máxima permitida na primeira FPU.

A Tabela 9 faz uma comparação para cada uma das operações da segunda FPU com precisão simples entre três importantes fatores a serem analisados: recursos da FPGA utilizados, latência e frequência máxima. São feitas as mesmas considerações que foram feitas na sessão 3.4.1 com respeito à frequência máxima calculada e aos recursos utilizados.

Tabela 9: Recursos da FPGA utilizados, latência e frequência máxima de cada operação da segunda FPU com precisão simples.

Operação	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Soma	792 EL	7	152,79 / 168,98
Subtração	800 EL	7	151,22 / 166,97
Multiplicação	260 EL 7 Mult.	5	177,21 / 197,01
Divisão	304 EL 16 Mult.	6	110,46 / 123,03
Raiz quadrada	770 EL	16	130,82 / 145,29
Conversão: Inteiro/Ponto flutuante	306 EL	6	250
Conversão: Ponto flutuante/Inteiro	427 EL	6	233,32 / 254,07

Este bloco não possui *pipeline* e é necessário esperar a latência mencionada para poder aplicar outros operandos. A Tabela 10 compara os mesmos fatores da mesma FPU, porém utilizando uma precisão dupla.

Tabela 10: Recursos da FPGA utilizados, latência e frequência máxima de cada operação da segunda FPU com precisão dupla.

Operação	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Soma	1.683 EL	7	124,58 / 137,61
Subtração	1.681 EL	7	127,82 / 141,12
Multiplicação	678 EL 18 Mult.	5	104,01 / 116,75
Divisão	1.243 EL 44 Mult.	10	85,88 / 95,27
Raiz quadrada	3.311 EL	30	87,05 / 96,7
Conversão: Inteiro/Ponto flutuante	626 EL	6	250
Conversão: Ponto flutuante/Inteiro	914 EL	6	164,42 / 186,53

Uma especificação mais detalhada deste *IP core* pode ser encontrada na referência bibliográfica [17] que é a especificação desta FPU.

3.4.3 Terceira FPU

A terceira FPU é de precisão simples e assim como a primeira, também foi encontrada no site do projeto *OpenCores*. Ela está descrita em VHDL e seu código também é livre e aberto. A arquitetura desta FPU é ilustrada na Figura 16.

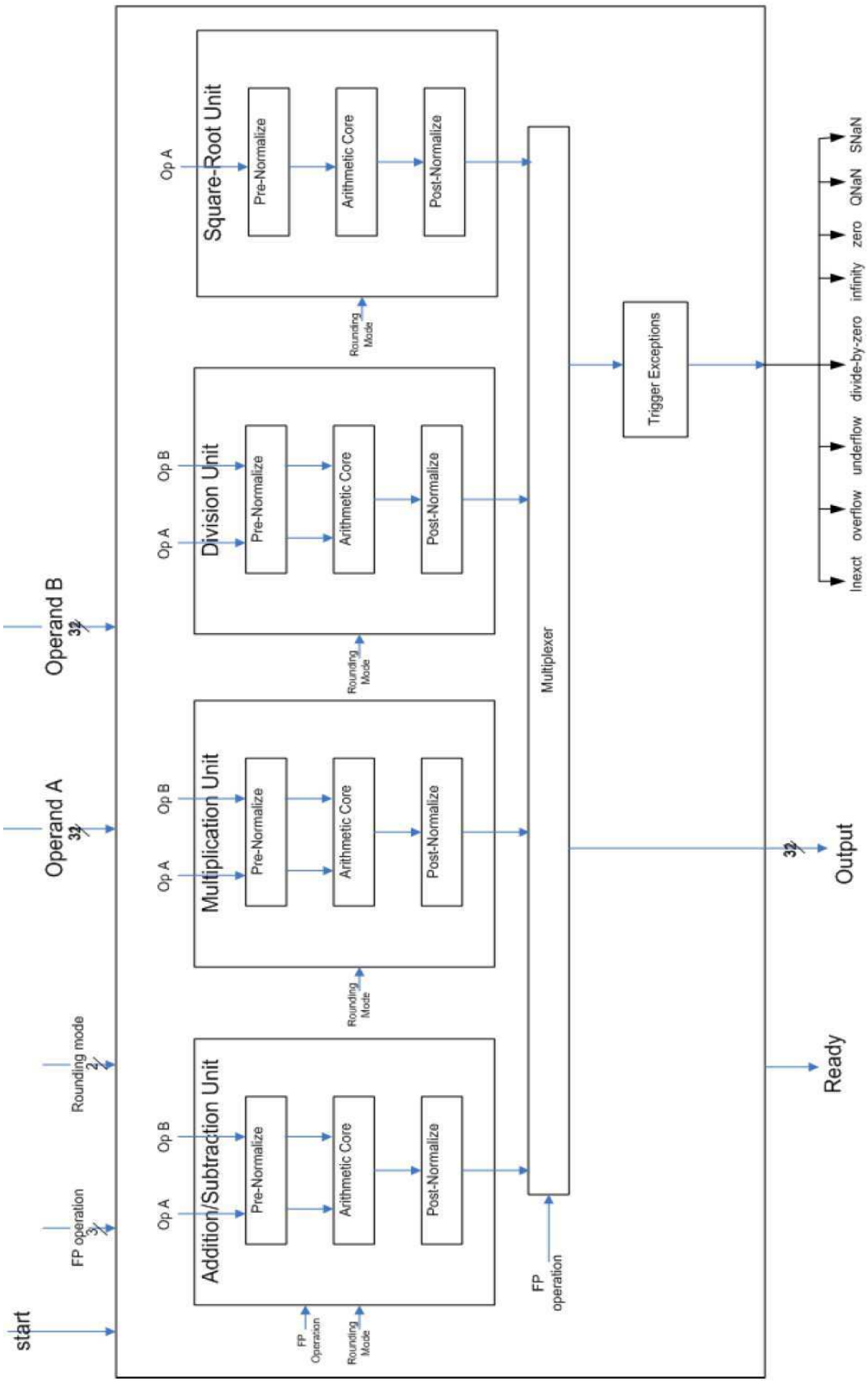

Figura 16: Arquitetura da terceira FPU

A Tabela 11 mostra todas as entradas e saídas deste *IP core*. Este bloco possui uma interface parecida com as outras duas FPU apresentadas aqui, embora ela tenha um simples protocolo de entrada e saída. O sinal de entrada *start_o* e o de saída *ready_o* representam esta interface.

Tabela 11: Entradas e saídas da terceira FPU [18]

Sinal	Largura	Direção	Descrição
clk_i	1	Entrada	*Clock* do sistema
rmode_i	2	Entrada	Modo de arredondamento
fpu_op_i	3	Entrada	Seleciona a operação
opa_i, opb_i	32	Entrada	Operandos A e B
start_i	1	Entrada	Inicia o cálculo
output_o	32	Saída	Resultado da operação
ready_o	1	Saída	Sinaliza o fim do cálculo
snan_o	1	Saída	Sinaliza se algum operando é sNaN
qnan_o	1	Saída	Sinaliza se a saída é qNaN
inf_o	1	Saída	Sinaliza se a saída é INF
ine_o	1	Saída	Sinaliza se a saída é inexata
overflow_o	1	Saída	Sinaliza se ocorrer um *overflow*
underflow_o	1	Saída	Sinaliza se ocorrer um *underflow*
div_zero_o	1	Saída	Sinaliza se ocorrer uma divisão por zero
zero_o	1	Saída	Sinaliza se a saída for zero

Esta unidade de ponto flutuante normalmente utiliza maiores recursos da FPGA que as outras duas aqui apresentadas. De mesmo modo, a latência também representa uma desvantagem no uso deste bloco. A frequência máxima atingida pelos cálculos do *TimeQuest* são relativamente altas quando comparadas aos outros resultados anteriormente obtidos para a primeira e segunda FPU.

A Tabela 12 faz uma comparação para cada uma das operações da segunda FPU com precisão simples entre três importantes fatores a serem analisados: recursos da FPGA utilizados, latência e frequência máxima. São feitas as mesmas considerações que foram feitas na sessão 3.4.1 com respeito à frequência máxima calculada e aos recursos utilizados.

Tabela 12: Recursos da FPGA utilizados, latência e frequência máxima de cada operação da terceira FPU com precisão simples.

Operação	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Soma	1.029 EL	7	121,82 / 132,84
Subtração	1.013 EL	7	114,35 / 126,02
Multiplicação	1.597 EL 4 Mult.	12	131,98 / 144,99
Divisão	1.210 EL	35	134,66 / 148,63
Raiz Quadrada	1.240 EL	35	88,32 / 98,26

Assim como a segunda FPU, este bloco não possui *pipeline* e é necessário esperar a latência mencionada para poder aplicar outros operandos.

Uma especificação mais detalhada deste *IP core* pode ser encontrada na referência bibliográfica [18] que é a especificação desta FPU.

3.4.4 Quarta FPU

Assim como a primeira e a terceira, a quarta FPU também foi encontrada no projeto OpenCores. Ela é uma unidade de ponto flutuante com precisão dupla. O seu código foi descrita em *Verilog* e é livre e aberto.

Assim como a terceira, ela possui um simples protocolo de entrada e saída com os sinais *enable* e *ready*. A Tabela 13 apresenta os sinais de entrada e saída deste *IP core*.

Por ser uma FPU de precisão dupla, ela utiliza mais recursos da FPGA do que as outras aqui apresentadas. Sua latência também é um ponto fraco por ser longo comparado até mesmo com as outras de precisão dupla. Contudo, a vantagem desta FPU é a frequência máxima aceitável sendo mais elevada do que as outras.

Tabela 13: Entradas e saídas da quarta FPU [19]

Sinal	Largura	Direção	Descrição
clk	1	Entrada	*Clock* do sistema
rst	1	Entrada	Sinal de *reset*
enable	1	Entrada	Inicia o cálculo
rmode	2	Entrada	Modo de arredondamento
fpu_op	3	Entrada	Seleciona a operação
opa, opb	64	Entrada	Operandos A e B
out	64	Saída	Resultado da operação
ready	1	Saída	Sinaliza o fim do cálculo
inexact	1	Saída	Sinaliza se a saída é inexata
overflow	1	Saída	Sinaliza se ocorrer um *overflow*
underflow	1	Saída	Sinaliza se ocorrer um *underflow*
exception	1	Saída	Sinaliza algumas exceções no resultado
invalid	1	Saída	Sinaliza uma saída invalida

A Tabela 14 faz uma comparação para cada uma das operações da quarta FPU com precisão simples entre três importantes fatores a serem analisados: recursos da FPGA utilizados, latência e frequência máxima. São feitas as mesmas considerações que foram feitas na sessão 3.4.1 com respeito à frequência máxima calculada e aos recursos utilizados.

Esta FPU possui um multiplicador, um somador e um subtrator separados que têm *pipeline* assim como o primeiro. Estes blocos foram utilizados nos casos dos simuladores que fazem uso desta característica.

Uma especificação mais detalhada deste *IP core* pode ser encontrada na referência bibliográfica [19] que é a especificação desta FPU.

Tabela 14: Recursos da FPGA utilizados, latência e frequência máxima de cada operação da quarta FPU de precisão dupla.

Operação	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Soma	3.696 EL	24	144,95 / 156,54
Subtração	3.701 EL	21	153,35 / 166,69
Multiplicação	4.732 EL 28 Mult.	28	136,13 / 146,43
Divisão	3.683 EL	71	133,33 / 144,78
Soma (com *pipeline*)	17.105 EL	25	71,68 / 79,94
Subtração (com *pipeline*)	17.105 EL	22	71,52 / 82,58
Multiplicação (com *pipeline*)	16.451 EL 28 Mult.	22	67,85 / 74,56

3.4.5 Resumo e comparação das FPU

Neste projeto foram utilizadas apenas as partes de adição e multiplicação das unidades de ponto flutuante. Normalmente a divisão é bastante complexa e muitas vezes, ocupa mais espaço, tem uma latência maior e tem uma frequência máxima inferior em frente às outras operações. Portanto, como os cálculos eram sempre divisões e multiplicações por constantes, foi preferido fazer sempre a multiplicação pelo inverso ao invés de dividir.

A Tabela 15 ilustra uma comparação entre as operações de adição de todas as unidades de ponto flutuante apresentadas anteriormente. A Tabela 16 faz a mesma comparação para a operação de multiplicação. Esta comparação é feita em termos de recursos da FPGA utilizados, latência e frequência máxima de operação.

Tabela 15: Recursos da FPGA utilizados, latência e frequência máxima da parte de adição das FPU apresentadas.

FPU	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Soma *(FPU 1)*	895 EL	4	48,82 / 53,18
Soma *(FPU 2) (precisão simples)*	792 EL	7	152,79 / 168,98
Soma *(FPU 2) (precisão dupla)*	1.683 EL	7	124,58 / 137,61
Soma *(FPU 3)*	1.029 EL	7	121,82 / 132,84
Soma *(FPU 4) (sem pipeline)*	3.696 EL	24	144,95 / 156,54
Soma *(FPU 4) (com pipeline)*	17.105 EL	25	71,68 / 79,94

Tabela 16: Recursos da FPGA utilizados, latência e frequência máxima da parte de multiplicação das FPU apresentadas.

FPU	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Multiplicação *(FPU 1)*	1.124 EL 7 Mult.	4	35,29 / 38,62
Multiplicação *(FPU 2) (precisão simples)*	260 EL 7 Mult.	5	177,21 / 197,01
Multiplicação *(FPU 2) (precisão dupla)*	678 EL 18 Mult.	5	104,01 / 116,75
Multiplicação *(FPU 3)*	1.597 EL 4 Mult.	12	131,98 / 144,99
Multiplicação *(FPU 4) (sem pipeline)*	4.732 EL 28 Mult.	28	136,13 / 146,43
Multiplicação *(FPU 4) (com pipeline)*	16.451 EL 28 Mult.	22	67,85 / 74,56

3.5 Elaboração dos simuladores

Para simular em tempo real na FPGA o modelo matemático da máquina PMSM, descrita pelas equações 14-16, foram propostos quatro diferentes simuladores onde cada um deles foi descrito de uma maneira diferente, variando a quantidade de multiplicadores e somadores de ponto flutuante utilizados e a ordem com que os cálculos foram realizados.

Esta sessão visa a explanação detalhada de cada um destes simuladores que foram descritos em *SystemVerilog*.

3.5.1 Simulador tipo 1

Este primeiro simulador se baseia no uso de um acumulador multiplicativo (MAC). O MAC multiplica dois operandos que são colocados em suas entradas e depois incrementa o resultado da multiplicação em um acumulador interno. Este bloco funciona com o auxílio de dois sinais de controle que habilita o início do cálculo e outro que zera o acumulador.

Um diagrama de blocos de um MAC genérico é apresentado na Figura 17.

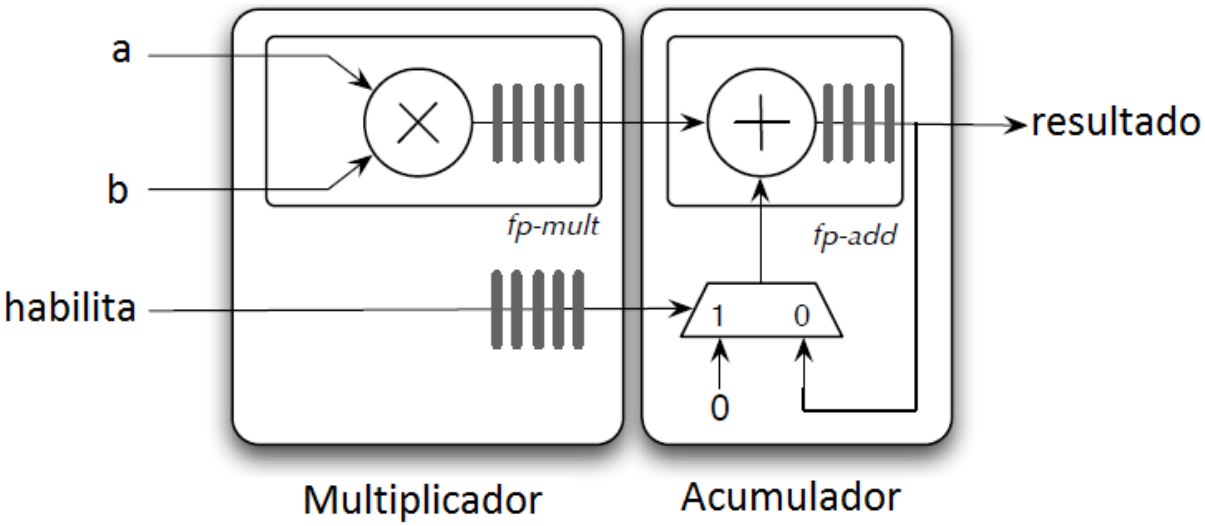

Figura 17: Representação do acumulador multiplicativo (MAC) [2]

Primeiramente foi projetado um MAC para cada uma das FPU encontradas utilizando apenas os multiplicadores e somadores de cada uma delas. A Tabela 17 mostra uma comparação feita entre todos os MAC projetados com respeito aos recursos da FPGA utilizados, latência e frequência máxima suportada.

Tabela 17: Recursos da FPGA utilizados, latência e frequência máxima de todos os MAC projetados.

FPU	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Primeira FPU	2.041 EL 7 Mult.	9	38,11 / 41,63
Segunda FPU (Precisão simples)	1.209 EL 7 Mult.	13	112,52 / 123,95
Segunda FPU (Precisão dupla)	2.648 EL 18 Mult.	13	92,72 / 102,03
Terceira FPU	2.503 EL 4 Mult.	20	108,01 / 119,09
Quarta FPU (Sem *pipeline)*	8.410 EL 28 Mult.	45	137,87 / 149,75
Quarta FPU (Com *pipeline)*	7.730 EL 28 Mult.	45	142,94 / 156,3

No caso da segunda FPU foi feito um MAC para a precisão simples e outro para a precisão dupla. Para a quarta FPU foi feito um MAC utilizando os blocos normais de multiplicação e adição e outro utilizando os blocos alternativos que acompanham esta FPU que possuem *pipeline*. Os projetos dos MAC foram simulados com o ModelSim.

Após projetar cada um dos MAC, foi feito o primeiro simulador das equações 14-16. Como cada equação é simplesmente a soma de três ou quatro termos com multiplicações em cada um deles, utilizou-se um MAC para calcular cada variável. Três multiplicadores adicionais foram acrescentados apenas para fornecer as multiplicações dos acoplamentos das variáveis i_q, i_d e ω_r. A Figura 18 ilustra os blocos utilizados para implementar este simulador.

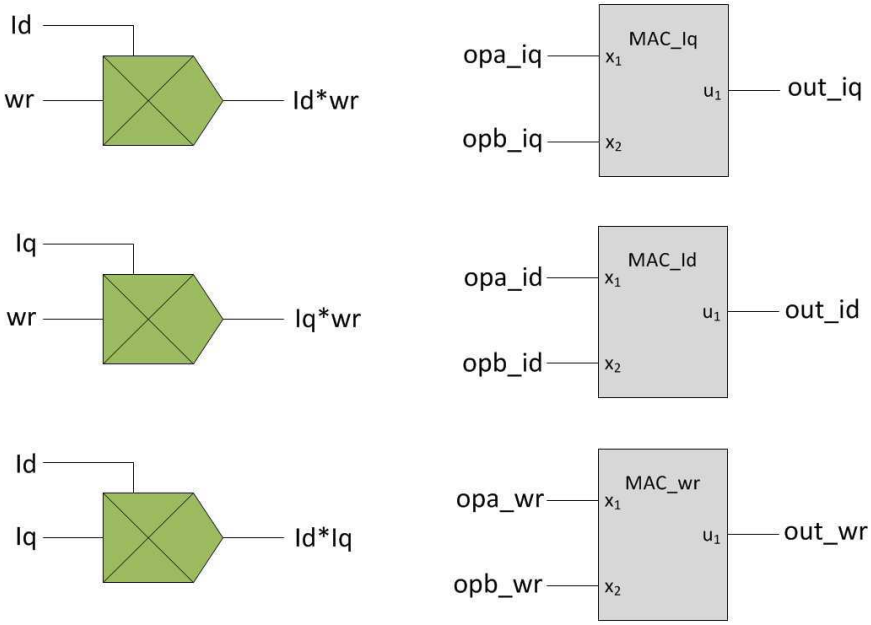

Figura 18: Blocos utilizados no simulador tipo 1

Posteriormente, estes blocos são controlados por uma máquina de estados que alimenta as suas entradas com os operandos corretos no tempo devido. A Figura 19 ilustra esta máquina de estados.

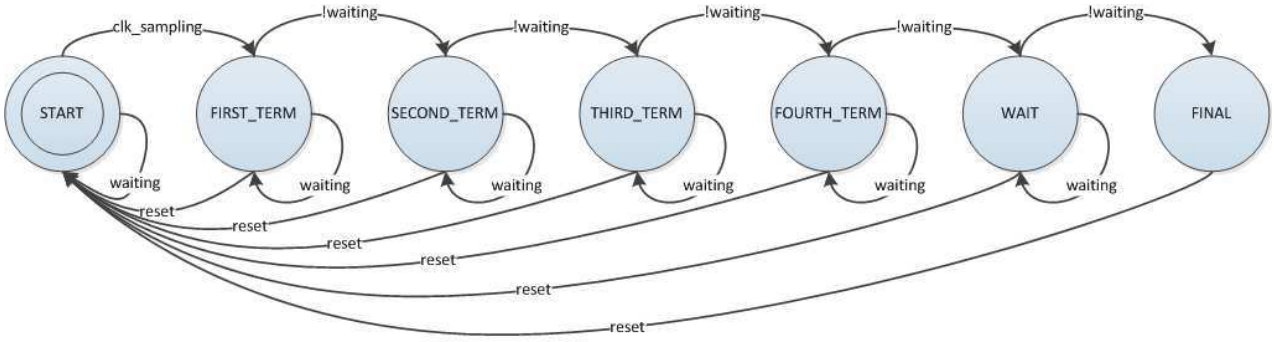

Figura 19: Estados do simulador tipo 1

A Figura 20 ilustra o pseudocódigo de cada um destes estados. Basicamente, cada estado vai alimentar os MAC com os operandos de cada termo e o valor que o acumulador guarda no estado final será a soma de todos estes termos de cada equação. No estado final também é feita a multiplicação do acoplamento das variáveis para o próximo passo de cálculo. Também são zerados os acumuladores após eles serem passados para a saída.

FIRST_TERM:
Operador A do MAC i_q <= a_1
Operador B do MAC i_q <= $v_q[k-1]$
Operador A do MAC i_d <= b_1
Operador B do MAC i_d <= $v_d[k-1]$
Operador A do MAC ω_r <= c_1
Operador B do MAC ω_r <= $i_q[k-1]$
Espera latência da multiplicação
SECOND_TERM:
Operador A do MAC i_q <= a_2
Operador B do MAC i_q <= $i_q[k-1]$
Operador A do MAC i_d <= b_2
Operador B do MAC i_d <= $i_d[k-1]$
Operador A do MAC ω_r <= c_2
Operador B do MAC ω_r <= $\omega_r[k-1]$
Espera latência da multiplicação
THIRD_TERM:
Operador A do MAC i_q <= a_3
Operador B do MAC i_q <= $\omega_r[k-1]$
Operador A do MAC i_d <= b_3
Operador B do MAC i_d <= $i_q[k-1] * \omega_r[k-1]$
Operador A do MAC ω_r <= c_3
Operador B MAC ω_r <= $i_q[k-1] * i_d[k-1]$
Espera latência da multiplicação
FOURTH_TERM:
Operador A do MAC i_q <= a_4
Operador B do MAC i_q <= $i_d[k-1] * \omega_r[k-1]$
Operador A do MAC ω_r <= c_4
Operador B do MAC ω_r <= $T_C[k-1]$
Espera latência da multiplicação
WAIT:
Espera a latência do MAC
FINAL:
Operador A do multiplicador $i_q * i_d$ <= $i_q[k-1]$
Operador B do multiplicador $i_q * i_d$ <= $i_d[k-1]$
Operador A do multiplicador $i_q * \omega_r$ <= $i_q[k-1]$
Operador B do multiplicador $i_q * \omega_r$ <= $\omega_r[k-1]$
Operador A do multiplicador $i_d * \omega_r$ <= $\omega_r[k-1]$
Operador B do multiplicador $i_d * \omega_r$ <= $i_q[k-1]$
Zera todos os MAC
Coloca iq, id e ω_r nas respectivas saídas

Figura 20: Pseudocódigo dos estados do simulador tipo 1

Este simulador foi implementado para todas as quatro FPU. A segunda FPU possui precisão simples e dupla e, portanto foi feito um simulador para cada uma delas. No caso da quarta FPU, existe a possibilidade de utilizar multiplicadores e somadores alternativos com *pipeline* que acompanham esta FPU. Portanto, mesmo sem fazer o uso do pipeline neste simulador, foi feito um para ambos os casos. A Tabela 18 mostra uma comparação realizada para as implementações deste simulador ao utilizar cada um das FPU mostradas na sessão 3.4.

Tabela 18: Recursos da FPGA utilizados, latência e frequência máxima do simulador tipo 1 para cada FPU

FPU	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Primeira FPU	9.260 EL 42 Mult.	30	36,66 / 40,27
Segunda FPU (Precisão simples)	4.586 EL 42 Mult.	45	111.25 / 123.11
Segunda FPU (Precisão dupla)	10.241 EL 108 Mult.	45	92,16 / 101,62
Terceira FPU	11.375 EL 24 Mult.	78	104,94 / 115,27
Quarta FPU (Sem *pipeline)*	33.152 EL 168 Mult.	168	120,51 / 129,89
Quarta FPU (Com *pipeline)*	42.564 EL 168 Mult.	132	85,65 / 94,75

3.5.2 Simulador tipo 2

A fim de utilizar o recurso de *pipeline* que algumas FPU possuem, foi projetado um segundo simulador que faz uso desta característica. Ela não utiliza mais o MAC e sim apenas um multiplicador e um somador para realizar os cálculos de cada termo. Os multiplicadores de acoplamento permanecem.

A Figura 21 ilustra os blocos somadores e multiplicadores utilizados neste simulador.

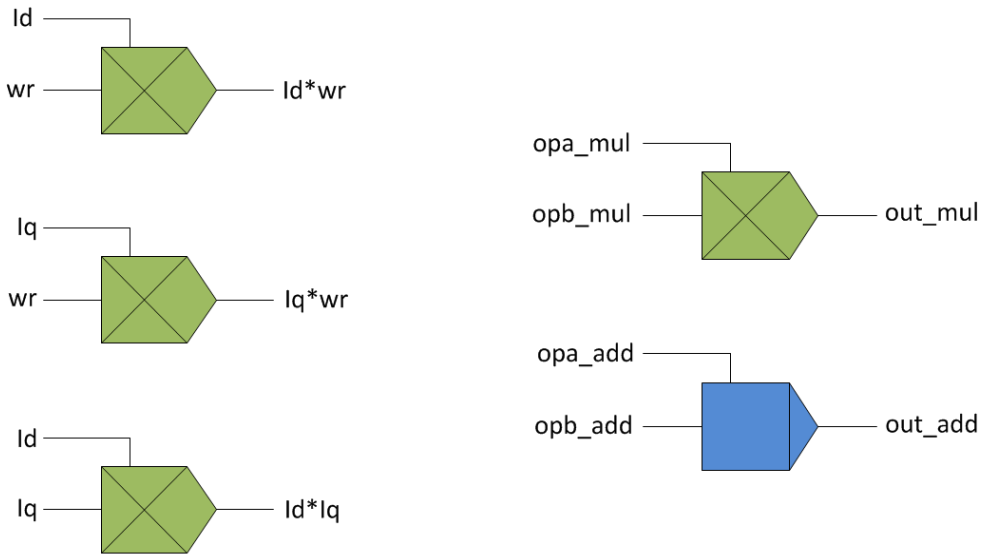

Figura 21: Blocos utilizados no simulador tipo 2

Este simulador é controlado também por uma máquina de estados. Esta máquina de estados é mais complexa do que a do primeiro simulador e pode ser vista na Figura 22. Em cada estado são enviados os operadores corretos para o multiplicador e para o somador. Os estados também fazem a aquisição dos resultados nos momentos corretos. Ao final são enviados os resultados para as saídas e realizada a multiplicação do acoplamento das variáveis para o próximo cálculo. A Figura 23 mostra o pseudocódigo destes.

A latência deste simulador é menor que a do primeiro, já que ele utiliza o *pipeline*. Contudo, o mesmo possui alguns estados em que não é realizado nenhum cálculo, apenas é aguardado o próximo estado. Isto foi feito simplesmente para facilitar a implementação. A ideia do terceiro simulador mostrado mais adiante consiste em melhorar este aspecto da implementação.

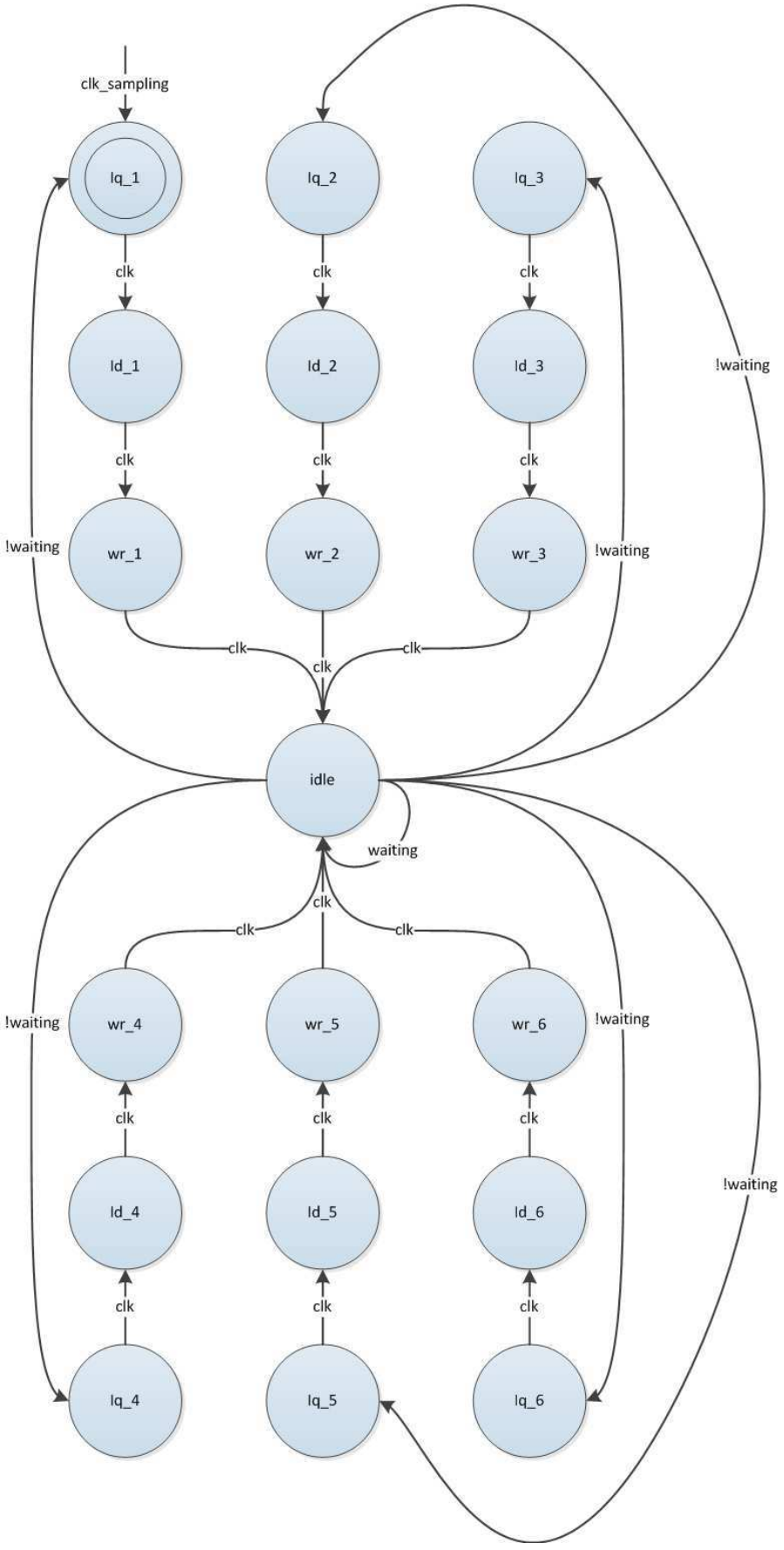

Figura 22: Estados do simulador tipo 2

iq_1:
Operador A do multiplicador <= a_1
Operador B do multiplicador <= $v_q[k-1]$
id_1:
Operador A do multiplicador <= b_1
Operador B do multiplicador <= $v_d[k-1]$
wr_1:
Operador A do multiplicador <= c_1
Operador B do multiplicador <= $i_q[k-1]$
iq_2:
Operador A do multiplicador <= a_2
Operador B do multiplicador <= $i_q[k-1]$
Guarda o resultado da operação de iq_1
id_2:
Operador A do multiplicador <= b_2
Operador B do multiplicador <= $i_d[k-1]$
Guarda o resultado da operação de id_1
wr_2:
Operador A do multiplicador <= c_2
Operador B do multiplicador <= $\omega_r[k-1]$
Guarda o resultado da operação de wr_1
iq_3:
Operador A do multiplicador <= a_3
Operador B do multiplicador <= $\omega_r[k-1]$
Operador A do somador <= Resultado de iq_1
Operador B do somador <= Resultado de iq_2
id_3:
Operador A do multiplicador <= b_3
Operador B do multiplicador <= $i_q[k-1] * \omega_r[k-1]$
Operador A do somador <= Resultado de id_1
Operador B do somador <= Resultado de id_2
wr_3:
Operador A do multiplicador <= c_3
Operador B do multiplicador <= $i_q[k-1] * i_d[k-1]$
Operador A do somador <= Resultado de wr_1
Operador B do somador <= Resultado de wr_2
iq_4:
Operador A do multiplicador <= a_4
Operador B do multiplicador <= $i_d[k-1] * \omega_r[k-1]$
Operador A do somador <= Multiplicação de iq_3
Operador B do somador <= Soma de iq_3
id_4:
Operador A do somador <= Multiplicação de id_3
Operador B do somador <= Soma de id_3
wr_4:
Operador A do multiplicador <= c_4
Operador B do multiplicador <= $T_C[k-1]$
Operador A do somador <= Multiplicação de wr_3
Operador B do somador <= Soma de wr_3
iq_5:
Operador A do somador <= Multiplicação de iq_4
Operador B do somador <= Soma de iq_4
id_5:
Guarda o resultado de i_d
wr_5:
Operador A do somador <= Multiplicação de iq_3
Operador B do somador <= Soma de iq_3
iq_6:
Guarda o resultado de i_q
id_6:
Espera
wr_6:
Guarda o resultado de ω_r
Operador A do multiplicador i_q*i_d <= $i_q[k-1]$
Operador B do multiplicador i_q*i_d <= $i_d[k-1]$
Operador A do multiplicador $i_q*\omega_r$ <= $i_q[k-1]$
Operador B do multiplicador $i_q*\omega_r$ <= $\omega_r[k-1]$
Operador A do multiplicador $i_d*\omega_r$ <= $\omega_r[k-1]$
Operador B do multiplicador $i_d*\omega_r$ <= $i_q[k-1]$
idle:
Espera e direciona para o próximo estado

Figura 23: Pseudocódigo dos estados do simulador tipo 2

Este simulador foi projetado para ter um bom entendimento de como funciona o pipeline da FPU. Nos casos da segunda e terceira FPU não foi possível implementar este simulador, pois não há *pipeline* na unidade de ponto flutuante. A Tabela 19 mostra a quantidade de recursos utilizados na FPGA, a latência e a frequência máxima aceita por este simulador.

O terceiro simulador, que será mostrado na próxima sessão, apresenta um melhor desempenho tanto em termos de área ocupada como de latência do que este simulador. Por esta razão, o segundo simulador foi implementado apenas para a primeira FPU e não para a quarta.

Tabela 19: Recursos da FPGA utilizados, latência e frequência máxima do simulador tipo 2 para cada FPU

FPU	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Primeira FPU	5.547 EL 28 Mult.	27	36,12 / 39,54
Quarta FPU (Com *pipeline*)	-	-	-

3.5.3 Simulador tipo 3

De forma a melhorar a implementação do simulador tipo 2 e utilizar melhor o recurso do *pipeline*, foi feito um simulador com uma máquina de estados que retira ao máximo a quantidade de estados mortos. O intuito deste simulador é também utilizar a menor área possível da FPGA.

Como se pode ver na Figura 24, foram utilizados apenas um multiplicador e um somador de ponto flutuante para realizar todas as operações. Isto proporciona uma solução com uma latência elevada, porém ela utiliza poucos recursos da FPGA.

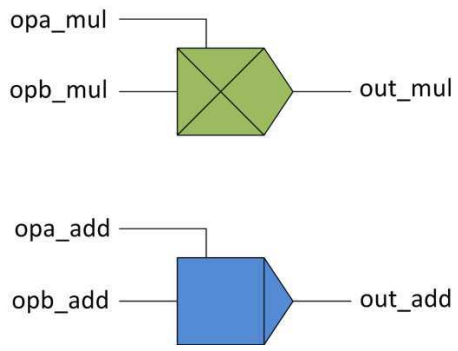

Figura 24: Blocos utilizados no simulador tipo 3

Tabela 20: Recursos da FPGA utilizados, latência e frequência máxima do simulador tipo 3 para cada FPU

FPU	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Primeira FPU	2.654 EL 7 Mult.	26	39,76 / 43,45
Quarta FPU (Com *pipeline*)	26.288 EL 28 Mult.	103	59,28 / 64,99

3.5.4 Simulador tipo 4

Diferentemente dos simuladores tipo 2 e 3, este simulador não utiliza *pipeline.* Este simulador não economiza espaço, mas é a solução mais rápida e mais simples de implementar as equações em tempo real. Ele utiliza muito mais recursos da FPGA, mas a sua latência é a menor de todos os outros simuladores.

A Figura 25 ilustra os blocos utilizados neste simulador e a forma como eles estão conectados. Diferentemente dos outros três, este simulador não possui uma máquina de estados para controlar. Os blocos da FPU estão simplesmente conectados de tal forma que cada um realiza apenas uma operação. Caso a planta a ser simulada tenha equações diferentes, esta é a maneira mais simples de implementar se existir a disponibilidade de uma FPGA com recursos suficientes para isto.

A Tabela 21 apresenta a quantidade de recursos utilizados para este simulador no caso de utilizar cada uma das FPU. Este simulador é bastante simples de descrever já que não é preciso projetar nenhuma máquina de estados e não utiliza *pipeline*, portanto foi possível realizar a implementação dele para todas as FPU.

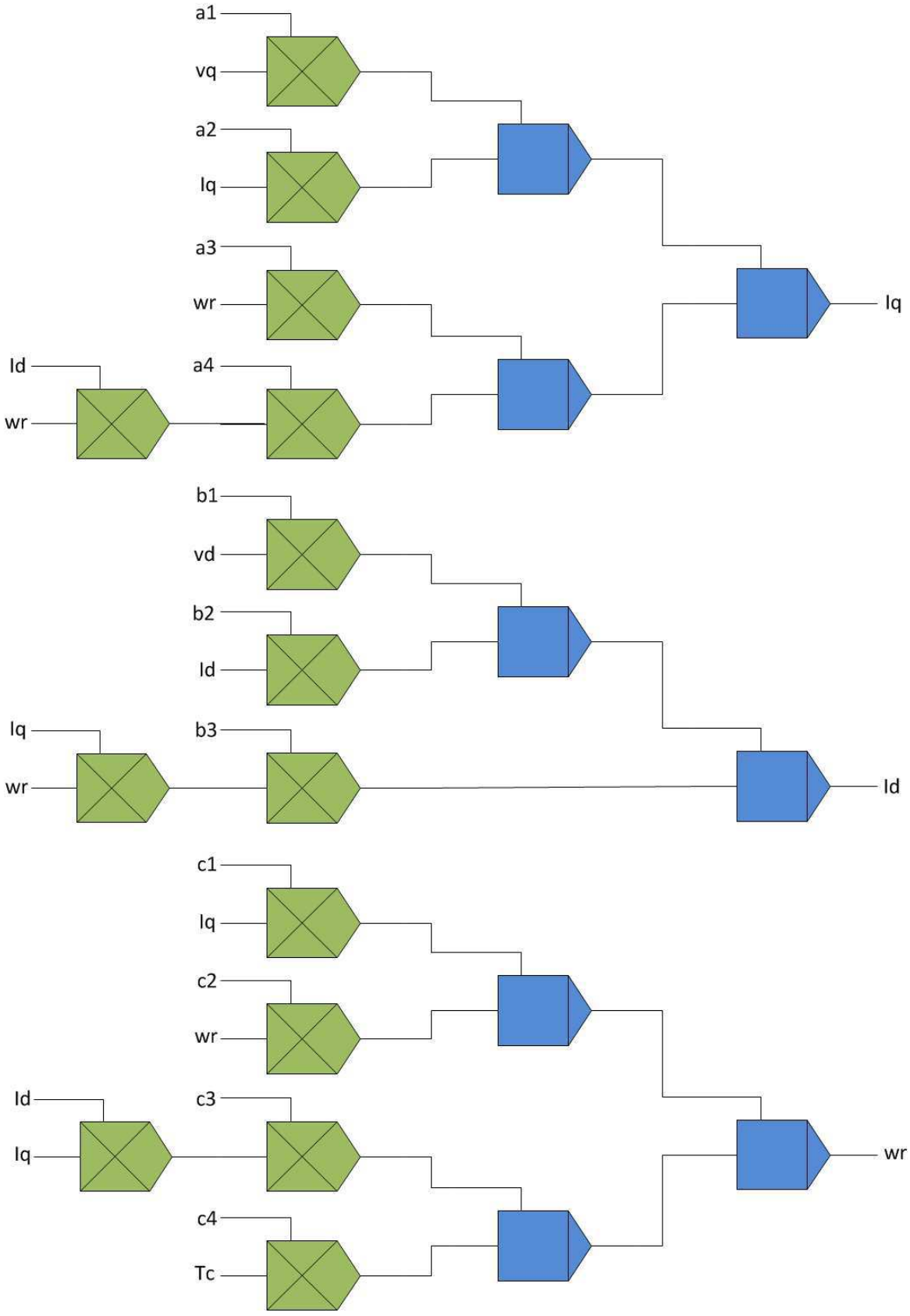

Figura 25: Blocos utilizados no simulador tipo 4

Tabela 21: Recursos da FPGA utilizados, latência e frequência máxima do simulador tipo 4 para cada FPU

FPU	Recursos utilizados da FPGA	Latência	Frequência Máxima (MHz)
Primeira FPU	18.988 EL 91 Mult.	16	37,02 / 40,45
Segunda FPU (Precisão simples)	9.721 EL 98 Mult.	24	107,81 / 119,95
Segunda FPU (Precisão dupla)	22.654 EL 252 Mult.	24	90,75 / 100,56
Terceira FPU	26.192 EL 52 Mult.	38	94,35 / 103,7
Quarta FPU (Sem *pipeline)*	107.360 EL 372 Mult.	104	75,83 / 84,32
Quarta FPU (Com *pipeline)*	78.960 EL 372 Mult.	94	74,22 / 82,93

3.5.5 Resumo e comparação dos simuladores

Após simular as equações do motor síncrono a ímã permanente de quatro formas diferentes, foi possível observar que a melhor maneira como o simulador em tempo real deve ser escrita depende da aplicação e da disponibilidade de recursos que o projetista dispõe.

O terceiro e quarto simulador são os casos extremos respectivamente de área ocupada, ou recursos da FPGA utilizados, e latência em ciclos de *clock* após a aplicação dos operandos na entrada para obter o resultado desejado.

Neste caso, o terceiro simulador utiliza apenas um somador e um multiplicador, desperdiçando pouca área da FPGA. Contudo, ele vai precisar de mais ciclos de *clock* para poder entregar o resultado. Portanto, não tem sentido utilizar este tipo de simulador caso a FPU não possua *pipeline*, porque seria necessária uma FPU que possa operar com um *clock* bastante elevado para poder obter um passo de cálculo menor do que 1 µs.

Enquanto isso, o quarto simulador utiliza 14 multiplicadores e 8 somadores para realizar as operações. Com isto, cada um destes blocos realiza apenas uma operação por passo de cálculo. Isto faz com que a FPGA seja mais utilizada em relação aos outros simuladores. Porém, este simulador será aquele que vai dar a resposta de forma mais rápida, ou seja, com uma menor latência. Esta é a maneira mais fácil de projetar o simulador, dado que não é preciso se preocupar com latências, basta conectar os blocos da maneira como as equações estão escritas.

Os simuladores do tipo 1 e 2 foram projetados como uma variação entre os tipos 3 e 4. Eles apresentam uma quantidade intermediária de somadores e multiplicadores. A Tabela 22 mostra a quantidade de instancias de somadores e multiplicadores de ponto flutuante em cada simulador implementado.

Tabela 22: Quantidade de somadores e multiplicadores em cada simulador

Simulador	Quantidade de Somadores	Quantidade de Multiplicadores
Simulador 1	3	6
Simulador 2	1	4
Simulador 3	1	1
Simulador 4	8	14

3.6 Aquisição dos dados da FPGA

O intuito deste projeto é realizar simulações em tempo real na FPGA, mas primeiramente é necessário provar que a simulação está de fato calculando corretamente os modelos matemáticos.

O ideal seria realizar uma comunicação veloz suficiente entre a FPGA e um computador para fazer a aquisição de dados em tempo real e poder comparar com um modelo de referência no MATLAB ou Simulink. Como uma comunicação deste tipo geralmente é demasiadamente complexa de ser implementada em FPGA, optou-se pela simplicidade: armazenar certa quantidade de amostras em uma memória e depois enviar através de uma comunicação serial RS232 que é bastante simples de ser implementada.

A princípio tentou-se utilizar a memória SDRAM externa que os kits de desenvolvimento citados possuem. No caso da DE2 a memória é de 8 MB e para a DE2-115 são 128 MB. Foi implementado um controlador simplificado para escrever nesta memória e depois realizar a leitura para poder enviar pela serial. Contudo, não se obteve sucesso na operação de escrita na SDRAM. Ao realizar a leitura através de um *software* conhecido por Painel de Controle que a DE2 fornece em seu CD-ROM, constatou-se que sempre existia alguns valores gravados que estavam corrompidos.

Devido a este problema, foi utilizada a DE2-115 para implementar três registradores, cada um com 10 mil amostras para cada variável. Foi escolhido um tempo de amostragem de aproximadamente 100 µs. Com isto, 1 segundo dos cálculos eram armazenados e depois enviados para o computador para serem analisados no MATLAB em comparação ao modelo implementado no Simulink.

3.7 Análise das simulações

Para realizar as simulações foram utilizados três tipos diferentes de estímulos. O primeiro deles aplicava uma tensão v_q constante em 50 V e v_d constante em 20 V enquanto que o torque da carga T_C era nulo. O segundo estímulo consistiu nas mesmas tensões, mas com um torque de 1.5 N.m. O terceiro consistiu em começar com o primeiro estímulo e mudar para o segundo em 0.5 segundo para ver a reação que o sistema teria quando este já estava funcionando.

A seguir foram feitas as análises dos resultados obtidos para cada um dos simuladores anteriormente citados. Estas análises estão separadas em duas partes: a primeira para os simuladores de precisão simples e a segunda para os simuladores de precisão dupla.

A Figura 26, Figura 27 e Figura 28 retratam os estes três estímulos que foram aplicados no sistema. A primeira figura corresponde ao estímulo 1, a segunda figura corresponde ao estímulo 2 e a última corresponde ao estímulo 3.

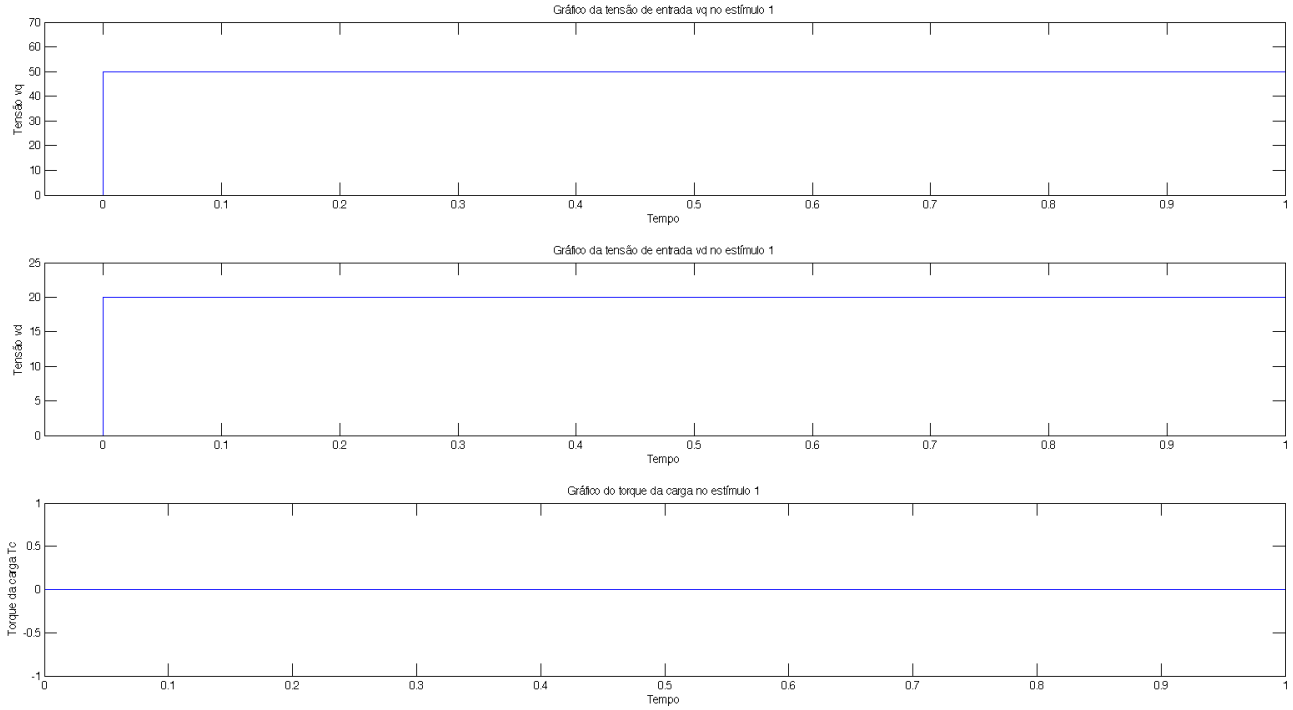

Figura 26: Estímulo 1 aplicado nas entradas do sistema

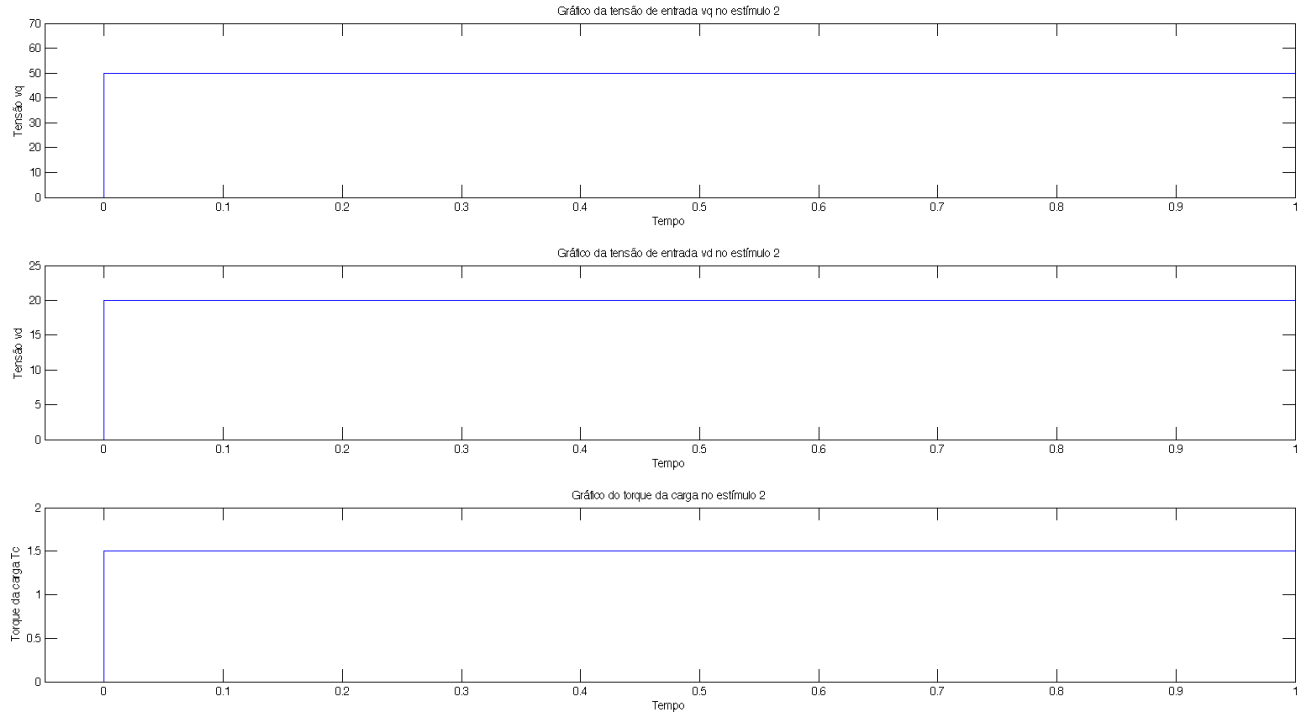

Figura 27: Estímulo 2 aplicado nas entradas do sistema

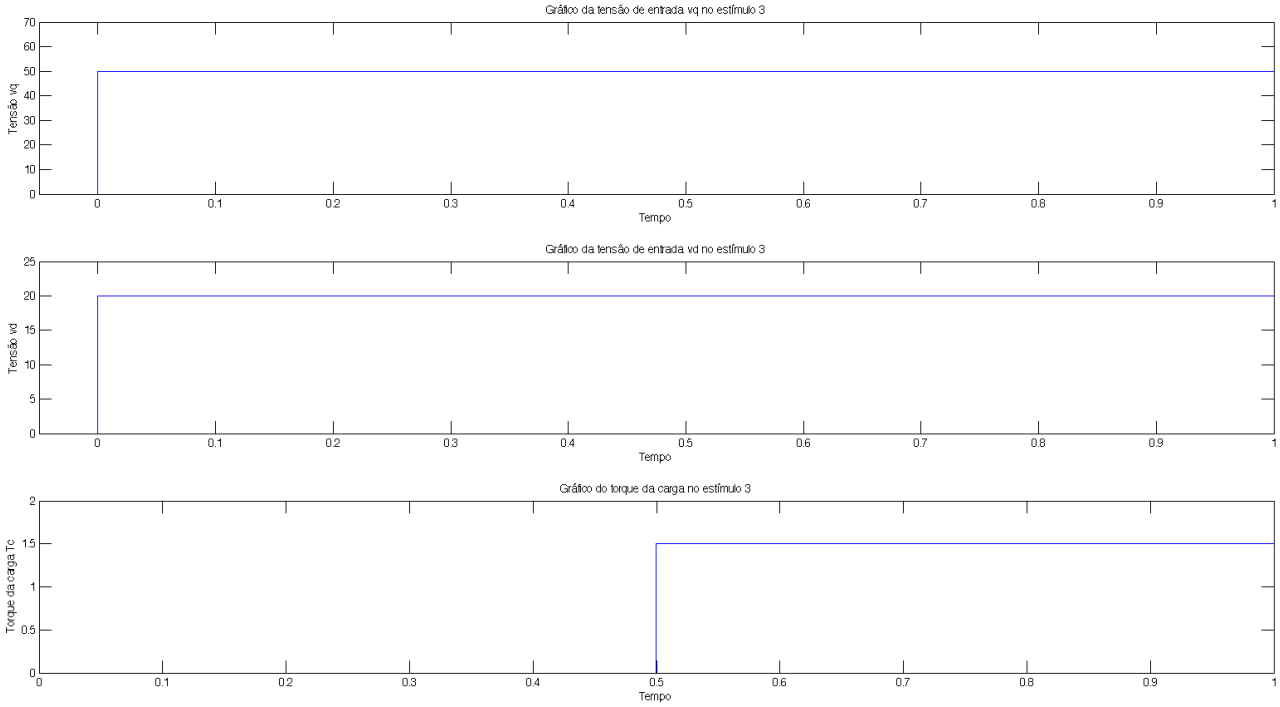

Figura 28: Estímulo 3 aplicado nas entradas do sistema

Os simuladores foram testados em diferentes frequências de operação e passos de cálculo. A ideia foi testar a FPU perto do limite de operação. Para isto, cada uma foi testada em diferentes frequências com incrementos de 50 MHz. Então, quando a FPU não se apresentava mais de forma satisfatória, a frequência anterior a esta foi utilizada neste projeto. O passo de cálculo dependia da frequência de operação e da latência do simulador. Assim, foi dada a preferência por sempre utilizar o mesmo passo de calculo, 0,64 µs. Contudo, isto não foi possível em um dos casos e um passo de cálculo de 1 µs foi utilizado. A Tabela 23 lista estes parâmetros para cada simulador testado neste projeto.

Tabela 23: Frequência de operação e passo de cálculo utilizado em cada simulador

Simulador	Frequência de operação (MHz)	Passo de cálculo (µs)
FPU1 – Simulador 1	50	0,64
FPU1 – Simulador 2	50	0,64
FPU1 – Simulador 3	50	0,64
FPU1 – Simulador 4	50	0,64
FPU2 – Simulador 1 *(precisão simples)*	100	0,64
FPU2 – Simulador 4 *(precisão simples)*	100	0,64
FPU2 – Simulador 1 *(precisão dupla)*	100	0,64
FPU2 – Simulador 4 *(precisão dupla)*	100	0,64
FPU3 – Simulador 1	200	1,00
FPU3 – Simulador 4	200	0,64
FPU4 – Simulador 1 *(sem pipeline)*	200	0,64
FPU4 – Simulador 1 *(com pipeline)*	200	0,64
FPU4 – Simulador 4 *(com pipeline)*	200	0,64

3.7.1 Precisão simples

Ao aplicar o primeiro estímulo que corresponde aos sinais da Figura 26, obtém-se o seguinte resultado para a corrente i_d ilustrado na Figura 29. Será analisado aqui apenas o resultado da saída da correte i_d, mas todas as saídas foram igualmente calculadas: i_q, i_d e ω_r.

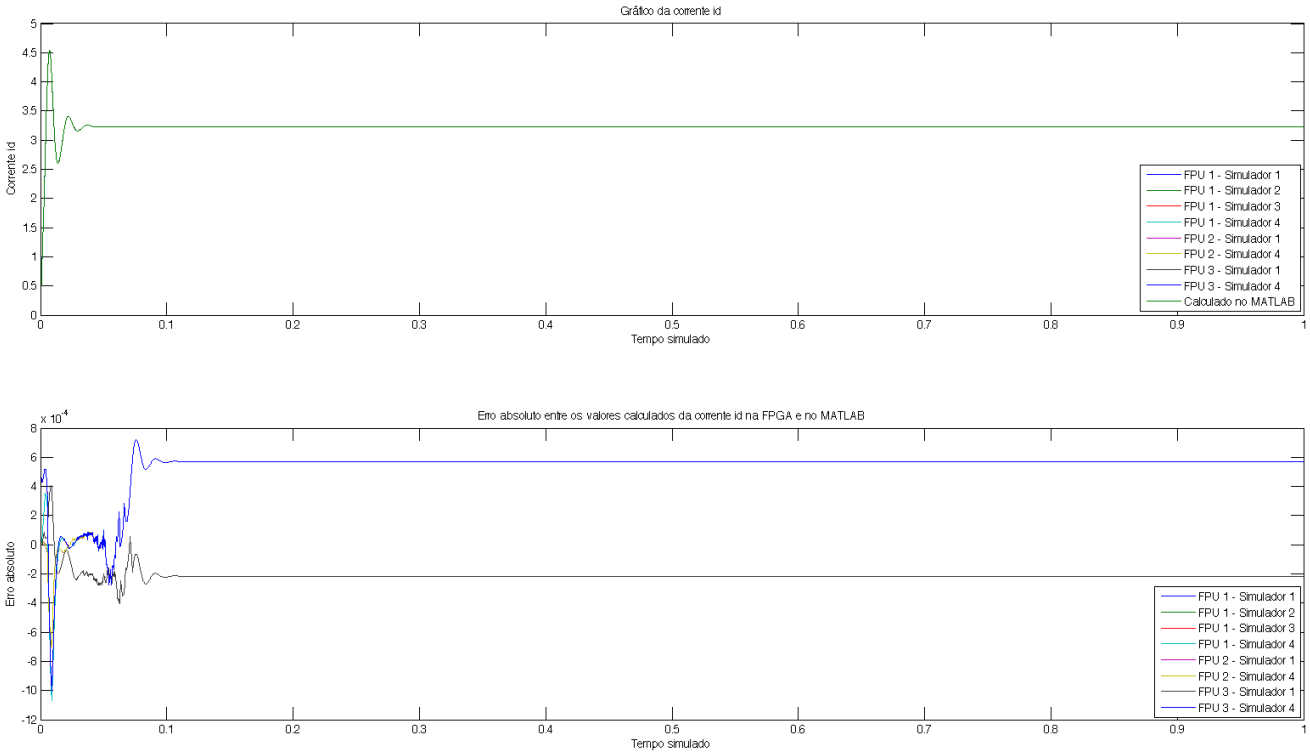

Figura 29: Corrente i_d para o estímulo 1 – precisão simples

O primeiro gráfico da Figura 29 ilustra a saída de todos os simuladores. Independente de qual FPU foi utilizada ou de como o simulador foi projetado, todas as saídas ficaram bastante próximas umas das outras quando se analisa a saída na escala das unidades. O mesmo ocorreu para i_q e ω_r. Não é possível diferenciar as curvas plotadas nesta escala.

Ao calcular o erro absoluto entre o valor calculado pelo modelo no MATLAB, é possível ver melhor a diferença entre elas principalmente no transitório, como pode ser ilustrado no segundo gráfico da Figura 29. Em regime permanente, elas permanecem constantes, mas ainda assim com um erro em relação ao valor do MATLAB. Contudo, este gráfico foi plotado na escala de 10^{-3}. Para muitas aplicações, este erro seria totalmente irrelevante. O que torna esta precisão suficiente para diversos fins. A corrente i_q também apresenta um erro absoluto bastante similar, assim como o ω_r, embora este seja plotado na escala de 10^{-2}.

Ao aplicar o segundo estímulo que corresponde aos sinais da Figura 27, obtém-se o seguinte resultado para a corrente i_d ilustrado na Figura 30. Este estímulo faz com que mais um termo entre nos cálculos da velocidade angular ω_r e por fim nas outras devido à dependência entre elas. A fim de detectar como estes

simuladores se comportam em precisão simples com este erro a mais, este estímulo foi aplicado e as saídas analisadas.

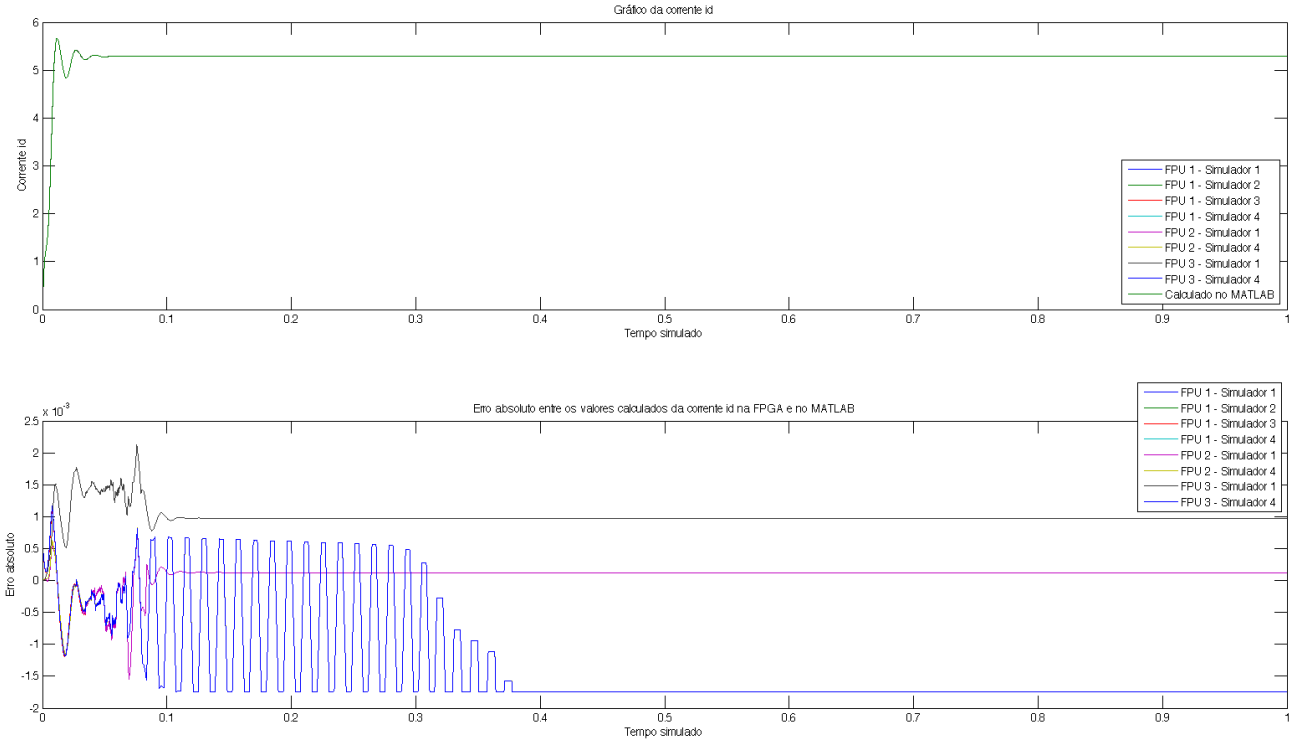

Figura 30: Corrente i_d para o estímulo 2 – precisão simples

Assim como no primeiro estímulo, as saídas são bem próximas umas das outras e o erro absoluto é da ordem de 10^{-3} também. A diferença que pode ser observada aqui é a oscilação que o simulador do tipo 4 da FPU 3 apresenta. Possivelmente estas oscilações podem ter sido causadas por causa da frequência de operação (200 MHz) que estava além da frequência máxima que a ferramenta *TimeQuest Timing Analyzer* do Quartus II aconselhou como pode ser visto na Tabela 21. Contudo, em regime permanente, assim como no primeiro caso, o erro fica constante.

Ao aplicar o terceiro estímulo que corresponde aos sinais da Figura 28, obtém-se o seguinte resultado para a corrente i_d ilustrado na Figura 31. Neste estímulo, o mesmo termo do caso anterior será acrescentado, mas enquanto o sistema já estiver funcionando.

A mesma oscilação presente no caso anterior pode ser observada aqui. Contudo o erro ficou constante e na mesma ordem de grandeza que os casos anteriores. O mesmo ocorreu para as saídas i_q e ω_r.

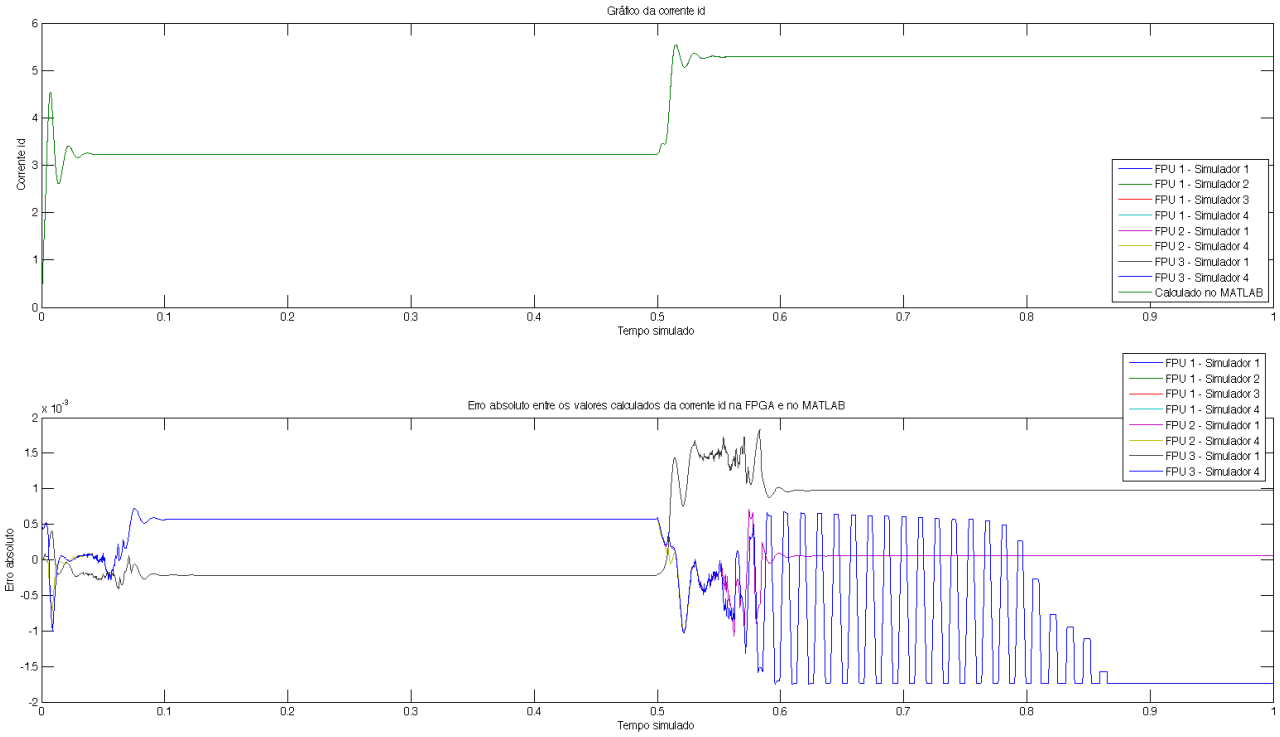

Figura 31: Corrente i_d para o estímulo 3 – precisão simples

O caso da precisão simples pode ser mais do que suficiente para muitas aplicações. Na situação apresentada, o erro absoluto seria da ordem de mA. Para as aplicações mais sofisticadas em que uma precisão maior é necessária, o uso da precisão dupla pode ser utilizado como será visto na sessão 3.7.2.

3.7.2 Precisão dupla

Nesta sessão, aplicam-se os mesmos estímulos ilustrados na Figura 26, Figura 27 e Figura 28 nos simuladores que possuem precisão dupla.

No caso da precisão dupla, infelizmente devido a algumas inconsistências apresentadas na quarta FPU e contradições em sua documentação e em seu código, não foi possível utilizá-la no simulador tipo 3 e no simulador do tipo 4. Contudo, foi possível utilizar no simulador tipo 4 a parte da quarta FPU que apresenta *pipeline*, mesmo não sendo necessário este recurso neste simulador.

O primeiro gráfico da Figura 32 ilustra a corrente i_d calculada pelos diversos simuladores de precisão dupla. Não é possível perceber a diferença entre estas curvas e a curva calculada pelo MATLAB usada como modelo de referência. No segundo gráfico da Figura 32 é possível ver o erro absoluto de cada uma das curvas em relação à curva calculada pelo MATLAB. Apesar da grande oscilação

apresentada por algumas FPU, este simulador calculou esta corrente com um erro da ordem de 10^{-11}, o que representa 10 pA. Para a maioria das aplicações, esta precisão é mais do que suficiente, dado que esta corrente é extremamente baixa quando comparada com o valor real da corrente que é aproximadamente 3,2 A. Sendo assim, esta oscilação não representa um grande erro para o sistema.

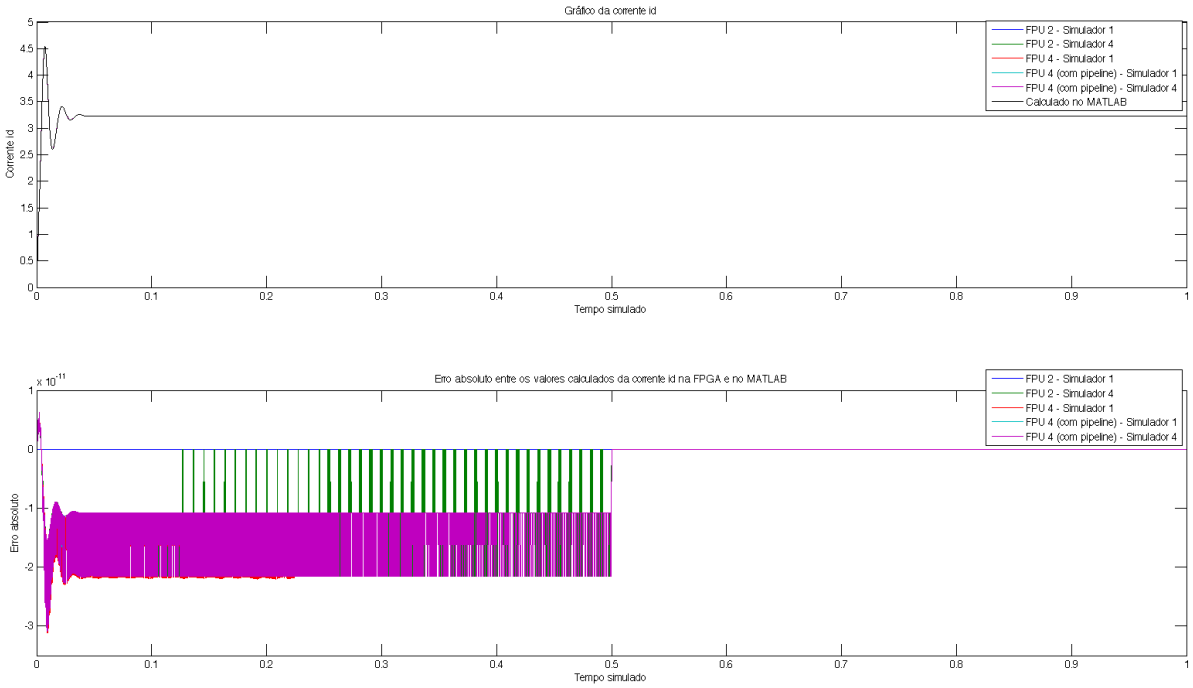

Figura 32: Corrente i_d para o estímulo 1 – precisão dupla

Na Figura 33 obteve-se a mesma oscilação em alguns simuladores, mas como a ordem é a mesma, isto não representa um problema para o sistema. Em ambos os casos, este erro se estabiliza em aproximadamente 0,5 s. O mesmo ocorre com as curvas apresentadas na Figura 34 quando o estímulo 3 é aplicado.

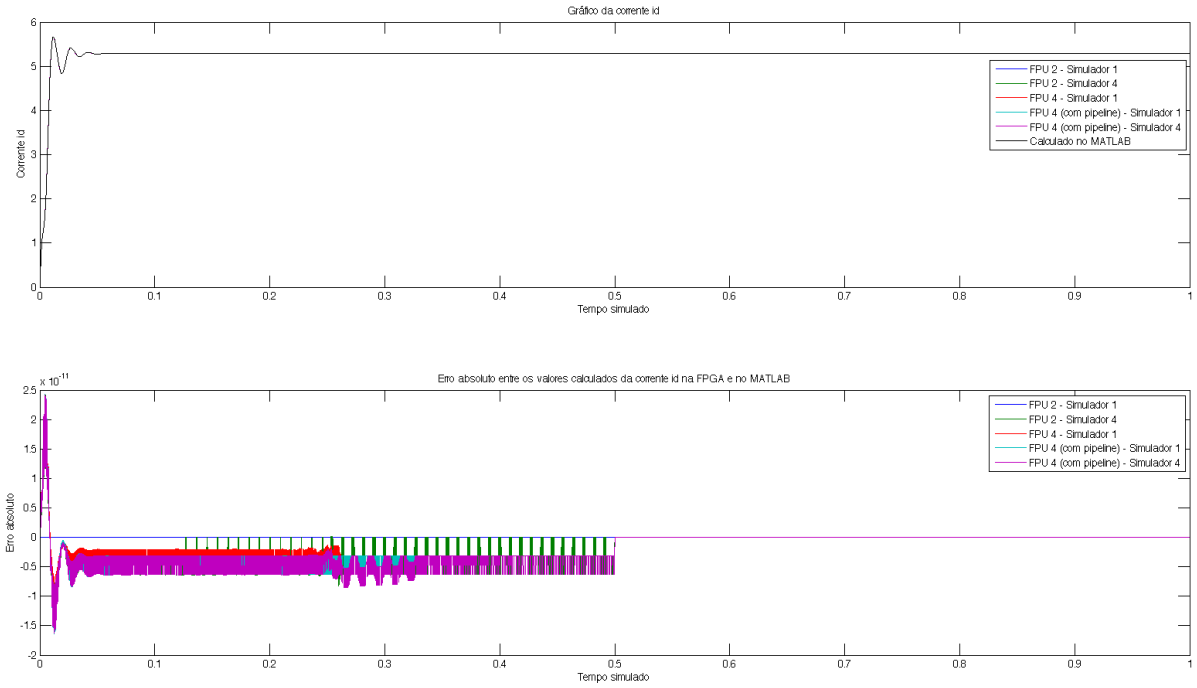

Figura 33: Corrente i_d para o estímulo 2 – precisão dupla

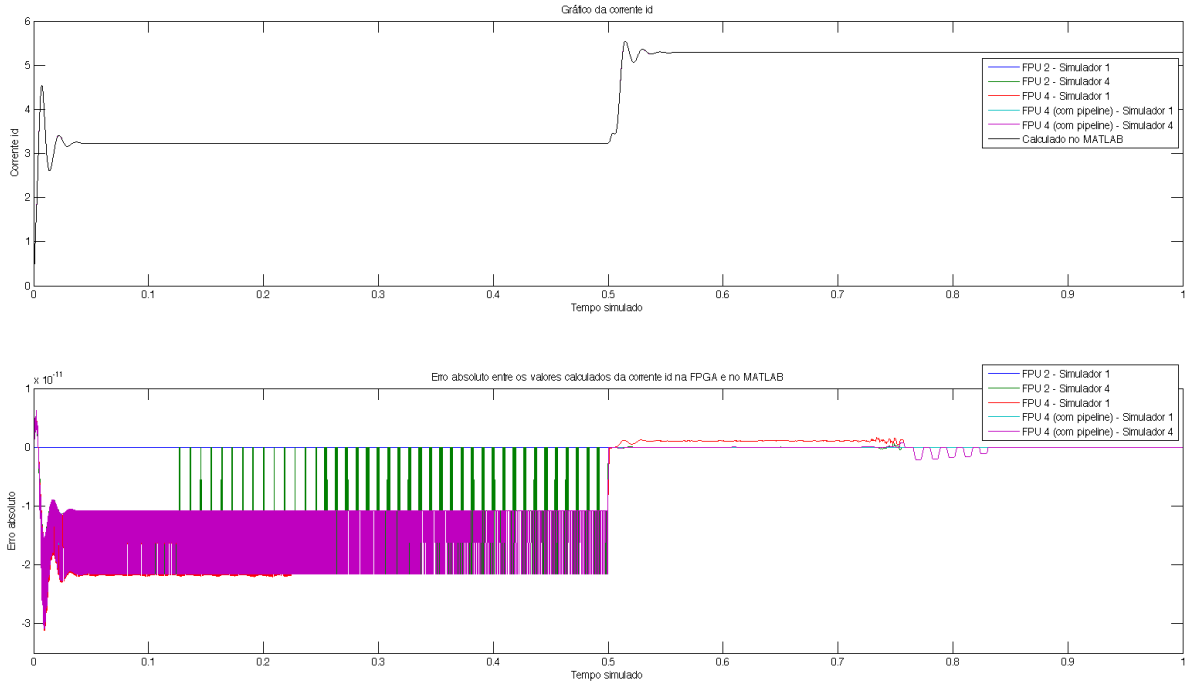

Figura 34: Corrente i_d para o estímulo 3 – precisão dupla

Em todos os casos, foi possível observar que um dos simuladores não oscila na escala mostrada e que seu valor permanece desde o início em aproximadamente zero. Ao plotar separadamente o erro absoluto deste simulador, percebe-se que o seu erro é ainda menor quando comparado com os outros. Este erro absoluto é da ordem de 10^{-15}, sendo este o melhor resultado obtido neste projeto. Para a grande

maioria das aplicações, este erro absoluto é suficiente já que apresenta um erro na ordem de 1 fA, o que é uma corrente muito baixa quando comparada com os valores da corrente em questão que é 3,22 A.

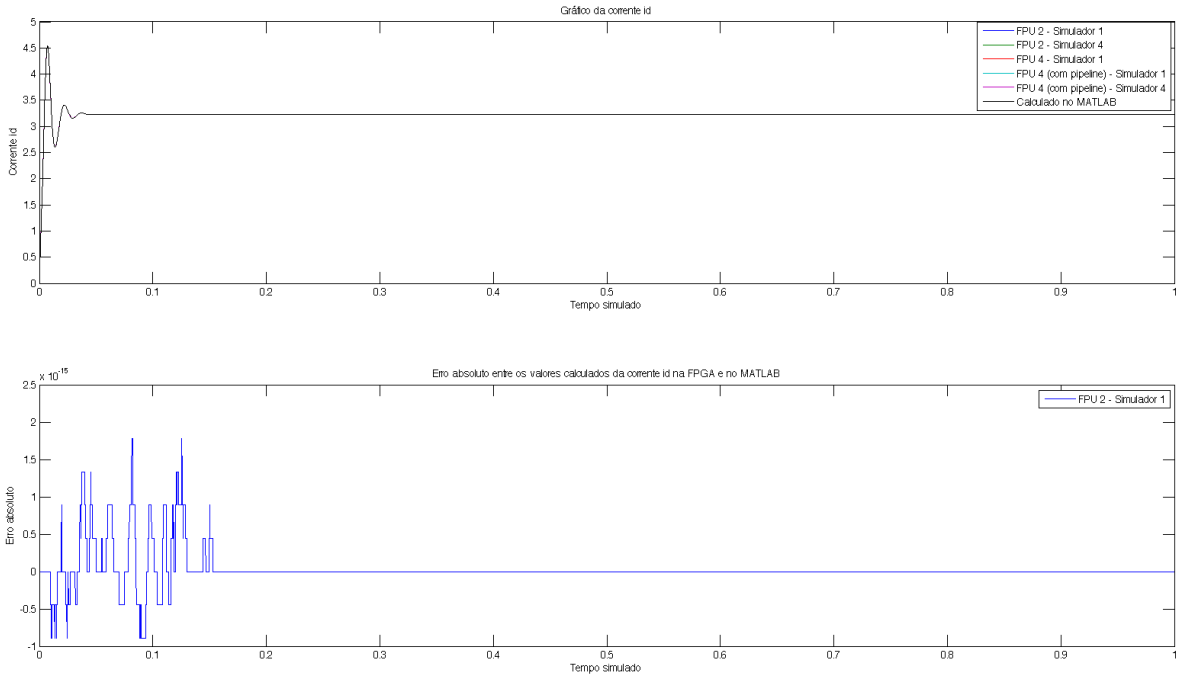

Figura 35: Corrente i_d para o estímulo 1 – precisão dupla – FPU 2 (Altera)

4. Conclusão

4.1 Resultados

Diversas FPU tanto com precisão simples como precisão dupla foram testadas. Isto possibilitou um maior entendimento sobre o assunto e que resultados mais concretos fossem obtidos ao final deste estudo no que se diz respeito à escolha da FPU e da forma como o simulador deve ser projetado.

A precisão simples se comportou como o esperado, isto é, mais preciso do que o formato em ponto fixo e sem a necessidade de dimensionar quantidade de bits para parte inteira e fracionaria, e apresentou erros absolutos baixos, o que poderia ser suficiente para muitas aplicações. No caso da precisão dupla, ele se comportou ainda melhor, no que se diz respeito à precisão em comparação com a precisão simples e com o ponto fixo, com um erro extremamente pequeno, o que aumenta a gama de aplicações que estes simuladores podem ser utilizados.

Apesar de diversas FPU terem sido utilizadas, um destaque maior deve ser dado à segunda FPU que tem a Altera como desenvolvedora. No caso da precisão simples ela se comportou de forma similar às outras. Contudo, na precisão dupla, ela foi a que apresentou o melhor desempenho com um erro absoluto menor do que as outras.

Também vale salientar que um recurso importante nestes simuladores é o *pipeline*. Não foi possível utilizar *pipeline* na FPU da Altera, embora tenha sido possível com a primeira FPU que também merece destaque por ter este recurso embutido e por apresentar as menores latências, apesar de ter apenas a versão com precisão simples.

A terceira e quarta FPU apresentaram maiores latências em suas operações e ocupam muito mais espaço, quando comparadas com as duas primeiras. Elas também apresentam algumas inconsistências em sua documentação que aparentam ser contraditórias em alguns pontos, como por exemplo, as latências de suas operações. Contudo, elas apresentam frequências máximas elevadas e possuem um protocolo simples em sua interface.

Talvez melhores resultados pudessem ser obtidos com uma FPGA que suportasse maiores frequências. A Cyclone II e a Cyclone IV precisam de algumas restrições de tempo quando as frequências se elevam acima de 150 MHz. Isto se torna mais difícil quando parte do projeto, neste caso as FPU, são de terceiros e praticamente impossível quando os módulos utilizados (FPU) são caixas pretas como a FPU da Altera.

4.2 Trabalhos futuros

Apesar de um ótimo resultado ter sido obtido ao final deste trabalho, muito ainda pode ser feito principalmente no que se diz respeito à aplicação deste sistema em uma simulação HIL.

Primeiramente, uma comunicação veloz o suficiente para poder transferir todos os pontos calculados da FPGA para o computador pode ser implementada. Como por exemplo, USB ou Ethernet Gigabit que apresentam elevadas taxas de comunicação.

Em segundo lugar uma melhor implementação do MAC pode ser realizada, já que este foi o simulador que melhor resultado apresentou com o uso de precisão dupla, principalmente no que se diz respeito à utilização de *pipeline*. A fim de diminuir a quantidade de recursos utilizados na FPGA, os três MAC utilizados no simulador 1 podem ser integrados em apenas um MAC que apresente *pipeline*, isto é, três acumuladores ligados a apenas um somador e um multiplicador com *pipeline*. Desta forma a cada ciclo de clock ele vai realizar um cálculo da devida equação e somar no acumulador correto.

A forma como o sistema foi escrito já apresenta um arquivo separado que permite alterar apenas as constantes do sistema sem a necessidade de modificações no código do simulador. Uma melhor automação da alteração e geração de constantes pode ser feita. Uma automação do processo de alteração, compilação e programação da FPGA também pode ser feita para facilitar o uso deste sistema.

Outras equações de diferentes aplicações podem ser simuladas para poder ter uma maior gama de resultados e testes. Com isto, pode-se possivelmente encontrar mais erros e mais questões a serem melhoradas nos simuladores. Isto

também pode ser obtido ao variar as formas de estímulos que este sistema está sendo alimentado, como por exemplo, senóides podem ser aplicadas.

Uma melhoria na forma como estes simuladores são descritos também pode ser estudada. Uma forma mais eficiente de implementar pode ser criada.

Por fim, testes poderiam ser feitos com FPGA de maiores capacidades e que suportem uma frequência de operação mais elevada como, por exemplo, alguma da linha da Virtex ou Stratix. Da mesma maneira, FPU que suportem maiores frequências, ocupem menos espaço da FPGA e com latências menores também podem ser pesquisadas, bom como, podem ser objetivo de desenvolvimento e síntese.

5. Referências Bibliográficas

[1] Y. Chen and V. Dinavahi, “FPGA-based real-time EMTP”, IEEE Transactions on Power Delivery, vol. 24, no. 2, pp. 892–902, April 2009;

[2] T. O. Bachir and J. P. David, “FPGA-Based Real-Time Simulation of State-Space Models Using Floating-Point Cores”, 14th International Power Electronics and Motion Control Conference, EPE-PEMC 2010;

[3] A. Myaing and V. Dinavahi, “FPGA-Based Real-Time Emulation of Power Electronic Systems With Detailed Representation of Device Characteristics”, IEEE Transactions on Industrial Electronics, Vol. 58, no. 1, January 2011;

[4] E. M. Fernandes, “Estimação de posição e velocidade de uma máquina síncrona a ímã permanente”, Dissertação de Mestrado, Universidade Federal de Campina Grande, Departamento de Engenharia Elétrica;

[5] IEEE Computer Society, “IEEE Standard for Floating-Point Arithmetic”, IEEE Std 754™-2008 (Revision of IEEE Std 754-1985);

[6] R. Usselmann, “Open Floating Point Unit”, The Free IP Cores Projects, www.opencores.org, 16 de Setembro de 2000;

[7] Accellera®, “SystemVerilog 3.1a Language Reference Manual; Accellera’s Extensions to Verilog®”, Copyright © 2002, 2003, 2004 by Accellera Organization, Inc.;

[8] Altera®, “DE2 Development and Education Board; User Manual”, Copyright © 2006 Altera Corporation, Version 1.4;

[9] Morris Mano, “Logic and Computer Design Fundamentals”, Third Edition (2003), Prentice Hall;

[10] S. C. Chapra e R. P. Chapra, “Numerical Methods for Engineers”, Fifth Edition (2008), The McGraw Hill Companies;

[11] C. Dufour, H. Blanchette, and J. Belanger, “Very-high speed control of an FPGA-based finite-element-analysis permanent magnet synchronous virtual motor drive

system," in 34th Annual Conference of IEEE Industrial Electronics, Nov. 2008, pp. 2411–2416;

[12] G. Parma and V. Dinavahi, "Real-time digital hardware simulation of power electronics and drives," IEEE Transactions on Power Delivery, vol. 22, no. 2, pp. 1235–1246, April 2007;

[13] R. Osornio-Rios, R. de J. Romero-Troncoso, L. Morales-Velazquez, J. de Santiago-Perez, R. de J. Rivera-Guillen, and J. de Jesus Rangel-Magdaleno, "A real-time FPGA based platform for applications in mechatronics," in International Conference on Reconfigurable Computing and FPGAs., Dec. 2008, pp. 289–294;

[14] T. Li and Y. Fujimoto, "Control system with high-speed and real-time communication links," IEEE Transactions on Industrial Electronics, vol. 55, no. 4, pp. 1548–1557, April 2008;

[15] J. C. G. Pimentel, "Implementation of simulation algorithms in FPGA for real time simulation of," in IEEE International Conference on Reconfigurable Computing and FPGA's, Sept. 2006, pp. 1–8;

[16] E. Duman, H. Can, and E. Akin, "Real time FPGA implementation of induction machine model - a novel approach," in International Aegean Conference on Electrical Machines and Power Electronics., Sept. 2007, pp. 603–606;

[17] Altera®, "Floating Point Megafunctions; User Guide", Copyright © 2011 Altera Corporation, Version 11.1;

[18] J. Al-Eryani, "Floating Point Unit", The Free IP Cores Projects, www.opencores.org, 28 de Março de 2006;

[19] D. Lundgren, "Double Precision Floating Point Unit", www.opencores.org.

6. Sumário dos Apêndices

Apêndice A. Entradas e saídas do bloco comparador da primeira FPU

Sinal	Largura	Direção	Descrição
opa, opb	32	Entrada	Operandos A e B
unordered	1	Saída	Sinaliza se opa ou opb é NaN
altb	1	Saída	Sinaliza se opa é maior do que opb
blta	1	Saída	Sinaliza se opb é maior do que opa
aeqb	1	Saída	Sinaliza se opa é igual a opb
inf	1	Saída	Sinaliza se opa ou opb é INF
zero	1	Saída	Sinaliza se opa é um zero numérico