Ministério da
Ciência e Tecnologia

INPE-15764-TDI/1507

EMULAÇÃO E CO-SIMULAÇÃO DO SISTEMA DE CONTROLE DE ATITUDE DA PMM E DO SISTEMA ELETRO-HIDRÁULICO DE UMA AERONAVE USANDO FPGAS

Guilherme Seelaender

Dissertação de Mestrado do Curso de Pós-Graduação em Enegenharia e Tecnologia Espaciais/Mecânica Espacial e Controle, orientada pelo Dr. Marcelo Lopes de Oliveira e Souza, aprovada em 2 de abril de 2009.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m18@80/2009/04.09.15.46>

INPE
São José dos Campos
2009

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6911/6923
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

Ministério da
Ciência e Tecnologia

INPE-15764-TDI/1507

EMULAÇÃO E CO-SIMULAÇÃO DO SISTEMA DE CONTROLE DE ATITUDE DA PMM E DO SISTEMA ELETRO-HIDRÁULICO DE UMA AERONAVE USANDO FPGAS

Guilherme Seelaender

Dissertação de Mestrado do Curso de Pós-Graduação em Enegenharia e Tecnologia Espaciais/Mecânica Espacial e Controle, orientada pelo Dr. Marcelo Lopes de Oliveira e Souza, aprovada em 2 de abril de 2009.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m18@80/2009/04.09.15.46>

INPE
São José dos Campos
2009

Dados Internacionais de Catalogação na Publicação (CIP)

S33e

Seelaender, Guilherme.
Emulação e co-simulação do sistema de controle de atitude da PMM e do sistema eletro-hidráulico de uma aeronave usando FPGAS / Guilherme Seelaender. – São José dos Campos : INPE, 2009.
195p. ; (INPE-15764-TDI/1507)

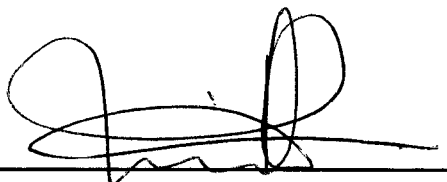
Dissertação (Mestrado em Mecânica Espacial e Controle) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2009.
Orientador : Dr. Marcelo Lopes de Oliveira e Souza.

1. Field Programmable Gate Array (FPGA. 2. Fluxo de desenvolvimento. 3. Co-simulação. 4. Emulação. 5. Sistemas aeroespaciais. I.Título.

CDU 629.7.018

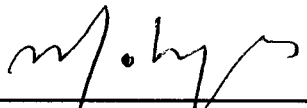
Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de Mestre em

Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle



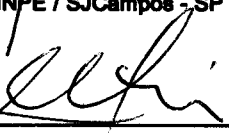
Dr. Paulo Giácomo Milani

Presidente / INPE / SJCampos - SP



Dr. Marcelo Lopes de Oliveira e Souza

Orientador(a) / INPE / SJCampos - SP

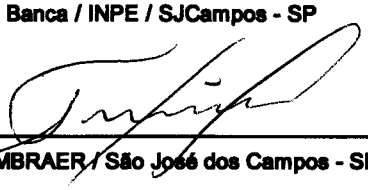


Dr. Mario Cesar Ricci

Membro da Banca / INPE / SJCampos - SP

Dr. Valdemir Carrara

Membro da Banca / INPE / SJCampos - SP



Dr. Francisco Javier Triveño Vargas

Convidado(a) / EMBRAER / São José dos Campos - SP

Aluno (a): Guilherme Seelaender

São José dos Campos, 02 de abril de 2009

“Não há só um método para estudar as coisas”

ARISTÓTELES

A meus pais,
Sergio e Theresinha Seelaender, e
à minha querida Angela.

AGRADECIMENTOS

Este trabalho coroa uma etapa importante no meu crescimento profissional e pessoal. Ela só foi possível de ser atingida porque alicerçada sobre pilares sólidos, partilhados por outras pessoas e instituições: paciência e sabedoria para ensinar; generosidade na disponibilidade de recursos; paz, estímulo e confiança. A essas pessoas e instituições rendo o meu agradecimento.

Ao INPE, pela utilização de suas instalações e oportunidade de compartilhamento de um ambiente agradável e estimulante para o aprendizado e a pesquisa.

Ao meu orientador, Marcelo Lopes, pela paciência, sabedoria e competência na transmissão de conhecimentos, bem como no apoio na elaboração deste trabalho.

À Embraer, em especial à minha gerência, pelas horas liberadas – e elas não foram poucas – para dedicação a esta pós-graduação, bem como pela compreensão com as eventuais interferências advindas da concorrência de atividades, na condição de profissional estudante.

À National Instruments, em especial na figura de Leandro Fonseca, pela generosidade na disponibilidade de alguns dos recursos computacionais, tanto de software como de hardware, fundamentais para a obtenção dos resultados práticos apresentados.

À minha esposa Angela, por todo o amor, carinho, suporte, compreensão e sacrifícios feitos durante os anos que duraram o período de aquisição dos créditos e elaboração desta dissertação. Sem dúvida a sua motivação e confiança na minha pessoa, foram um dos principais estímulos para a superação dos momentos difíceis surgidos durante esta jornada.

A meus pais, pelo exemplo de vida e retidão. Pela infinita compreensão com as minhas ausências e pouca atenção dispendida a eles, em especial neste último ano. A vida não é tão longa a ponto de se despediçar o convívio com os mais velhos.

Existem ainda várias outras pessoas, que indiretamente ou a distância, através de perguntas eventuais, conversas esparças ou sugestões ajudaram a manter vivo o entusiasmo e o rumo tomado neste trabalho. A todas elas, o meu muito obrigado.

RESUMO

O presente trabalho aborda os diferentes processos de desenvolvimento que levam à implementação de algoritmos de simulação e controle em FPGAs. Foram adotados como objeto de aplicação o modo nominal de operação de um satélite artificial estabilizado em três eixos com apontamento fino, e o controle de posição de um atuador eletro-hidráulico de uma aeronave. No processo de desenvolvimento exercitado, foram utilizados conceitos de modelagem visual, geração automática de código, migração para ambientes de co-simulação e emulação com características de tempo real. Foram desenvolvidos dois ambientes de co-simulação, baseados na execução concorrente e coordenada de um FPGA e um processador tradicional. Os resultados obtidos atestam a fidelidade e a vialibilidade na transição de um sistema modelado e validado em um ambiente virtual até o ambiente de co-simulação.

EMULATION AND CO-SIMULATION OF PMM ATTITUDE CONTROL AND AIRCRAFT ELECTRO-HYDRAULIC SYSTEMS USING FPGAS

ABSTRACT

This work addresses the different development processes that leads to the implementation of simulation and control algorithms into a FPGA. As case study two applications were selected: first, the three axis stabilization of an artificial satelite with fine pointing, in its Nominal operational mode; and second, the position control loop of an electro-hydraulic actuator for an aircraft. For the application of the development process, concepts of visual modelling, automatic code generation, migration to real-time co-simulation and emulation environments were used. Two co-simulation environments were developped, based on concurrent and coordinated execution of a FPGA and a standard processor. The achieved results show the accuracy and feasibility of the transition made between modelled system, validated in a simulated virtual environment, to its co-simulation counterpart.

SUMÁRIO

Pág.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

A_P	Área do pistão do atuador
dm	Partícula de massa do satélite
$\vec{H}_R$	Momento ângular da roda de reação.
$\dot{\vec{H}}_R$	Torque devido à roda de reação, ou torque de controle.
$\vec{H}_{RS}$	Momento ângular da roda de reação com relação ao satélite
$\vec{H}_S$	Momento ângular do satélite, com relação à origem dos eixos de coordenada do corpo.
I_S	Matriz de inércia do satélite
K_d	Ganho, ou coeficiente, derivativo do controlador PID
K_i	Ganho integral, ou coeficiente de integração, do controlador PID
K_p	Ganho proporcional do controlador PID
K_W	Ganho da roda de reação
L_C	Indutância da bobina do motor da servoválvula
M_P	Massa do pistão do atuador
P_A	Pressão na câmara A do atuador
P_B	Pressão na câmara B do atuador
P_S	Pressão da fonte hidráulica
Q_A	Fluxo hidráulico entre o port A da servoválvula e a câmara A do atuador
Q_B	Fluxo hidráulico entre o port B da servoválvula e a câmara B do atuador
Q_L	Fluxo hidráulico fornecido ao atuador
Q_{PUMP}	Fluxo máximo fornecido pela fonte hidráulica

$\vec{r}$	Vetor posição do elemento de massa *dm* em relação à origem
R_C	Resistência da bobina do motor da servoválvula
T_s	Período de amostragem de um sistema discreto
$\vec{T}_{Ext}$	Vetor torque, resultante dos torques perturbadores provenientes de efeitos ambientais
T_{Ext_x}	Componente de $\vec{T}_{Ext}$ na direção do eixo x
T_{Ext_y}	Componente de $\vec{T}_{Ext}$ na direção do eixo y
T_{Ext_z}	Componente de $\vec{T}_{Ext}$ na direção do eixo z
T_{Rmax}	Torque máximo da roda de reação
T_W	Constante de tempo da roda reação
ω_{Rmax}	Velocidade máxima da roda de reação
$\vec{\omega}_{RS}$	Vetor velocidade relativa das rodas de reação em relação ao satélite
$\vec{\omega}_S$	Vetor velocidade ângular do satélite
X_P	Posição do pistão do atuador
β	"bulk modulus" do óleo utilizado no sistema hidráulico
ϕ	Ângulo da terceira rotação ao redor do eixo x ("roll") do satélite com relação ao referencial VLHL
ψ	Ângulo da primeira rotação ao redor do eixo z ("yaw") do satélite com relação ao referencial VLHL
ζ	Coeficiente de amortecimento
θ	Ângulo da segunda rotação ao redor do eixo y ("pitch") do satélite com relação ao referencial VLHL

LISTA DE SIGLAS E ABREVIATURAS

ACE - Actuator Control Electronics

API - Application Program Interface

ASIC - Application Specific IC

CAD - Computer-Aided Design

CDFG - Control Data Flow Graph

DLL - Dynamic Loadable Library

DSP - Digital Signal Processor

DOS - Disk Operating System

E/S - Entrada/Saída

EDA - Electronic Design Automation

EHSV - Electro-Hydraulic Servo Valve

FDA - Filter Design Analysis

FPGA - Field Programmable Gate Array

FTMF - Função de Transferência de Malha Fechada

HDL - Hardware Description Language

HIL - Hardware-in-the-Loop

HLL - High Level Languages

I/O - Input/Output

IC - Integrated Circuit

IP - Intelectual Property

IAE - Integral Absolute-Error

ISE - Integral Square-Error

ITAE	- Integral-of-Time-multiplied Absolute-Error
NI	- National Instruments
OBC	- On-Board Computer
PC	- Personal Computer
PCI	- Peripheral Computer Interface: barramento de computadores pessoais
PMM	- Plataforma Multi-Missão
RIO	- Reconfigurable I/O
RTL	- Register-Transfer Level
RTW	- Real Time Workshop
SCAO	- Sistema de Controle de Atitude e Órbita
SIT	- Simulation Interface Toolkit
SOTR	- Sistema Operacional de Tempo Real
SSR	- Satélite de Sensoriamento Remoto
VHDL	- VHSIC HDL
VHSIC	- Very High Speed IC
VI	- Virtual Instrument: nome de um bloco funcional qualquer em LabVIEW
VLHL	- Vertical Local e Horizontal Local

CAPÍTULO 1

INTRODUÇÃO

O estágio atual de desenvolvimento na área de Microeletrônica permite aos seus projetistas colocar uma quantidade impressionante de componentes dentro de uma única pastilha semicondutora. Com isso, sistemas completos, onde antigamente seriam necessários diversos cartões eletrônicos e, mais recentemente, seriam integradas diversas funções em poderosos microprocessadores, têm agora uma nova fronteira, podendo ser integrados em único componente, chamado "Field Programmable Gate Array", ou FPGA.

Um FPGA pode ser entendido como uma espécie de circuito integrado de propósito geral, onde seus componentes e blocos funcionais são personalizados e interligados para atender a uma necessidade específica.

Assim, por exemplo, se uma determinada aplicação necessitar de vários coprocessadores aritméticos para executar um algoritmo em paralelo, é possível personalizar o componente para tal. Em uma das configurações possíveis, pode-se configurar o FPGA como um processador tradicional, de forma a manter a compatibilidade com dispositivos convencionais.

No entanto, a forma de se utilizar tal componente na sua plenitude traz uma mudança de paradigma para as fases de arquitetura e implementação dos desenvolvimentos de sistemas, uma vez que o FPGA é baseado em processamento paralelo, enquanto em processadores convencionais as instruções são executadas seqüencialmente.

O processo tradicional de desenvolvimento de tais sistemas ocorre em estágios, com ferramentas específicas para o projeto de controle, engenharia de software,

aquisição de dados e testes. Tal processo tem sido profundamente modificado com a introdução de pacotes de produtos tais como MATRIXx/SystemBuild e Matlab/Simulink. Esses pacotes permitem uma rápida e suave transição de um estágio para o outro dentro do mesmo ambiente integrado, trazendo enormes impactos na velocidade de execução desse processo.

Com um pacote como Matlab/Simulink, pode-se projetar, simular, analisar e depurar um modelo de sistema. A seguir, através do "Real Time Workshop" – RTW – uma de suas ferramentas, pode-se gerar automaticamente código com características de tempo real, em uma linguagem de alto nível, como C. Normalmente, essas aplicações são desenvolvidas tendo por premissa a adoção de algum tipo de sistema de tempo real como base de execução. Assim, o código gerado está virtualmente pronto para ser embarcado, tipicamente usando um microprocessador como unidade de processamento.

Para o caso de se desejar embarcar esse projeto em um componente como o FPGA, em função da arquitetura do componente não ser a mesma dos microprocessadores, surgem algumas dificuldades: o código foi construído essencialmente para uma máquina sequencial uma vez que, apesar da evolução de ambientes como Matlab/Simulink, a geração da implementação física para um dispositivo de hardware não é um processo trivial (BANERJEE, 2003; KRUKOWSKI, 1999), envolvendo diversas etapas até a obtenção de um código sintetizável em VHDL.

Normalmente, o fabricante do FPGA possui alguma ferramenta que permite ao usuário passar por essas etapas de uma forma integrada, mas o problema principal ainda persiste: converter uma funcionalidade descrita na forma Matlab/Simulink para VHDL.

Um caminho alternativo é usar um ambiente onde se possa definir um sistema no nível mais alto possível e, a partir daí, obter-se o código sintetizável para FPGA.

Dessa forma, algumas das dificuldades listadas anteriormente podem ser minimizadas, ou até suprimidas.

Uma vez embarcado o código desejado no FPGA, em um ambiente de co-simulação, é possível integrá-lo com outras partes de uma simulação que por sua vez está sendo executada em um processador genérico. Esta será a abordagem deste trabalho. Assim, o ambiente de co-simulação adotado possuirá como um dos integrantes, um hardware dedicado, nesse caso um FPGA reconfigurável.

1.1 Objetivo

Este trabalho tem por objetivo desenvolver um ambiente de co-simulação de sistemas de controle aeroespaciais, implementados parcialmente em Field Programmable Gate Arrays – FPGA – e parcialmente em processadores tradicionais. Para tanto, ele:

1) Estuda diversas possibilidades de fluxo de desenvolvimento, desde ferramentas orientadas a projeto baseado em modelos ("Model Based Design" – MBD), até a sua síntese utilizando FPGA como elemento de implementação, selecionando pelo menos uma alternativa;

2) Estuda e seleciona um Sistema de Controle de Atitude e Órbita – SCAO – de satélite, visando a sua simulação em um ambiente distribuído;

3) Implementa a simulação de uma roda de reação e seu controle, embarcando-os em um FPGA;

4) Integra-os em um ambiente de co-simulação, onde a dinâmica e atitude do satélite estarão sendo executados usando um sistema operacional com características de tempo real, e analisa os seus resultados;

5) Estuda e seleciona um sistema de controle de atuadores eletro-hidráulicos ("Actuator Control Electronics – ACE) para aplicação aeronáutica em sistemas

Fly-By-Wire (FBW), com ênfase no atuador e seu controlador, visando a sua simulação em um ambiente distribuído;

6) Integra-os nesse ambiente distribuído, onde a dinâmica da aeronave estará sendo executada usando um sistema operacional de tempo real, e analisa os seus resultados.

1.2 Motivação

O uso de FPGAs como alternativa de implementação de sistemas de controle tem crescido rapidamente, devido principalmente ao aumento da sua capacidade de processamento. Junte-se a isso o desenvolvimento de ferramentas integradas onde pode-se exercitar um ciclo completo de desenvolvimento, desde a sua concepção à sua implementação, tem-se assim dois elementos fundamentais que suportam a transformação por que passa a Engenharia de Controle atualmente.

Neste contexto, é de suma importância a existência de estudos a respeito da viabilidade da aplicação desses conceitos no âmbito de aplicações aeroespaciais, tônica deste trabalho.

Os exemplos de aplicação escolhidos procuram realçar algumas das vantagens na sua adoção. O caso espacial permite avaliar e validar o ciclo de desenvolvimento, uma vez que essa aplicação já foi extensamente abordada em trabalhos anteriores.

Já no caso aeronáutico, quando se deseja um modelo com razoável fidelidade, acaba-se adotando o próprio sistema físico, ao invés de algum tipo de simulação numérica, em virtude das características desse sistema, principalmente no que tange às não linearidades da planta e requisitos apertados para o tempo de resposta do controlador. Um ambiente de co-simulação em tempo real permite mudar essa tradição, trazendo como benefício a simplificação e o barateamento na implantação de um laboratório de hidráulica para tais estudos.

1.3 Organização

Este trabalho procura abordar o tema proposto ordenado em 6 capítulos, sumarizados a seguir.

No Capítulo 1 é feita uma breve introdução a respeito das principais tecnologias envolvidas e a motivação na escolha do assunto proposto.

No Capítulo 2 são apresentados os conceitos e tecnologias necessários para o desenvolvimento do trabalho, bem como as referências bibliográficas que suportam tais conceitos e tecnologias.

No Capítulo 3 é feita a escolha do método de desenvolvimento a ser adotado, o problema espacial é formulado, os componentes do problema considerados relevantes para o trabalho são modelados, e é obtida uma estimativa inicial para os ajustes dos controladores de atitude do satélite.

No Capítulo 4 é construído um ambiente para a simulação do problema espacial em tempo virtual, usando os modelos obtidos no Capítulo 3. A partir da execução desse modelo, novos valores para os ganhos dos controladores são obtidos, e o algoritmo é manipulado de forma a minimizar o esforço de migração para o FPGA.

No Capítulo 5 uma parte do modelo do problema espacial é migrada para um FPGA que, junto com um processador tradicional, forma o ambiente de co-simulação com características de tempo real. Duas arquiteturas baseadas nessa estrutura são apresentadas e utilizadas. Resultados da simulação em tempo real do modelo espacial são obtidos e comparados com os do ambiente de simulação em tempo virtual.

No Capítulo 6 o processo de desenvolvimento adotado nos Capítulo 3 a 5 é repetido para uma aplicação aeronáutica. Um sistema composto por um atuador hidráulico, seu servoacionamento e controle é modelado, simulado em tempo virtual, e migrado para um ambiente de co-simulação em tempo real.

O Capítulo 7 apresenta a conclusão do trabalho e sugestões para futuros desdobramentos.

CAPÍTULO 2

CONCEITOS BÁSICOS E REVISÃO BIBLIOGRÁFICA

A elaboração de um ambiente de co-simulação para sistemas aeroespaciais usando FPGAs é baseada na integração de várias tecnologias e conceitos, provenientes de diversas áreas. Os tópicos abordados neste capítulo referem-se aos conceitos e tecnologias necessários para a construção desse ambiente.

2.1 Sistemas de Referência

Para conceituar o exemplo espacial, normalmente são utilizados 3 sistemas de referência: um fixo em relação ao corpo do veículo espacial; outro fixo no espaço inercial; e por último um definido em relação à órbita, ou seja, nem fixo em relação ao veículo, nem ao espaço inercial. Esses sistemas são os adotados neste trabalho.

2.1.1 Referencial Inercial

Conforme Arantes (2005) o sistema "Earth Centered Inertial" – ECI – é considerado um referencial de direções inerciais. Esse referencial "inercial" (O, X, Y, Z) definido tem origem O no centro da Terra. O eixo Z aponta na direção do pólo norte geográfico, a aproximadamente 1° da Estrela Polar. O eixo X aponta na direção do Equinócio Vernal γ, definido como o ponto da Eclíptica, ou plano orbital aparente do Sol ao redor da Terra que cruza o Equador na direção sul para norte, no primeiro dia da primavera. O eixo Y é encontrado usando a regra da mão direita, completando o sistema dextrógiro.

De acordo com Moreira (2006) esse sistema de referência não pode ser considerado verdadeiramente inercial, uma vez que ele não é fixo em relação às estrelas na proximidade do Sol. A força gravitacional da Lua e do Sol provocam um torque que resulta em uma vagarosa rotação do eixo da Terra em relação à eclíptica. Esse fenômeno é conhecido como precessão dos equinócios e tem período aproximado de

28000 anos, ou 50 segundos de arco por ano. Mas, para os objetivos deste trabalho, essa pequena discrepância é desprezível, podendo-se considerar o sistema ECI como verdadeiramente inercial.

2.1.2 Referencial Orbital

O sistema orbital adotado neste trabalho é conhecido como VLHL (Vertical Local e Horizontal Local), definido por um sistema de coordenadas (O′, x, y, z) com origem no centro de massa O′ do satélite e girante no plano da órbita do satélite. O eixo z aponta na direção do centro da Terra, o eixo y aponta na direção normal ao plano da órbita. O eixo x é obtido pela regra da mão direita, e coincide com a direção do vetor velocidade orbital linear, para uma órbita circular (WERTZ, 1978, segundo MOREIRA, 2006).

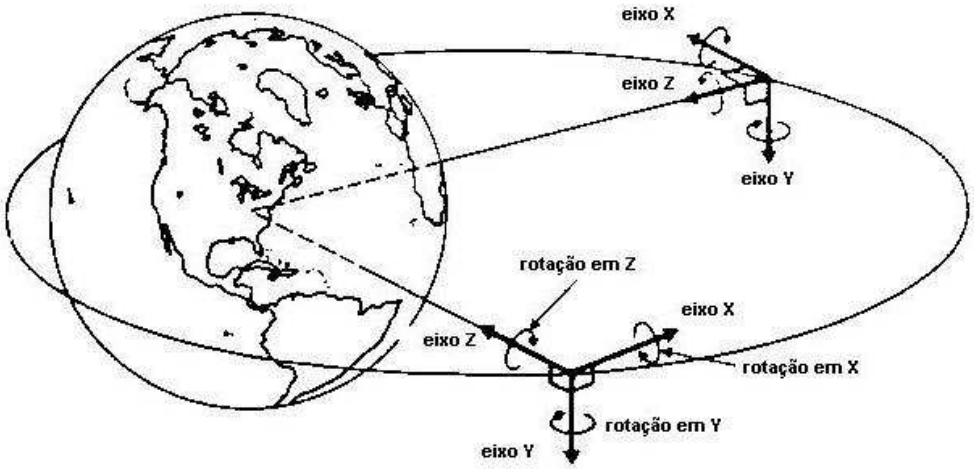


FIGURA 2.1 – Sistema de referência VLHL.
FONTE: Moreira (2006).

2.1.3 Referencial do Satélite

O referencial do satélite, ou do corpo, é um sistema de coordenadas com origem no centro de massa O′ do satélite e os eixos são escolhidos de forma a serem coincidentes com os eixos principais de inércia. Para estudos de satélites estabilizados em três eixos, é prático renomear os eixos para "roll", "pitch" e "yaw". Ou seja, "roll" está nominalmente alinhado com x_0, "pitch" com y_0, e "yaw" com z_0.

2.2 A Plataforma Multi-Missão

A Plataforma Multi-Missão (PMM) é um conceito em arquiteturas de satélites que consiste em reunir, em uma única plataforma, equipamentos essenciais à operação do satélite, independente de sua órbita e missão específica.

Nessa arquitetura, existe uma separação física entre a plataforma e o módulo de carga útil, permitindo que ambos possam ser desenvolvidos, construídos e testados separadamente, antes da integração e testes finais do satélite (GOBATO, 2007). A Figura apresenta uma ilustração da PMM, com seus painéis solares abertos.

FIGURA 2.2 – PMM com os painéis solares abertos.
FONTE: Moreira (2006).

A PMM é composta dos seguintes subsistemas:

- Estrutura: provê suporte mecânico para todos os outros subsistemas, equipamentos e acessórios, e módulo de carga útil;

- Suprimento de Energia Elétrica: converte a energia solar incidente nos painéis fotovoltaicos em energia elétrica, armazena-a em baterias e fornece essa energia para diversos equipamentos da plataforma e carga útil;

- Controle de Temperatura: provê distribuição de calor de forma que os equipamentos a bordo operem dentro das suas faixas especificadas de temperatura;

- Controle de Atitude e Gerenciamento de Dados: provê controle de atitude e órbita, estabilizado em três eixos, permitindo apontamento para a Terra, Sol alguma posição Inercial. Também é responsável pelo processamento de dados e provê capacidade de armazenamento, através do computador de bordo;

- Propulsão: provê aquisição e manutenção de órbita através de um tipo de propulsor monopropelente a hidrazina;

- Telemetria e Telecomando: provê comunicação entre a PMM e estações de Terra.

O subsistema de Controle e Atitude e Gerenciamento de Dados – ACDH – possui uma arquitetura baseada em um computador de bordo modular, o qual irá comandar os equipamentos de controle de atitude, comunicar-se com outros subsistemas e com a carga útil. A composição do subsistema é:

- Computador de Bordo ("On Board Computer – OBC");

- Sensores

    1. **Magnetômetros** (2): cada um fornece medição do campo magnético em três eixos;

    2. **Unidade Inercial**: provê velocidade angular nos três eixos principais de inércia do satélite;

    3. **Sensores Solares** (8): provê informação suficiente para determinação da direção do Sol em três eixos, com cobertura total do céu;

4. **Sensores de Estrelas** (2): cada um fornece informação de atitude em três eixos;

5. **GPS** (2): cada uma das unidades é composta por um receptor e suas antenas, provendo hora, posicionamento e velocidade do satélite.

- Atuadores

1. **Bobinas Magnéticas** (3): provê torque magnético em três eixos;

2. **Rodas de Reação** (4): provê controle de atitude em três eixos.

O comportamento do satélite, durante a sua vida útil, é descrito por diferentes **Modos de Operação**, os quais estão associados com a configuração de seus equipamentos e seu monitoramento. Os **Modos de Operação** são:

- Vôo

1. Modo de Inicialização ("Start Model" – STM);

2. Modo de Contingência ("Continency Mode" – COM);

3. Modo de Navegação Fina ("Fine Navigation Mode" – FNM);

4. Modo Nominal ("Nominal Model" – NOM);

5. Modo de Dessaturação das Rodas através dos Propulsores ("Wheels Desaturation Mode" – WDM);

6. Modo de Correção de Órbita ("Orbit Correction Mode" – OCM);

7. Modo de Correção de Órbita Backup ("Orbit Correction Mode-Backup" – OCMB).

- Solo

1. Modo Desligado (OFM);

2. Modo de Integração e Testes (ITM).

No Modo Nominal, a PMM será configurada para que sua carga útil cumpra sua missão; esse será o modo utilizado neste trabalho. A atitude do satélite é mantida em direção ao alvo. Assim, a atitude, bem como a sua taxa de variação, devem ser controladas nos três eixos, atendendo aos seguintes requisitos:

1. Precisão de apontamento: < 0,05° (3σ);

2. Deriva ("drift"): < 0,001°/s;

3. Determinação de atitude: 0,005° (3σ);

4. Desvio ("off pointing") de até 30° em 180s.

2.3 Caso Aeronáutico: Superfície de Controle usando EHSV

Uma prática comum ao estudar a mecânica de vôo de uma aeronave é representá-la como corpo rígido, definido por um conjunto de eixos alinhados ao corpo, conforme a Figura 2.3. As forças e momentos atuantes sobre o veículo podem então ser modeladas usando essa referência. Ao atuar sobre as superfícies de controle, um sistema de comandos de vôo permite controlar essas forças e momentos, agindo sobre esse veículo. Consequentemente, é possível controlar as suas acelerações, velocidades, translações e rotações.

Uma das principais preocupações do meio aeronáutico sempre esteve relacionada com a capacidade do piloto em realizar manobras, de forma que a aeronave fosse estável e não o sobrecarregasse, tanto física como mentalmente. Assim, com o passar dos anos, foram sendo adotados mecanismos hidráulicos de potenciação dos comandos do piloto, de forma a ajudá-lo na tarefa de comandar as superfícies de controle da aeronave.

Essa ajuda pode ser tanto do ponto de vista do esforço físico envolvido, como através de funções auxiliares que auxiliam na controlabilidade, tais como compensadores e controladores, cujas saídas se somam à do piloto na tarefa de comandar as superfícies (STEVENS, 2003; FIELDING, 2000).

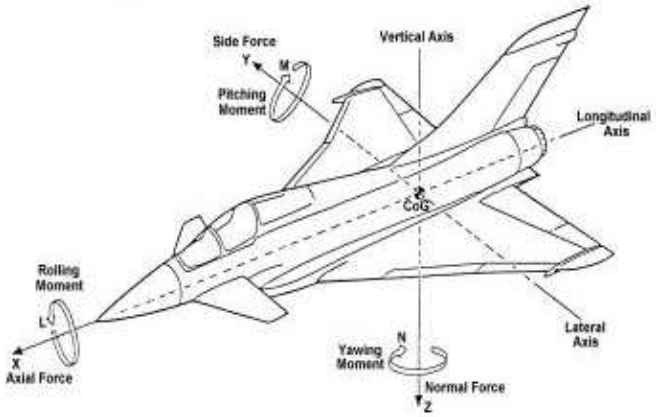


FIGURA 2.3 – Aeronave: sistema de coordenadas do corpo.
FONTE: Fielding (2000).

As primeiras gerações de sistemas de comando de vôo eram baseadas em um acoplamento mecânico direto entre os controles do piloto e as superfícies de controle, chamado "Fly-By-Cable".

Atualmente, modernos sistemas "Fly-By-Wire", ou FBW, são usados para estabilizar uma aeronave, seja por causa dos objetivos mencionados anteriormente, seja por ela ser inerentemente instável. O termo "Fly-By-Wire" implica um sistema de controle transmitindo eletricamente os comandos executados pelo piloto para computadores dedicados que os processam e acionam atuadores eletro-hidráulicos para movimentar as superfícies aerodinâmicas de controle (DEMARCHI, 2005).

Para fins deste trabalho, a parte do sistema de comando de vôo a ser detalhada e estudada em maior profundidade refere-se ao atuador e sua malha de controle de posição.

Um sistema hidráulico servo assistido pode gerar forças bastante elevadas, possui um reduzido tempo de resposta, e exibe uma alta relação potência/peso (POLEY, 2005). Por outro lado esse sistema, especialmente quando analisado na sua totalidade, exibe um comportamento não linear significativo, associado a características de fluxo/pressão, compressibilidade de óleos, efeitos de transmissão não lineares, fricção, dentre outros (CHATZAKOS, 2003).

A Figura 2.4 mostra a estrutura típica de um sistema hidráulico servo assistido em malha fechada típica, consistindo em: fonte hidráulica, válvula de controle de fluxo, atuador linear, sensor de posição e um servo controlador.

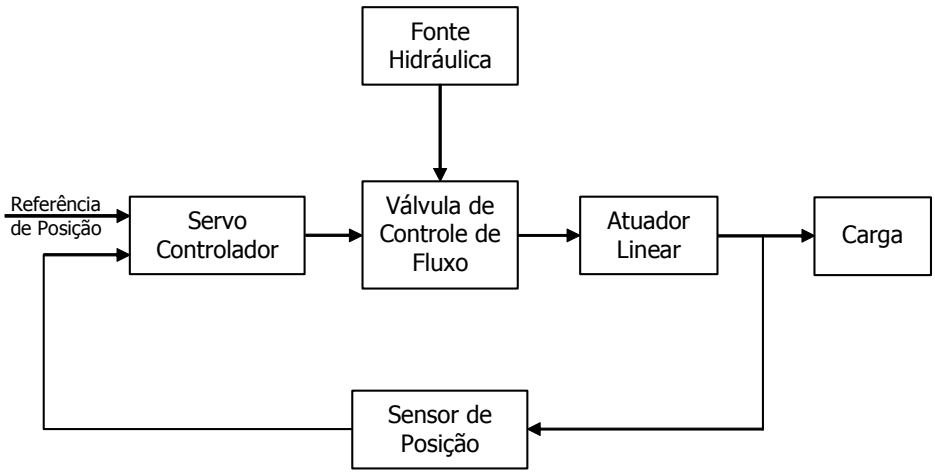


FIGURA 2.4 – Diagrama de blocos de um controle de posição para um sistema hidráulico.

Um sistema hidráulico necessita de uma fonte de fluido pressurizada, normalmente um óleo mineral. A escolha da pressão do sistema depende de vários fatores. Baixa pressão significa menos vazamentos, mas implica utilizar componentes maiores para produzir uma dada quantidade de força. Sistemas de alta pressão padecem de um vazamento maior mas, por outro lado, possuem desempenho dinâmico melhor e são menores e mais leves: vantagens significativas para aplicações móveis e aeronáuticas. Nesses casos, a pressão do sistema normalmente utilizada é de 3000 psi (POLEY, 2005).

A válvula de controle de fluxo – EHSV – atua como um transdutor eletro-hidráulico de alto ganho, cuja entrada é um sinal de corrente elétrica e a saída um fluxo variável de óleo. Neste trabalho, a válvula considerada é uma de duplo estágio, sendo o primeiro estágio de bocal-palheta e o segundo estágio uma válvula carretel de centro crítico e 4 vias. Uma válvula típica desta categoria está apresentada na Figura 2.5.

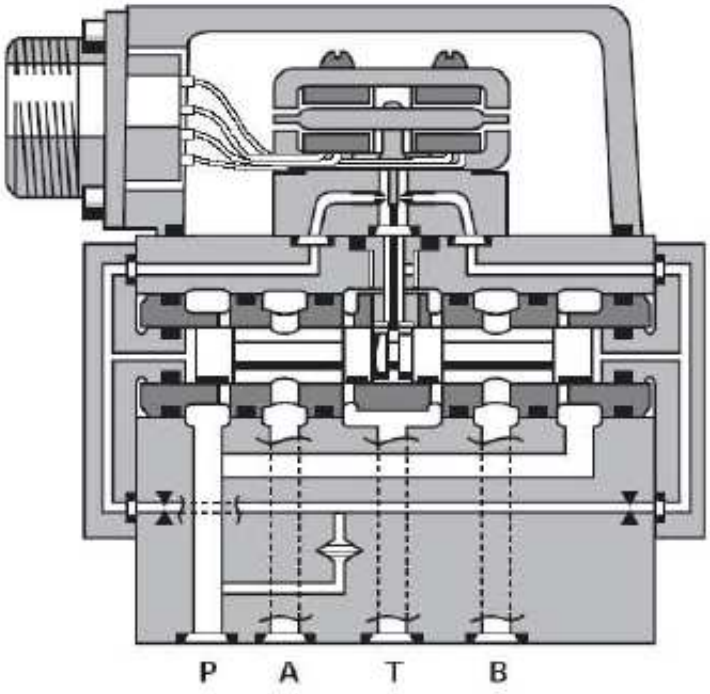


FIGURA 2.5 – Corte de uma válvula de duplo estágio.
FONTE: Moog (2000).

O movimento do carretel modifica o tamanho da abertura dos orifícios, alterando o diferencial de fluxo de óleo entre as câmaras da válvula. A Figura 2.6 mostra uma configuração típica considerada no segundo estágio da válvula.

Um atuador hidráulico é um dispositivo que converte energia hidráulica em força mecânica, ou movimento. Para fins deste trabalho, o atuador é considerado linear, e capaz de controlar o movimento em duas direções, através da pressão exercida em ambos os lados do pistão.

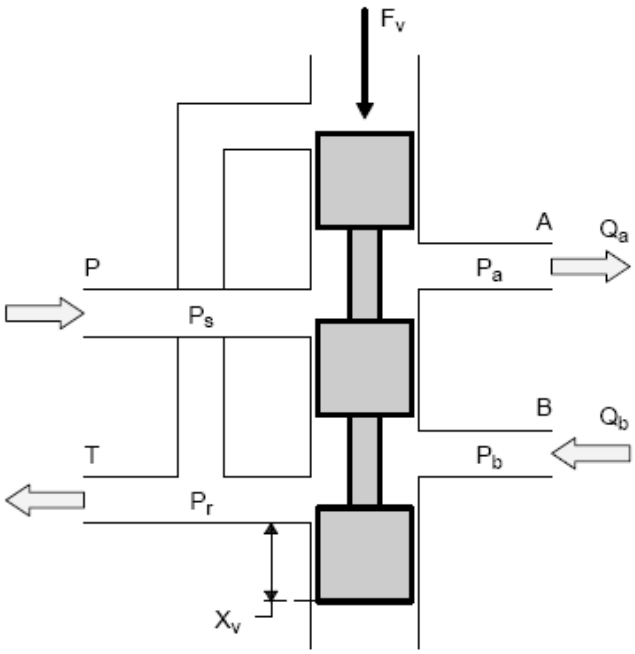


FIGURA 2.6 – Diagrama esquemático da válvula carretel de 4 vias.
FONTE: Poley (2005).

A posição do pistão é determinada pelas pressões nas câmaras do atuador, e pode ser ajustada através da injeção de fluido em um dos lados e retirada pelo outro lado. No caso da Figura 2.7, o fluido entrando pelo lado A e ao mesmo tempo saindo pelo lado B, gera uma pressão líquida que exerce uma força na área ativa do pistão, movendo-o para a direita, conforme indicado.

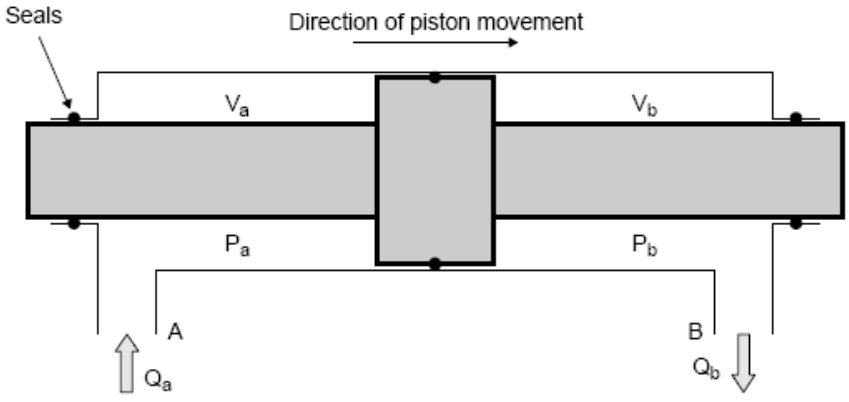


FIGURA 2.7 – Atuador linear de movimento para ambas as direções.
FONTE: Poley (2005).

O controlador da figura 2.4, nesta aplicação conhecido como ACE – "Actuator Control Electronics" – compara o sinal de realimentação de posição do atuador com a sua referência para determinar o sinal de comando para a EHSV. Na aplicação em questão, essa referência provém do piloto e das malhas externas de controle. Por sua vez, o controlador gera um sinal de comando para a válvula de controle de fluxo, que ajusta o fluxo de óleo pressurizado para mover o atuador até a posição desejada.

2.4 Controladores Digitais

Controladores digitais têm substituído equivalentes analógicos, implementados tanto eletrônica como pneumática ou hidraulicamente, em conseqüência da capacidade de miniaturização, baixo custo e incorporação de elementos complexos provenientes do campo da microeletrônica, em especial aplicados a microprocessadores e microcontroladores.

Em controle digital baseado em microprocessadores, os sinais do mundo externo são amostrados e digitalizados, resultando em sinais discretos que são quantizados em amplitude e tempo. Esse processo de amostragem é executado periodicamente, com período T_0, normalmente usando um multiplexador que conecta o sinal amostrado a um conversor analógico-digital (A/D). Esse valor digitalizado de entrada é processado e gerada uma saída, que normalmente é enviada a um conversor digital-analógico (D/A) (ISERMAN, 1989). Como conseqüência direta do processo de discretização no tempo, existe a introdução de atrasos para qualquer controlador implementado com essas características.

Essa forma (digital) de implementação de controle, tem uma qualidade muito importante: a flexibilidade e a quantidade de alternativas de abordagens de controle, permitindo a adoção de técnicas de controle moderno. Por outro lado, reforça a questão sobre qual algoritmo de controle é mais adequado para um problema específico (ISERMAN, 1989).

De qualquer forma, em virtude da relativa facilidade e do grande número de projetos já desenvolvidos, normalmente uma abordagem inicial para um controle digital é a sua conversão a partir um equivalente analógico. Dentre as diversas conversões possíveis (OPPENHEIM, 1989), a transformação bilinear, ou Tustin, é uma das mais populares.

Uma vantagem desse reprojeto baseado no equivalente analógico é que o período de amostragem T_0 pode ser selecionado após a conclusão do projeto analógico. Porém, conversões baseadas em transformações como a de Tustin são aproximações, e portanto, o período de amostragem T_0 deve ser pequeno – comparativamente às constantes de tempo analógicas – para que o desempenho dos controladores analógico e digital sejam equivalentes (STEVENS, 2003).

2.5 Field Programmable Gate Array – FPGA

FPGAs surgiram em 1984, na Xilinx, inicialmente para resolver dificuldades na criação de circuitos integrados customizados, usados na integração de componentes complexos de sistemas computacionais. Com o avanço da microeletrônica, os FPGAs tornaram-se eles mesmos complexos, incluindo uma série de recursos impensáveis na época de sua concepção. Hoje eles podem ser entendidos como uma forma de se organizar circuitos eletrônicos diferente dos microprocessadores e, dependendo da aplicação, com desempenho superior.

Um microprocessador é um agregado de subsistemas, que são baseados em blocos lógicos (e por sua vez baseados em circuitos eletrônicos básicos). Assim, em um processador, encontram-se funções ou subsistemas para processamento de lógica aritmética, registradores, relógio e barramentos de dados, dentre outros. Cada um desses subsistemas é composto por um grande número de blocos lógicos, agrupados para um objetivo específico.

Em um FPGA, inicialmente os blocos lógicos encontram-se desagrupados. Durante o processo de configuração, ou programação, de um FPGA é que esses blocos lógicos são agrupados para executarem alguma tarefa específica.

A estrutura de um microprocessador normalmente é bastante padronizada e genérica, porque existe o pressuposto de que um outro componente, o *software*, irá finalmente ordenar o processo de execução de uma série desses subsistemas, gerando como saída o resultado esperado por algum processamento de dados.

Uma classificação usualmente adotada para arquiteturas de microprocessadores (DESCHAMPS, 2006) está relacionada com acesso à memória. Nesse caso, existem duas classes fundamentais, representadas na Figura 2.8: Von Neumann e Harvard. Na primeira, a memória é compartilhada entre instruções (programa) e dados. Instruções e dados precisam ser acessados em ordem sequencial (conhecido como gargalo de Von Neumann), limitando o desempenho. Por outro lado, a arquitetura Harvard adota memórias diferentes para instruções e dados, utilizando barramentos dedicados para cada um deles. Instruções e dados podem, assim, ser acessados simultaneamente.

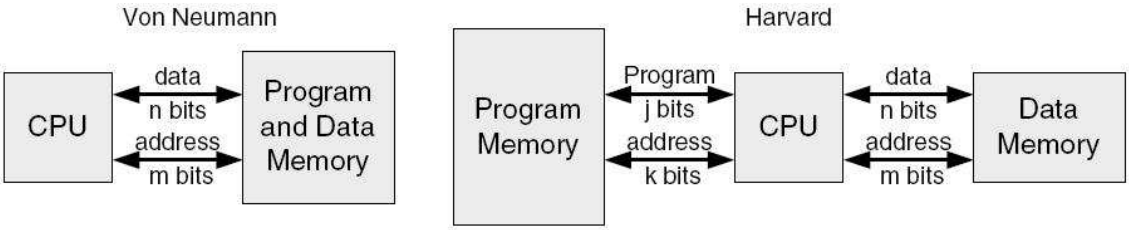


FIGURA 2.8 – Arquiteturas Von Neumann e Harvard.
FONTE: Deschamps (2006).

A arquitetura Von Neumann é mais simples que a de Harvard. A primeira possui uma utilização mais eficiente de memória, e é adotada pela maioria dos processadores de uso geral. Por outro lado, a maioria dos DSPs e microcontroladores usa a arquitetura Harvard por causa da alta capacidade de movimentar dados.

Ao se usar um FPGA para processamento desse mesmo dado, o pressuposto é outro. Como será configurado para executar um algoritmo específico, pode-se personalizar os seus subsistemas, definindo como eles se interconectam para o processamento dos dados. Não há software nessa solução; o próprio FPGA, em virtude da sua capacidade de configuração, é capaz de resolver o problema sozinho por meio dos seus blocos lógicos. Em outras palavras, quando um FPGA é configurado, seus circuitos internos são interconectados de forma a criar uma implementação por *hardware* de uma aplicação em *software*, que também poderia ser executada por um microprocessador.

A Figura 2.9 mostra os principais tipos de componentes básicos de um FPGA, a serem organizados durante o processo de configuração do componente, para implementar alguma aplicação específica.

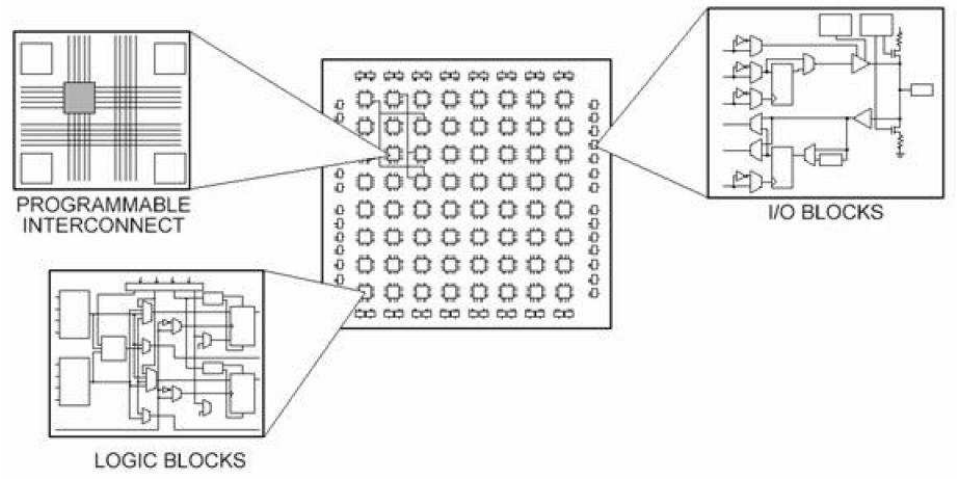


FIGURA 2.9 – Estrutura de um FPGA.
FONTE: Xilinx (2007).

FPGAs podem ser entendidos como enormes regiões de portas lógicas programáveis, com capacidade para serem organizadas em diversos ramos paralelos de processamento, onde as operações não precisam competir por recursos compartilhados.

Essa forma de execução gera um aumento no desempenho e determinismo que a maioria das soluções por software, baseadas em processadores convencionais não consegue atingir. Um processamento de dados sendo executado em um FPGA não terá as características de "jitter" associadas à execução por software e priorização de processos, típicos da maioria dos sistemas operacionais e também presentes em sistemas operacionais de tempo real, ainda que em grau bem menor (NI, 2006).

Para compreender melhor essas características de desempenho, deve-se considerar a forma com que o código é sintetizado em circuitos digitais. Síntese é o processo de tradução a partir de uma linguagem de programação em alto nível para implementação física. Para qualquer pedaço de código sintetizável, seja esse código escrito textual ou graficamente, existirá um circuito esquemático correspondente, descrevendo como blocos lógicos estarão interconectados (NI, 2006).

2.5.1 Fluxo de Desenvolvimento de Aplicações para FPGAs – Visão Componente

Dentre as alternativas para o fluxo de desenvolvimento de uma aplicação embarcada em FPGA (LALL, 2005), a mais tradicional possui uma visão bastante orientada ao próprio componente que se deseja projetar. Nesse nível de visão, a prioridade é a alocação e distribuição dos circuitos eletrônicos do componente, ou seja, orientada para hardware.

Essa é a abordagem que normalmente os fabricantes oferecem, algumas vezes gratuitamente, como é o caso das empresas Xilinx e Altera. Nesse contexto, ao se referir a uma linguagem de alto nível, ela é textual (HDL – VHDL ou Verilog) com alguns recursos de estruturação similar a sub-rotinas, ou esquemática, onde procura-se projetar a aplicação da mesma forma que em projetos de circuitos eletrônicos.

A Figura 2.10 apresenta um fluxo de desenvolvimento padrão para FPGAs. As sucessivas fases desse processo são descritas a seguir:

- Entrada do Projeto (“Design entry”): criação de arquivos de projeto, usando editor esquemático, ou HDL (Verilog ou VHDL);

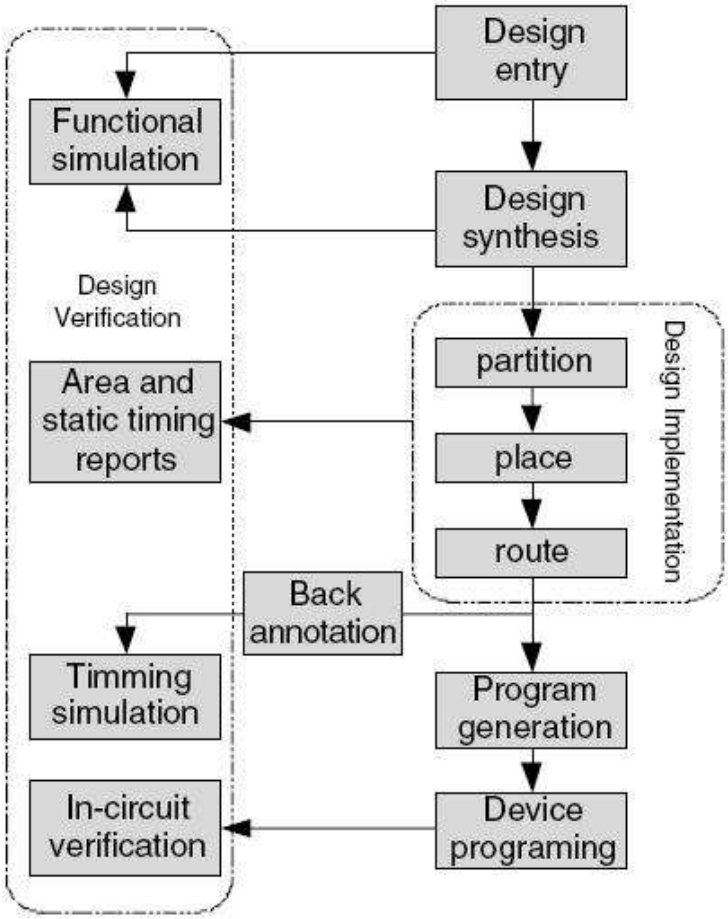


FIGURA 2.10 – FPGA: Fluxo de desenvolvimento.
FONTE: Deschamps (2006).

- Síntese do Projeto (“Design synthesis”): processo iniciado a partir de um alto nível de abstração lógica, normalmente escrito em Verilog ou VHDL, e que gera automaticamente um nível mais baixo de abstração, usando bibliotecas de primitivas;

- Partição (“Partition” ou Mapeamento): processo de atribuição de cada elemento lógico a um elemento físico que implementa essa função lógica;

- Alocação (“Place”): mapeamento da lógica em posições específicas dentro do FPGA;

- Roteamento (“Route”): interligação da lógica mapeada;

- Geração do Programa (“Program generation”): um arquivo binário é gerado para programar o dispositivo;

- Programação do Componente (“Device programing”): “download” do arquivo para dentro do FPGA

- Verificação do Projeto (“Design Verification”): simulação para verificar as funcionalidades especificadas. A simulação pode ser feita em diversos níveis, por exemplo, uma simulação funcional ou comportamental não leva em consideração atrasos do componente, ou das interconexões. Outros relatórios dessa fase são também gerados, para verificar resultados adicionais da implementação, tais como freqüência máxima e utilização dos recursos.

A partição, alocação e roteamento são normalmente agrupadas em um único termo, denominado Implementação do Projeto.

É importante observar que, no fluxo apresentado, uma vez executada a fase de Entrada de Projeto, toda a sucessão de fases é realizada automaticamente pelas ferramentas.

2.5.2 Hardware Description Language (HDL)

Quando se usa o termo HDL para designar uma linguagem, formalmente existem duas alternativas – VHDL e Verilog – apesar da primeira ser muito mais difundida que a segunda. Segue uma breve descrição a respeito delas.

VHDL é a sigla para “Very High Speed Integrated Circuits” – VHSIC – “Hardware Description Language” – HDL. É uma linguagem para descrever sistemas digitais que surgiu no começo dos anos 80, a partir de um programa do Departamento de Defesa

dos Estados Unidos, com a finalidade de descrever a estrutura e funcionalidade dos circuitos integrados (CHU, 2006). O objetivo era documentar o comportamento dos ASICs que estavam sendo incluídos nos equipamentos fornecidos, ou seja, foi desenvolvido como uma alternativa para enormes e complexos manuais, sujeitos a detalhes específicos de implementação (WIKIPEDIA, 2008).

A idéia de ser capaz de simular essa documentação era tão atrativa que logo surgiram Simuladores Lógicos que podiam ler arquivos VHDL. O próximo passo foi o desenvolvimento de ferramentas de Síntese Lógica, que liam o arquivo VHDL e geravam como saída a definição da implementação física do circuito (WIKIPEDIA, 2008).

Foi rapidamente adotado pelo IEEE nos EUA – através do padrão IEEE Std 1076-1987, e em outros países. Atualmente, o padrão IEEE Std 1076-2002 é o mais usado (YABARRENA, 2006). Esta linguagem permite ao desenvolvedor (CHU, 2006):

- Descrever a estrutura de um sistema, especificando como é decomposto em sub-projetos, e como eles estão interconectados;
- Especificar a funcionalidade dos projetos usando uma sintaxe parecida com C;
- Simular o projeto antes de programar o componente, de forma que possa ser possível comparar enfoques alternativos e testá-los, sem os atrasos e custos associados a múltiplas prototipações.

VHDL é uma linguagem geral de programação, similar ao C, com extensões para modelar fluxos de execução sequencial e concorrente. Em uma primeira aproximação, VHDL pode ser considerada uma combinação de duas linguagens: uma descrevendo a estrutura do circuito integrado – descrição estrutural – e outra o seu comportamento usando construções algorítmicas – descrição comportamental (KRUKOWSKI, 1999).

Verilog é outra alternativa de programação para FPGAs. Foi inventada por Phil Moorby e Prabhu Goel em 1985 na "Automated Integrated Design Systems" (depois renomeada para "Gateway Design Automation"), como uma linguagem de modelagem de hardware. "Gateway Design Automation" foi posteriormente adquirida pela "Cadence Design Systems" em 1990, que é a atual detentora dos direitos da linguagem (WIKIPEDIA, 2008).

Em meados dos anos 90, a Cadence decidiu tornar Verilog de domínio público, submetendo-a a seguir ao IEEE, que publicou a norma IEEE Std 1364-1995, comumente chamada de Verilog-95. A versão atual da norma é IEEE Std 1364-2005.

2.5.3 Fluxo de Desenvolvimento de Aplicações para FPGAs – visão Sistema

Como consequência do aumento da densidade de circuitos nos FPGAs, tornou-se possível usá-los como destino para execução de algoritmos mais complexos. Com isso, fluxos de desenvolvimento como o descrito na seção 2.5.1, baseado em HDLs tornaram-se inadequados em função da limitação de linguagens do tipo VHDL ou Verilog. Estudos têm sido feitos a partir do final da década de 90, com linguagens que permitam um nível de abstração mais alto, bem como permitir o encapsulamento de um sistema completo, desde a sua modelagem até o hardware (SHANBLATT, 2005).

Ao mesmo tempo, ambientes como MATLAB/Simulink evoluíram de ferramentas utilizadas por pesquisadores para uma linguagem voltada a desenvolvedores de sistemas, e bastante popular nos ambientes aeroespaciais. Essa popularidade foi alcançada principalmente em virtude da capacidade de modelagem através de uma interface gráfica orientada a blocos funcionais, e posterior geração automática de código, em C.

Nesse cenário, diversos fluxos de desenvolvimento tornaram-se possíveis. Eles podem ser agrupados nas seguintes categorias: C para FPGA; MATLAB para FPGA; Simulink para FPGA (LALL, 2005); e LabVIEW para FPGA. Em todas elas, o objetivo final é

obter um fluxo automático onde, a partir de definições de projeto no nível desejado, chega-se à sintetização dessas definições para o hardware.

Antes de explorar melhor essas alternativas, é importante identificar uma série de papéis relacionados ao processo inteiro de desenvolvimento. Esses papéis podem ser desempenhados por uma ou mais pessoas dependendo do porte do projeto, mas o fundamental é a percepção de que tais conhecimentos, vindos de áreas distintas, devem estar presentes e alinhados.

Dentre os papéis identificados por Lall (2005) no desenvolvimento de projetos utilizando FPGAs, aqueles normalmente encontrados são: Arquiteto de Sistema, Desenvolvedor de Algoritmo, e Projetista de Hardware.

Arquitetos de Sistema estão envolvidos desde o início do projeto, analisando ou desenvolvendo os seus requisitos. Eles necessitam de ferramentas que permitam trabalhar em alto nível, bem como simplificar a integração entre as partes do projeto desenvolvidas por outros membros da equipe. Nesse contexto, o Simulink tornou-se uma alternativa bastante popular.

Desenvolvedores de Algoritmo são responsáveis pelas bases matemáticas do projeto, compreendendo as nuances do algoritmo e determinando de que forma obter fidelidade suficiente para atingir as especificações desejadas, sem criar ineficiências. Em projetos complexos, com várias taxas de amostragem, o desenvolvedor projetará cada parte, integrando-as junto com o Arquiteto de Sistema ao FPGA. Pela forma concisa na descrição dos algoritmos, linguagens como a usada no MATLAB permitem a melhor produtividade.

O papel do Projetista de Hardware inclui a qualificação de IPs comprados externamente, verificação dos IPs junto com o resto do circuito, avaliação do possível reúso de blocos já desenvolvidos e desenvolvimento de novos baseados em requisitos. Normalmente, tais requisitos vêm na forma de simulações em Simulink, que precisam

ser traduzidos para o formato RTL. O projetista, a partir de um conceito idealizado do algoritmo, converte-o em hardware de alto desempenho. Nesse papel, um ambiente de projeto e verificação para RTL é mandatório.

Assim, voltando à discussão sobre os diversos fluxos de desenvolvimento, especialmente quando esses fluxos tornam-se completamente automáticos, diversas especialidades, relacionadas com os papés identificados anteriormente, precisam estar incorporadas nas ferramentas que implementam tais fluxos.

2.5.3.1 Fluxo de Desenvolvimento: C para FPGA

Um fluxo de desenvolvimento como o apresentado em 2.5.1 está voltado essencialmente a usuários com experência e conhecimento em projetos de circuitos eletrônicos e HDL, estranho para a grande maioria de programadores de software (WAIN, 2006).

Com o objetivo de aumentar a gama de usuários, várias linguagens consideradas como HLL ("High Level Languages") têm aparecido para programar FPGAs, simplificando a transição na programação de um novo dispositivo. A imensa maioria dessas linguagens está baseada em C, principalmente por ser uma das poucas linguagens familiares tanto a engenheiros de software, como de hardware (GENEST, 2007). O escopo dessas ferramentas é oferecer um tradutor C→VHDL, de forma a integrar a saída com outras já usadas a partir daí (ver Seção 2.5.1).

A linguagem C, também conhecida como ANSI C, é definida por normas produzidas em 1989 e 1999. Para a implementação dos tradutores C→VHDL, como nenhuma das ferramentas disponíveis atualmente adere a qualquer padrão, parece mais correto identificá-los como dialetos da linguagem C.

Alguns dos tradutores disponíveis, também chamados de compiladores, são: DIME-C, SRC, Impulse-C, Mitrion-C, Catapult-C, Handel-C, Trident Compiler.

Alguns dialetos, como DIME-C são um sub-conjunto do padrão ANSI-C. Outros, como Impulse-C e Handel-C são super-conjuntos do padrão, incorporando características proprietárias. Algumas sintaxes, como Mitrion-C, divergem tanto do ANSI-C que fica difícil considerá-la na categoria C→VHDL (GENEST, 2007).

Em todas as alternativas, o algoritmo deve ser estruturado de forma a permitir estruturas de hardware otimizadas para o FPGA, tais como paralelismo e *pipelining*. É essa necessidade de estruturação que torna as ferramentas dialetos do padrão ANSI C e diferentes entre si. Em algumas delas, tal como Handel-C, o usuário deve explicitamente identificar os trechos de código a serem paralelizados ou "pipelined". Em outras, como Trident ou DIME-C, a filosofia é que a ferramenta deverá tentar paralelizar ou "pipeline" o código sempre que possível. DIME-C possui também "pragmas", ou diretivas de compilação, que permitem ao usuário especificar duplicação de lógica sem necessitar adicionar sintaxes não existentes no padrão ANSI (WAIN, 2006).

Algumas ainda têm a capacidade de integrar bibliotecas de funções na linguagem e chamá-las como uma função normal. É o caso de DIME-C e SRC. Essas bibliotecas são criadas, seja na própria linguagem (no caso de DIME-C), ou usando HDL, possibilitando a extensão da sua funcionalidade básica (GENEST, 2007).

O processo de compilação é executado em duas etapas. Primeiro é criada uma versão comportamental do algoritmo, especificando em detalhes como as operações lógicas devem ocorrer em relação aos ciclos do relógio. Essa descrição tem a forma de alguma linguagem HDL. A seguir, essa descrição comportamental passará por um segundo processo, onde as operações lógicas abstratas são atribuídas a recursos lógicos do FPGA, roteado e mapeado a pinos físicos (GENEST, 2007).

Do ponto de vista do projetista, normalmente o processo é completamente automático. Mas pode-se notar que o tradutor C→VHDL realiza a primeira parte da compilação,

sendo que a parte seguinte pode ser executada por alguma ferramenta voltada para o fluxo descrito em 2.5.1.

Algumas das alternativas citadas, em especial DIME-C e Handel-C, foram experimentadas em aplicações para computação de alto desempenho, como coprocesssadores em supercomputadores (WAIN, 2006; GENEST, 2007).

Através de ferramentas C→VHDL é possível a programadores sem conhecimento de VHDL gerarem aplicações para FPGA. Independentemente, se foram adicionados ou não comandos à versão ANSI-C, é necessário algum entendimento sobre a organização do hardware para desenvolver algoritmos que atinjam o desempenho esperado.

Nessa categoria de fluxo de desenvolvimento, a ênfase está no algoritmo a ser embarcado, em um nível de abstração mais alto com relação a uma implementação realizada diretamente em HDL.

Neste trabalho fixou-se a análise apenas em conversores a partir da linguagem C onde, apesar de terem por base uma sintaxe relativamente simples, observa-se a dificuldade na sua padronização. Dificuldades ainda maiores para efetuar comparações entre linguagens HLL aparecem quando a base dessas linguagens é C++, por exemplo, adotando conceitos de orientação a objetos. Pode-se citar nesse caso, o SystemC e o Precision-C, onde a primeira é definida pela norma IEEE Std 1666-2005, e a segunda é comercializada pela Mentor Graphics, um tradicional fabricante de ferramentas EDA.

2.5.3.2 Fluxo de Desenvolvimento: MATLAB para FPGA

Normalmente, ao se mencionar Matlab, pode-se estar usando o mesmo nome com dois significados distintos. Matlab pode ser entendido como um ambiente de modelagem e simulação, integrando Simulink e diversos pacotes cuja área de aplicação se extende a uma vasta gama de aplicações matemáticas e simulações dinâmicas. Também pode ser entendido como uma linguagem matemática, textual, cujas qualidades têm sido reconhecidas no meio acadêmico e para desenvolvimento de algoritmos na área de

processamento digital de sinais (BHATT, 2005). Nesta seção, o enfoque será para o segunda significado. Quando houver necessidade de usar o primeiro significado, este será chamado de Ambiente Matlab.

Um dos motivos da popularidade do ambiente Matlab, e da sua linguagem, no meio científico é a sintaxe concisa para executar funções complexas (VANEVENHOVEN, 2007). Por exemplo, para o cálculo de uma FFT de uma matriz de tamanho 4x1024 é suficiente apenas um comando, sem preocupações com escala ou alocação de dados:

```
y = fft( data , 1024 );
```

Para implementar a passagem de um algoritmo preocupado apenas em capturar o comportamento funcional, para uma versão implementável em hardware, existe uma ferramenta chamada AccelDSP (antigamente chamada de AccelFPGA), comercializada pela empresa AccelChip (VANEVENHOVEN, 2007). Ela foi evoluída a partir de um compilador chamado MATCH, desenvolvido pela Universidade de Northwestern, e patrocinado pelo programa de Sistemas Computacionais Adaptativos da DARPA (BANERJEE, 2003).

De forma parecida com o fluxo descrito para os tradutores C→VHDL, nesse caso para obtenção do fluxo completo usando o ambiente Matlab, é necessário haver integração com alguma ferramenta que implementa o fluxo descrito em 2.5.1.

A Figura 2.11 mostra uma parte do fluxo de desenvolvimento, desde a sua concepção ("Research Idea") até a geração de um algoritmo candidato a ser implementado em FPGA. Nela nota-se que o ambiente Matlab ("Matlab testbed") provê ao Arquiteto de Sistema e ao Desenvolvedor de Algoritmo um meio comum de desenvolvimento, através de etapas sucessivas de detalhamento. Uma etapa importante, relacionada com o Desenvolvedor de Algoritmo, é a conversão de um projeto utilizando aritmética em ponto flutuante para ponto fixo, auxiliada pelo pacote de Projeto e Análise de Filtros, integrado no próprio ambiente Matlab.

Alternativamente, AccelDSP pode gerar, durante a conversão do algoritmo para VHDL, também a conversão entre aritmética de ponto flutuante para ponto fixo (VANEVENHOVEN, 2007). Mas por questões de análise do comportamento funcional do algoritmo, parece mais recomendável que a conversão de ponto flutuante para ponto fixo seja feita dentro do ambiente Matlab.

Em todas as etapas mostradas na Figura 2.11, o ambiente Matlab está provendo a infraestrutura necessária para o desenvolvimento, sendo gerado como produto final um (ou mais) arquivo-fonte, textual, também chamado de código em formato M, com o algoritmo candidato à implementação.

A seguir, várias diretivas no formato AccelDSP precisam ser incorporadas manualmente ao arquivo. Essas diretivas servem como preparação para o processo de compilação indicando, por exemplo (BANERJEE, 2004):

- Trechos a serem copiados para serem paralelizados, como os comandos que estão dentro de laços de repetição;
- Trechos a serem "pipelined". Nesse caso, são acrescentados registradores na conexão de dados entre blocos que eram originalmente sequenciais. Para maiores detalhes sobre "pipeline", ver Fluxo de Desenvolvimento: LabVIEW para FPGA;
- Alocação de memória para recursos específicos da arquitetura destino;
- Endereçamento otimizado para regiões de memória compartilhada por vários blocos paralelizados.

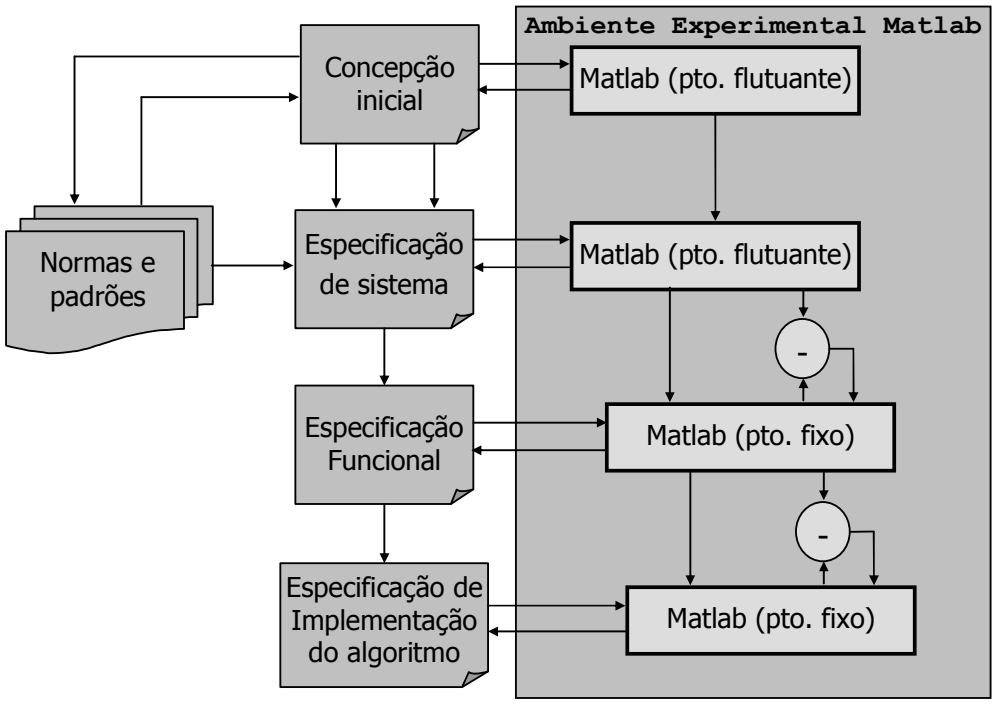


FIGURA 2.11 – Fluxo de desenvolvimento usando Matlab para avaliação de algoritmos.
FONTE: adaptado de Bhatt (2005).

AccelDSP é construído como um conjunto de passos de compilação, operando com representações intermediárias incrementais. A ferramenta usa como entrada um arquivo no formato Matlab como o gerado na etapa de geração de um algoritmo candidato, incorporado com as diretivas incluídas na preparação para o processo de compilação. Ela gera como saída um(vários) arquivo(s) no formato VHDL, a ser(em) usado(s) na etapa final, que é o fluxo definido na Seção 2.5.1.

Uma alternativa mencionada em Bhatt (2005) gera como saída do ambiente Matlab, na etapa de geração do algoritmo candidato, arquivos em formato C/C++, e os integra com uma ferramenta específica do fluxo de desenvolvimento C→VHDL, chamada Precision-C, produzida pela Mentor Graphics.

2.5.3.3 Fluxo de Desenvolvimento: Simulink para FPGA

O Simulink faz parte do Ambiente Matlab, sendo ele uma ferramenta para modelar sistemas dinâmicos através de uma interface gráfica bastante intuitiva, baseada em blocos funcionais. Como parte da sua funcionalidade, pode-se executar o sistema modelado em modo simulado, simplificando e otimizando o processo de validação.

Com extensões como o "Real-Time Workshop" (RTW), o Simulink permite a geração de código C a ser embarcado em diversos dispositivos e processadores, com ou sem sistemas operacionais.

Para aplicações específicas, baseadas exclusivamente em filtros digitais, o Simulink possui um pacote chamado "Filter Design and Analysis" – FDA – que permite projetá-los, simulá-los e analisá-los. Junto com outros dois pacotes – "Fixed-Point" e "Filter Design HDL Coder" – pode-se gerar arquivos em formato VHDL ou Verilog diretamente. Assim, a partir de requisitos funcionais de alto nível, um desenvolvedor pode obter uma topologia de filtro e sua implementação de uma forma bastante integrada e eficiente.

Um outro pacote, chamado "Xilinx System Generator", criado pela Xilinx e integrado ao ambiente Simulink, permite a implementação em FPGA diretamente, através do "Xilinx Blockset", um conjunto de blocos especializados para esse objetivo.

Contudo, não existe ainda uma maneira de gerar uma implementação de hardware de modelos que utilizem diretamente os blocos básicos do Simulink, restringindo a aplicabilidade dos recursos mencionados (SHANBLATT, 2005).

Uma maneira sugerida para se alcançar esse objetivo é a criação de um nível intermediário entre o Simulink e a VHDL, conhecido como Control Data Flow Graph (CDFG), mencionado explicitamente por Shanblatt (2005) e implicitamente por Krukowski (1999).

No método sugerido por Shanblatt (2005), a partir do arquivo Simulink, em formato MDL, inicialmente se analisa a estrutura para identificar quais blocos usados no modelo já possuem equivalentes implementados em hardware. A seguir, os blocos são expandidos em CDFG separados e, posteriormente, são combinados para aplicação de algoritmos de agendamento e alocação.

Esses algoritmos deverão resolver questões de restrição de tempo, de recursos e coordenação no agendamento de trechos do tipo "As Soon As Possible" – ASAP – e "As Late As Possible" – ALAP. Esse método, enquanto automático, ainda é uma sugestão conceitual, não havendo registros da obtenção de um produto final.

No entanto, o conceito em si parece bastante promissor e, adotando-se uma linguagem baseada em fluxo de dados como o LabVIEW, pode ser uma alternativa de implementação do método proposto.

Nesse caso, extrapolando o uso do Simulink como um ambiente para o desenvolvimento de um sistema até o nível de um algoritmo, pode-se dizer que qualquer outra ferramenta equivalente pode ser usada neste tipo de fluxo de desenvolvimento. Para citar alguns exemplos, existem MATRIXx/SystemBuild e Scilab/Scicos.

2.5.3.4 Fluxo de Desenvolvimento: LabVIEW para FPGA

LabVIEW – "Laboratory Virtual Instrument Engineering Workbench" – é uma linguagem de programação gráfica, criada e desenvolvida pela National Instruments há vinte anos, baseada em ícones ao invés de linhas de texto. Um exemplo dessa forma de programação pode ser vista na Figura 2.12. Sob esse aspecto, o LabVIEW pode ser considerado similar ao Simulink, apesar de originalmente essas ferramentas terem sido desenvolvidas com objetivos bastante distintos.

Em contraste com a maioria das linguagens de programação textuais, onde a seqüência de instruções determina a ordem de execução, o LabVIEW usa um conceito de

programação por fluxo de dados, onde o fluxo dos dados através dos nós no diagrama de blocos é que determina a ordem de execução da aplicação.

Outra característica importante dessa linguagem é uma inerente capacidade de paralelizar a execução de trechos do diagrama de blocos que são independentes entre si. Como ambiente de programação, o LabVIEW possui vários recursos de execução e depuração da aplicação, bem como uma grande facilidade de integração com dispositivos de aquisição e estímulo de dados.

O LabVIEW, inicialmente oferecido como uma ferramenta executando no Windows, mas atualmente tendo uma versão também para Linux, tem sofrido constantes atualizações tanto no aumento de funções disponíveis como na evolução do próprio ambiente.

Para sistemas que possam ser executados em computadores tradicionais, onde seja possível incluir cartões específicos de aquisição e estímulo de sinais, o desenvolvimento de uma aplicação pode ser executado em apenas um ambiente, desde a elaboração do seu algoritmo até a integração com outros elementos de hardware. Com isso, os papéis de Arquiteto de Sistemas, Desenvolvedor de Algoritmos, Projetista de Hardware e Desenvolvedor de Software compartilharão o mesmo ambiente. Exemplos onde essa organização é aplicável são: prototipações rápidas e ambientes de testes para simulações HIL.

Adicionalmente ao núcleo dessa linguagem, alguns módulos adicionais têm surgido, um deles sendo uma especialização do LabVIEW para desenvolvimento de aplicações voltadas a FPGAs chamado LabVIEW FPGA.

O módulo LabVIEW FPGA usa um sub-conjunto das funções disponíveis do LabVIEW padrão, já otimizado para ser embarcado em um FPGA, e permite apenas operações aritméticas em ponto fixo.

Esse módulo é especialmente integrado a alguns equipamentos, também fabricados pela National Instruments, que incluem um FPGA como dispositivo programável. Nesse caso existem cartões no formato PCI, para operação em computadores de mesa tradicionais, bem como um equipamento de classe industrial cuja arquitetura é baseada em FPGA.

Uma vez desenvolvida a aplicação dentro do ambiente LabVIEW FPGA, é feita uma chamada, também dentro do ambiente, para um compilador – esse sim externo – da Xilinx, gerando automaticamente o arquivo de configuração do FPGA. Apesar de usar VHDL como linguagem intermediária, o processo de compilação é totalmente transparente ao usuário.

A seguir, são apresentados alguns recursos e técnicas já mencionados anteriormente, em especial, paralelismo e "pipelining", cuja explicação é tornada mais simples graças à forma visual de apresentação através do LabVIEW.

A Figura 2.12 ilustra uma forma de paralelização possível, usando LabVIEW FPGA.

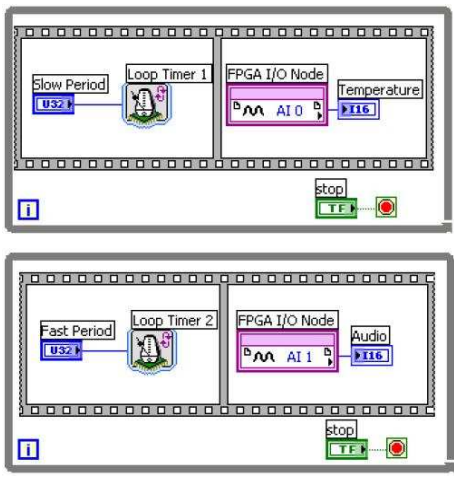


FIGURA 2.12 – LabVIEW: exemplo de execução paralela de código.

Nesse exemplo, duas variáveis – *Temperature* e *Audio* – estão sendo atualizadas independentemente, com dados vindos dos canais de aquisição AI0 e AI1, respectivamente. A taxa de aquisição de *Temperature* é definida pela variável *Slow*

Period, e de forma equivalente, a taxa de aquisição de *Audio* é definida pela variável *Fast Period*.

Supõe-se que os canais analógicos de entrada AI0 e AI1 fazem parte do sistema, e que seu acesso é feito de maneira automática pelo LabVIEW FPGA. Isto necessita de poucas operações para o seu uso correto, reduzindo muito o esforço de configuração desse tipo de sinal.

Pipeline é uma técnica usada para aumentar o desempenho de um trecho de código onde estão presentes diversas operações seqüenciais. Para a sua implementação, divide-se primeiro esse trecho de código em diversos sub-trechos. Em linguagens textuais, esses sub-trechos normalmente são chamados de subrotinas. Em LabVIEW eles são chamados de sub VIs, mas o conceito é o mesmo.

A Figura 2.13 ilustra essa etapa inicial, onde o trecho seqüencial é representado pelas sub VIs A, B e C.

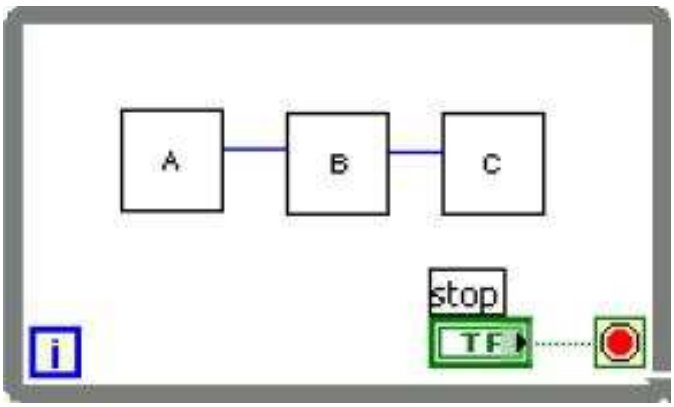


FIGURA 2.13 – "Pipeline": trecho de código original.
FONTE: National Instruments (2008).

Essas sub VIs são interconectadas através de variáveis intermediárias que, no caso do LabVIEW, são chamadas de "shift registers", ou registradores de deslocamento. Em termos de um sistema digital, isto é equivalente ao operador z^{-1}. A Figura 2.14 mostra o resultado da aplicação da técnica de pipeline ao trecho inicial apresentado na Figura 2.13.

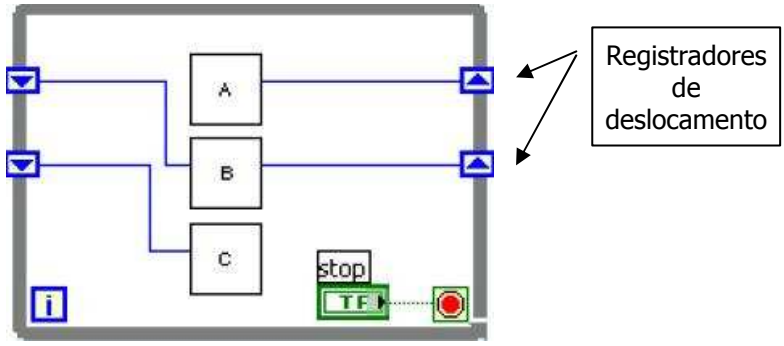


FIGURA 2.14 – "Pipeline": trecho de código otimizado.
FONTE: National Instruments (2008).

Quando se implementa um "pipeline", a saída do passo final estará atrasada da entrada pelo número de passos do "pipeline", e essa saída estará inválida enquanto as variáveis intermediárias não estiverem preenchidas. O número de passos em um "pipeline" é chamado de profundidade do "pipeline", e a latência, medida em ciclos de relógio, corresponde à sua profundidade. Assim, para um "pipeline" de profundidade N, o resultado será inválido até o N-ésimo ciclo de relógio. Uma visualização do processo pode ser vista na Figura 2.15, abaixo.

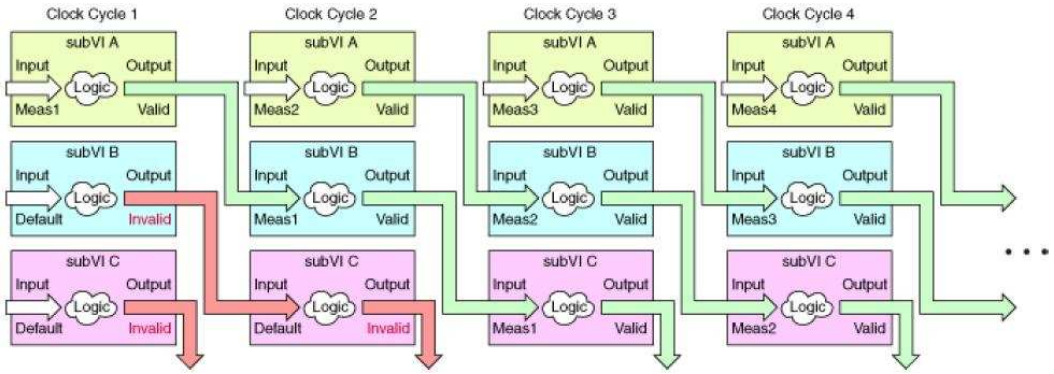


FIGURA 2.15 – "Pipeline": fluxo de dados.
FONTE: National Instruments (2008).

Uma das conseqüências de se usar uma estrutura "pipeline", é o aumento na latência do sinal de saída, frente a uma estrutura seqüêncial. Isso fica claramente mostrado na parte superior da Figura 2.16. No entanto, o "pipeline" cria períodos ociosos no ciclo de execução, podendo ser usados para aumentar a freqüência desse ciclo, de forma que a

latência medida em unidades de tempo não deve sofrer alterações significativas, como mostrado na parte inferior da Figura 2.16.

Para um FPGA uma estrutura desse tipo não interfere significativamente na sua utilização de recursos, diferentemente do que ocorreria em um processador tradicional.

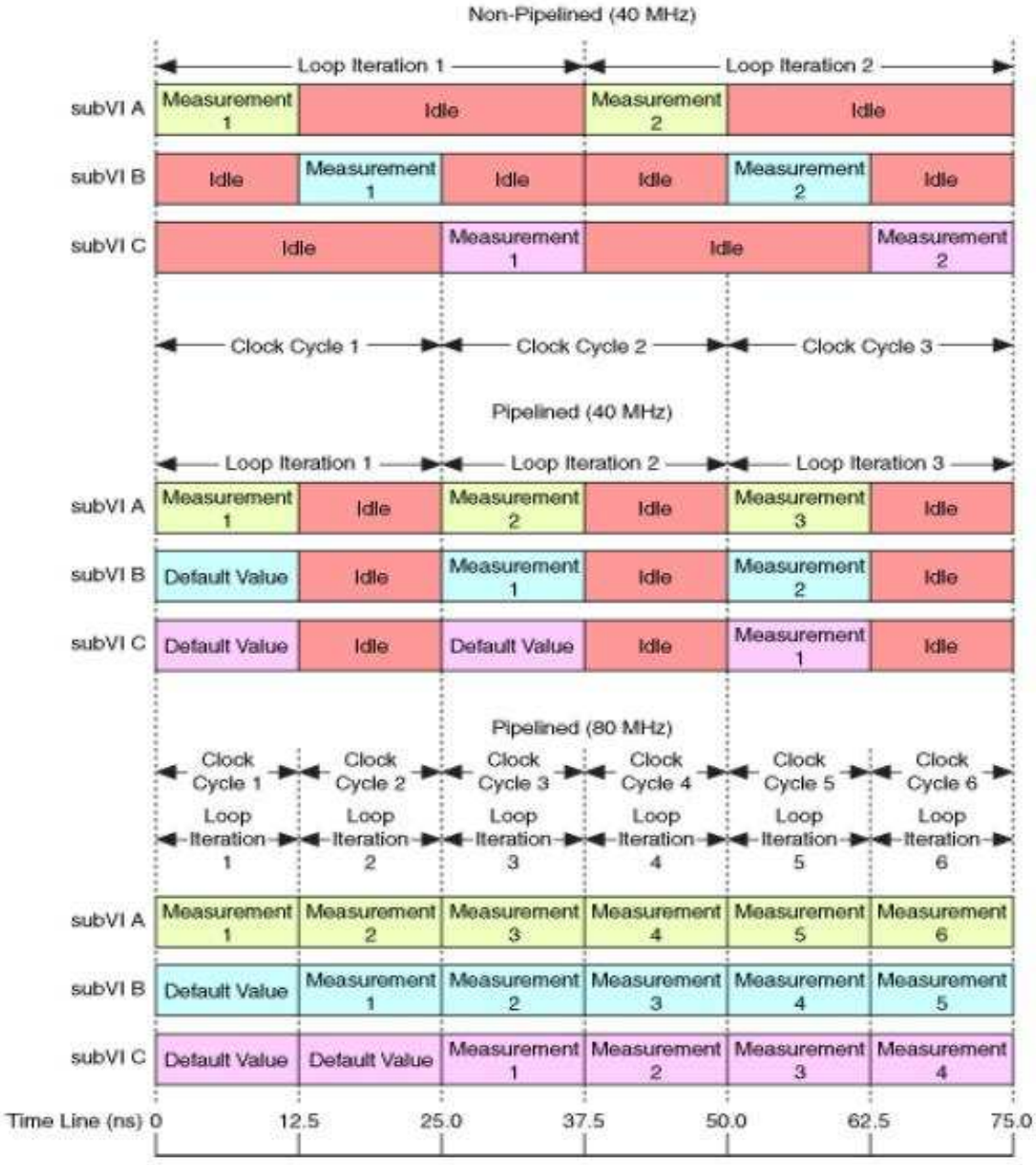


FIGURA 2.16 – "Pipeline": comparação do desempenho temporal.
FONTE: National Instruments (2008).

2.6 Sistemas Operacionais

Um sistema operacional pode ser entendido como um componente da infraestrutura de software de um sistema computacional, cuja responsabilidade é gerenciar e coordenar atividades e compartilhamentos dos recursos de um computador. Um sistema operacional funciona como um hospedeiro para aplicações que são executadas, proporcionando uma maneira estável e consistente para lidar com o hardware, sem precisar conhecê-lo em todos os detalhes.

Sistemas operacionais oferecem diversos serviços para programas aplicativos e usuários. No caso das aplicações, esses serviços são invocados por meio de interfaces, que podem ser uma passagem de parâmetros, a execução de um serviço, ou a recepção do resultado de alguma operação. Usuários podem interagir com o sistema operacional de forma parecida, usando interfaces gráficas ou por linhas de comando.

Dentre as diversas formas de se categorizar sistemas operacionais, a mais abrangente tem sido com respeito ao número de usuários e aplicações que podem ser atendidas. Assim, 3 categorias se destacam:

- monousuário, monotarefa: O sistema operacional é criado de forma que um único usuário possa executar apenas uma tarefa, ou aplicação, por vez. Normalmente, esse tipo de sistema operacional está associado a um sistema computacional com limitados recursos de hardware. O MS-DOS e Palm OS são exemplos nessa categoria;

- monousuário, multitarefa: Este tipo de sistema operacional é o mais utilizado nos computadores de mesa e laptops atuais, cujos recursos de hardware são adequados a esse tipo de configuração, onde um usuário pode executar diversos programas ao mesmo tempo. Para tanto, recursos precisam ser alocados e protegidos reduzindo ao mínimo a possibilidade de problemas de

um aplicativo interferir com os outros. O Microsoft Windows e o Apple MacOS são exemplos nessa categoria;

- multiusuário: Neste caso, diversos usuários podem utilizar simultaneamente os recursos do computador. Está implícito que o sistema possui também características multitarefa. É responsabilidade do sistema operacional permitir que cada aplicação disponha de recursos suficientes e independentes, assegurando que problemas em um dado aplicativo de usuário não afete os outros (tanto aplicativos como usuários). O AT&T Unix e o DEC VMS são sistemas operacionais que se encaixam nesta categoria.

2.6.1 Sistemas Operacionais de Tempo Real

Sistemas de tempo real são aqueles onde o aspecto temporal de seu comportamento faz parte da sua especificação. O correto funcionamento do sistema depende não somente da correção lógica e computacional dos resultados gerados, mas também da correção temporal dos resultados gerados. Caracterizam-se por comportar-se com exatidão tanto lógica e funcionalmente, como temporalmente (YABARRENA, 2006). Exemplos de sistemas nessa categoria podem ser sistemas embarcados, robôs industriais, controle industrial, sistemas aeroespaciais, dentre outros.

Ao se projetar um sistema de tempo real baseado em algum tipo de aplicativo implementado por software, normalmente ele é executado sobre um Sistema Operacional de Tempo Real (SOTR). Um SOTR não possui necessariamente uma grande capacidade de processamento, o seu maior objetivo é garantir que os requisitos de tempo das aplicações, especialmente os prazos (“deadlines”), sob seu controle sejam atingidos.

Ele facilita a criação de um sistema de tempo real, mas um SOTR sozinho não garante que o resultado final tenha características de tempo real. Isso vai depender também do desenvolvimento correto do programa aplicativo.

Um SOTR é basicamente um sistema operacional com requisitos adicionais relacionados a tempo, cuja implementação gera a propriedade mais relevante para esse tipo de sistema: a previsibilidade. Ou seja, o seu comportamento temporal deve ser tão previsível em tempo de projeto quanto for necessário para atender às especificações do sistema em tempo de execução.

Os SOTR podem ser divididos em dois grandes grupos: "soft real-time" e "hard real-time". No primeiro, os requisitos de tempo nem sempre são atingidos, mas isso não provoca um problema irreversível ao sistema. Já no segundo, os requisitos de tempo sempre têm que ser atingidos, sob pena de se causar um problema irreversível ao sistema.

De certa maneira, pode-se considerar que qualquer sistema operacional é avaliável quanto às suas características de tempo real, especialmente para a categoria "soft real-time". Mesmo que ele não atenda a algum requisito para considerá-lo como sendo "hard real-time", o desempenho geral pode ser aceitável para uma aplicação em particular.

Diferente dos sistemas operacionais genéricos, onde apenas uns poucos fabricantes dominam o mercado, no mundo dos SOTR isso não acontece. Referências indicam que o número de SOTR ultrapassa 50, sendo que nenhum deles possui hegemonia de mercado, e muitos deles atendem a nichos específicos.

Dentre esses nichos específicos, no meio aeronáutico pode-se citar o VxWorks da WindRiver, como um sistema operacional de tempo real tradicional e popular. Uma de suas características mais importantes é o crédito que autoridades certificadoras dão a respeito da sua confiabilidade, no que se refere a falhas próprias.

2.6.2 Windows

O Windows é o exemplo mais popular de um sistema operacional de uso geral, ou seja, otimizado para executar várias aplicações simultaneamente, garantindo que todas elas

recebam algum intervalo de tempo de processamento, bem como o atendimento a interrupções vindas de periféricos do tipo mouse e teclado. Uma outra característica de sistemas operacionais de propósito geral é a incapacidade de aplicações ou "drivers" de nível do "kernel" inibirem completamente interrupções para ganhar controle sobre o sistema operacional. Com isso, apesar de latências do sistema operacional terem valores baixos em média, estas podem eventualmente atingir valores altos, da ordem de centenas de milissegundos (INTERVALZERO, 2008).

Dentre as diversas versões, especialmente o Windows2000 e o WindowsXP podem ser considerados como exemplos para computação rápida, ou seja, uma grande quantidade de dados é processada o mais rapidamente possível, se forem executados em um sistema computacional poderoso.

Apesar de computação rápida ("fast computing") não ser igual à computação em tempo real, dependendo dos requisitos de uma aplicação, essa distinção pode tornar-se pouco importante.

2.6.3 LabVIEW RT

O LabVIEW RT é considerado pelo seu fabricante como um sistema operacional de tempo real, tendo sido desenvolvido a partir de outro, chamado Pharlap.

O Pharlap, por sua vez, é um descendente do DOS mas com recursos de tempo real, monousuário, monotarefa, mas multi-processo ("multithread"). A maior característica do Pharlap é a sua compatibilidade com a API do Windows, ou seja, em princípio uma DLL criada para ser executada no Windows também pode ser executada no Pharlap – e, portanto, no LabVIEW RT.

Por ser monotarefa, mas multi-processo, aplicações de tempo real construídas sobre esse sistema devem ser projetadas na forma de um grande executável que, por sua vez, disparará a execução de vários processos.

Um processo pode ser entendido como uma alternativa mais leve que uma tarefa, onde a vantagem é a simplificação (e conseqüente rapidez) na sua criação e destruição pelo sistema operacional, mas como desvantagem a carência de mecanismos de isolamento, ou compartimentação. Assim, um processo mau comportado pode afetar todos os outros processos, corrompendo-os.

Uma das extensões que o LabVIEW RT possui frente ao Pharlap é a capacidade de execução em processadores de núcleos múltiplos – "multicore". Dessa forma, novos processadores podem ter suas características melhor exploradas.

Outra características importante é a compatibilidade com vários cartões de aquisição, estímulo ou reconfiguráveis, tornando esse sistema operacional uma alternativa interessante dentro de um contexto onde é necessário incluir algum hardware especial para uma aplicação específica, como por exemplo, um controlador digital.

CAPÍTULO 3

METODOLOGIA E PROJETO DOS SISTEMAS CONTROLADORES

Este trabalho procura exercitar um dos métodos apresentados no Capítulo 2 para desenvolvimento de um sistema de controle em aplicações aeroespaciais, a ser implementado em um ambiente de simulação distribuída.

Nesse processo, várias etapas são realizadas, cada uma gerando resultados intermediários a serem refinados ou confirmados na etapa posterior. Começar-se-á com uma estimativa de ganho do controlador com um sistema linearizado, prosseguindo com uma otimização do controlador em um ambiente simulado em tempo virtual, finalizando com a migração do controlador e parte da simulação para um FPGA, integrando com o resto da simulação sendo executada em tempo real.

Assim, neste capítulo, inicialmente é apresentado o fluxo de desenvolvimento a ser utilizado. A seguir, o problema é formulado, o sistema é modelado e linearizado, e finalmente o controlador é definido e seus parâmetros são estimados com uma versão linearizada do modelo.

3.1 Fluxo de Desenvolvimento Adotado

O fluxo de desenvolvimento adotado neste trabalho para as partes do sistema e controlador simulados a serem embarcados em FPGA será uma combinação do fluxo de desenvolvimento Simulink→FPGA e LabVIEW→FPGA.

O Simulink é um ambiente que pode ser usado tanto para definir a arquitetura do sistema como para permitir o estudo dos algoritmos a serem embarcados no FPGA. Nesse sentido, ao se detalhar os algoritmos, os blocos utilizados possuem funcionalidades similares aos existentes no LabVIEW FPGA. Nesse momento, a transição entre os ambientes Simulink e LabVIEW pode ser feita de maneira suave, já

que ambas as descrições, desconsideradas as particularidades de apresentação gráfica, serão parecidas.

Outra vantagem no uso do Simulink na criação do ambiente de simulação distribuída é a sua geração automática de código para as partes que forem ser executadas por um processador tradicional, usando algum mecanismo de tempo real.

Alguns cuidados devem ser tomados ao usar um ambiente do tipo Simulink em um fluxo como o proposto. Um está relacionado com o nível de abstração: o Simulink possui blocos com funcões bastante compactas para serem usadas na definição de um sistema. No entanto, esse tipo de representação não tem um equivalente adequado em LabVIEW. Nas partes do sistema a serem embarcadas em um FPGA, é recomendável detalhar o algoritmo para facilitar a transição entre os ambientes.

Outro cuidado está relacionado à representação numérica dos parâmetros e variáveis do sistema modelado. No Simulink o padrão é ponto flutuante no formato "double precision", enquanto que nativamente, o LabVIEW FPGA usa apenas aritmética de ponto fixo. Como recurso para mitigar essa diferença, foi criada uma biblioteca em LabVIEW FPGA para permitir operar com aritmética em ponto flutuante, no formato "single precision".

Dessa forma, o fluxo proposto pode ser representado conforme a Figura 3.1

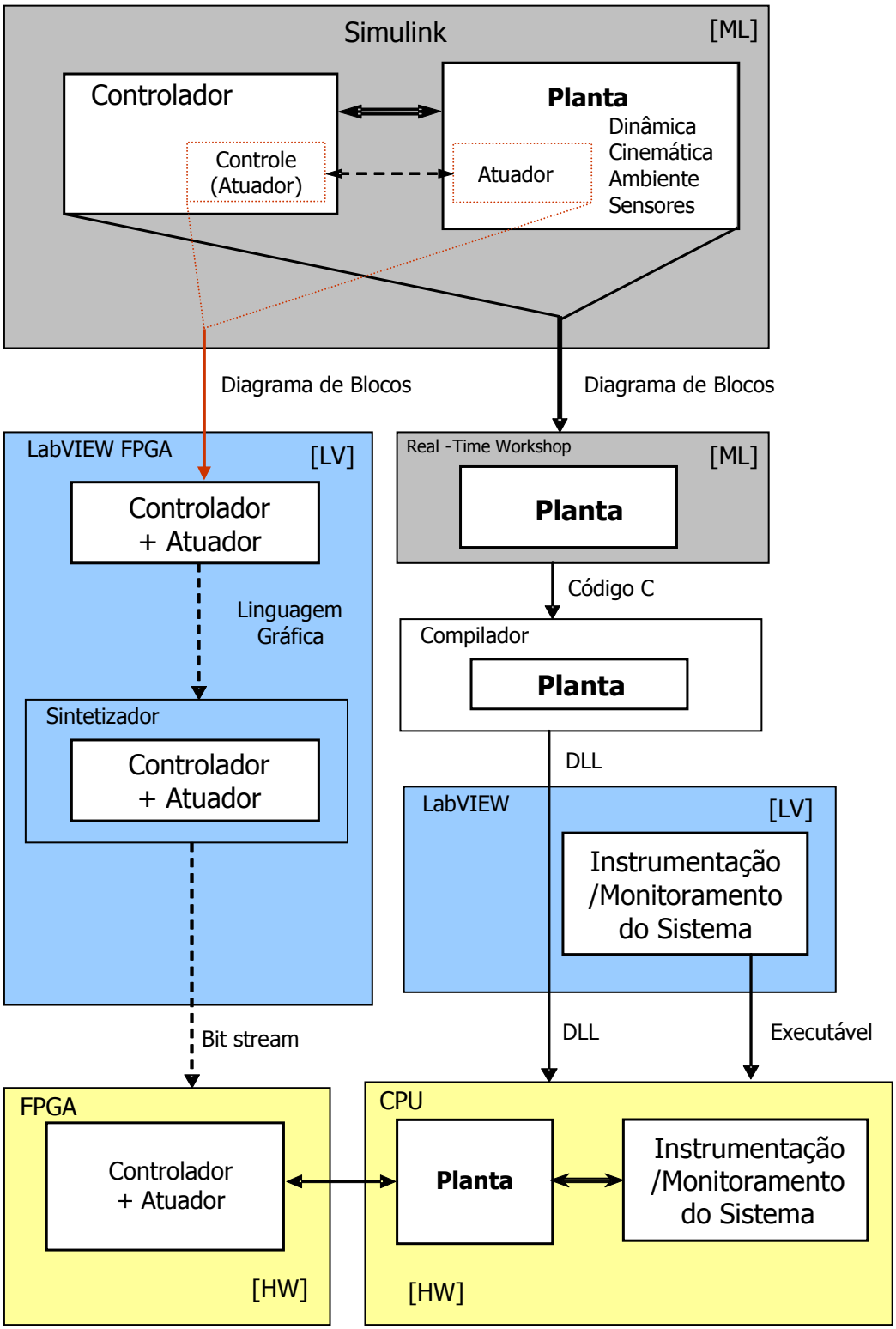


FIGURA 3.1 – Fluxo de desenvolvimento proposto.

3.2 Formulação do Problema

Como parte da formulação do problema, deve-se definir a missão a ser adotada para a utilização da PMM. A missão escolhida é denominada Satélite de Sensoriamento Remoto (SSR), e segundo INPE (2001c), ela será responsável pela monitoração da Floresta Amazônica Brasileira, em uma faixa de latitude entre 5° Norte a 15° Sul, compreendendo uma área aproximada de 5 milhões de km^2.

Assim sendo, as características da órbita a serem adotadas neste trabalho para a missão SSR são:

- Altitude: 905 km;
- Inclinação: 0°;
- Excentricidade: 0.

Dentre os diversos Modos de Operação do satélite (GOBATO, 2007), o Modo Nominal será o adotado, cujos requisitos relacionados à atitude da PMM são:

- Precisão de apontamento: <0.05° (3σ);
- Desvio ("off pointing") de até 30° em 180s.

3.3 Modelos Matemáticos

Normalmente nos processos de modelagem de sistemas reais são adotadas algumas condições simplificadoras. No desenvolvimento a seguir não se foge à regra.

3.3.1 Caso Espacial: PMM

Para o caso espacial, a PMM foi modelada considerando-se as seguintes características:

- satélite como sendo um corpo rígido, sem flexão;

- torques internos nulos;
- atrito da roda e momento inicial nulos;
- torques externos de perturbação constantes, no valor de $1x10^{-4}$ N.m em cada eixo do satélite.
- simulação dinâmica do satélite considerando rodas de reação modificadas, com características definidas conforme Souza (1980).
- Planta – Dinâmica do Movimento

Assumindo o satélite como um corpo rígido composto de um grande conjunto de partículas de massa *dm*, conforme Meirovitch (1970), o momento angular em relação à origem dos eixos de coordenadas do corpo, coincidente com o seu centro de massa é:

$$\vec{H}_S = \int \vec{r} \times \left(\vec{\omega}_S \times \vec{r}\right) \cdot dm \tag{3.1}$$

onde $\vec{r}$ é o vetor posição do elemento de massa dm em relação à origem, e $\vec{\omega}_S$ o vetor velocidade angular do satélite.

Aplicando propriedades de Álgebra Vetorial à relação acima, obtém-se a expressão do momento angular do satélite em termos de suas componentes ortogonais:

$$\vec{H}_S = h_{Sx}\hat{x} + h_{Sy}\hat{y} + h_{Sz}\hat{z} \tag{3.2}$$

onde,

$$\begin{aligned} h_{Sx} &= I_{Sx} \cdot \omega_{Sx} - I_{Sxy} \cdot \omega_{Sy} - I_{Sxz} \cdot \omega_{Sz} \\ h_{Sy} &= -I_{Sxy} \cdot \omega_{Sx} + I_{Sy} \cdot \omega_{Sy} - I_{Syz} \cdot \omega_{Sz} \\ h_{Sz} &= -I_{Sxz} \cdot \omega_{Sx} - I_{Syz} \cdot \omega_{Sy} + I_{Sz} \cdot \omega_{Sz} \end{aligned} \tag{3.3}$$

que, em uma forma mais compacta, fica:

$$\vec{H}_S = I_S \cdot \vec{\omega}_S \tag{3.4}$$

onde I_S é definida como sendo a matriz de inércia do corpo.

De acordo com a 2ª lei de Newton para a dinâmica de rotação, pode-se dizer que a taxa de variação temporal do momento angular, definido e escrito em coordenadas do referencial inercial, é igual à resultante dos torques externos em relação à mesma origem, relação essa que define a equação básica da dinâmica de atitude do satélite, equipado com rodas de reação. Assim,

$$\frac{d\vec{H}}{dt} = \vec{T}_{Ext} \tag{3.5}$$

onde:

$$\vec{H} = \vec{H}_S + \vec{H}_R \tag{3.6}$$

Sendo $\vec{H}_R$ o momento angular das rodas de reação.

Reescrevendo a equação acima em termos do referencial móvel que está girando com velocidade angular $\vec{\omega}_S$, tem-se:

$$\frac{d\vec{H}}{dt} + \vec{\omega}_S \times \vec{H} = \vec{T}_{Ext} \tag{3.7}$$

Portanto,

$$\dot{\vec{H}}_S + \dot{\vec{H}}_R + \vec{\omega}_S \times \left(\vec{H}_S + \vec{H}_R\right) = \vec{T}_{Ext} \tag{3.8}$$

onde

$$\vec{H}_R = I_R \cdot \vec{\omega}_R = I_R \cdot \vec{\omega}_S + I_R \cdot \vec{\omega}_{RS} \tag{3.9}$$

$\vec{\omega}_{RS}$ é o vetor velocidade relativa das rodas de reação em relação ao satélite:

$$\vec{\omega}_{RS} = \vec{\omega}_R - \vec{\omega}_S \tag{3.10}$$

Substituindo as equações 3.9 e 3.10 em 3.8, chega-se a:

$$\dot{\vec{H}}_S + \dot{\vec{H}}_R + \vec{\omega}_S \times \left(I_S \cdot \vec{\omega}_S + I_R \cdot \vec{\omega}_S + I_R \cdot \vec{\omega}_{RS}\right) = \vec{T}_{Ext} \tag{3.11}$$

$$\dot{\vec{H}}_S + \dot{\vec{H}}_R + \vec{\omega}_S \times \left[\left(I_S + I_R\right) \cdot \vec{\omega}_S + I_R \cdot \vec{\omega}_{RS}\right] = \vec{T}_{Ext} \tag{3.12}$$

Observando que o segundo termo com o sinal trocado opera como torque de controle – $\dot{\vec{H}}'_R$, com $\dot{\vec{H}}_R = -\dot{\vec{H}}'_R$ – torque este gerado pelas rodas de reação sobre o satélite. Reordenando os termos tem-se:

$$\dot{\vec{H}}_S + \vec{\omega}_S \times \left[\left(I_S + I_R\right) \cdot \vec{\omega}_S + I_R \cdot \vec{\omega}_{RS}\right] = \vec{T}_{Ext} + \dot{\vec{H}}'_R \tag{3.13}$$

A equação 3.13 é uma forma de representação das equações do movimento do satélite definido em relação ao referencial inercial, porém descrito nas coordenadas do satélite, e inclui as seguintes fontes de torques giroscópicos:

- devidos ao produto de $\vec{H}_S = I_S \cdot \vec{\omega}_S$ para os casos onde I_S não é diagonal;
- devidos ao satélite: $\vec{\omega}_S \times \left[\left(I_S + I_R\right) \cdot \vec{\omega}_S\right]$;
- devidos às rodas de reação: $\vec{\omega}_S \times \left(I_R \cdot \vec{\omega}_{RS}\right)$.

O vetor torque $\vec{T}_{Ext}$ é o resultante dos torques perturbadores provenientes de efeitos ambientais, tais como gradiente de gravidade, correntes parasitas, arrasto aerodinâmico e pressão da radiação solar. O vetor $\dot{\vec{H}}'_R$ é a taxa de variação temporal do momento angular das rodas de reação, também chamado de torque de controle, uma vez que representa a parcela controlável da equação.

3.3.1.1 Planta – Cinemática do Movimento

A atitude de um satélite pode ser representada por quaternions ou por ângulos de Euler. Para cálculos computacionais, a representação por quaternions evita singularidades nos resultados e simplifica o cálculo por não necessitar de relações trigonométricas, mas carece de uma clara interpretação física. Por outro lado, a representação por ângulos de Euler possui singularidades nos resultados, usa relações trigonométricas, mas também propicia uma interpretação física mais direta.

Para os objetivos deste trabalho, foi adotada a representação por ângulos de Euler, uma vez que nos casos a serem explorados pode-se evitar a condição de singularidade e isto simplifica o processo de geração dos resultados gráficos. Como a execução da cinemática será feita em um processador tradicional com coprocessador aritmético, estima-se que esses cálculos tenham impacto desprezível no desempenho geral da simulação.

A orientação de um sistema cartesiano com respeito a um outro sistema cartesiano pode ser descrita por três rotações sucessivas, sendo os ângulos dessas rotações chamados de ângulos de Euler (STEVENS, 2003). Essa técnica é usada em vários campos diferentes das ciências e, no caso aeroespacial, essas rotações obedecem a uma seqüência sobre os três eixos cartesianos.

No caso de um satélite artificial, ao se descrever a orientação de um referencial fixo ao corpo (B) em relação ao referencial VLHL (R) em termos dos ângulos (ϕ, θ, ψ) de Euler, pode-se escolher uma seqüência de rotações como $C_3(\psi) \rightarrow C_2(\theta) \rightarrow C_1(\phi)$ para levar um vetor descrito no sistema R para o sistema B. Essa seqüência também é conhecida como 3-2-1, onde cada rotação é definida conforme as equações 3.14 – 3.16 abaixo (STEVENS, 2003):

$$C_3(\psi)=\begin{bmatrix}\cos\psi & sen\,\psi & 0\\ -sen\,\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix} \tag{3.14}$$

$$C_2(\theta)=\begin{bmatrix}\cos\theta & 0 & -sen\,\theta\\ 0 & 1 & 0\\ sen\,\theta & 0 & \cos\theta\end{bmatrix} \tag{3.15}$$

$$C_1(\phi)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & sen\,\phi\\ 0 & -sen\,\phi & \cos\phi\end{bmatrix} \tag{3.16}$$

Ou, em termos visuais, conforme a Figura 3.2

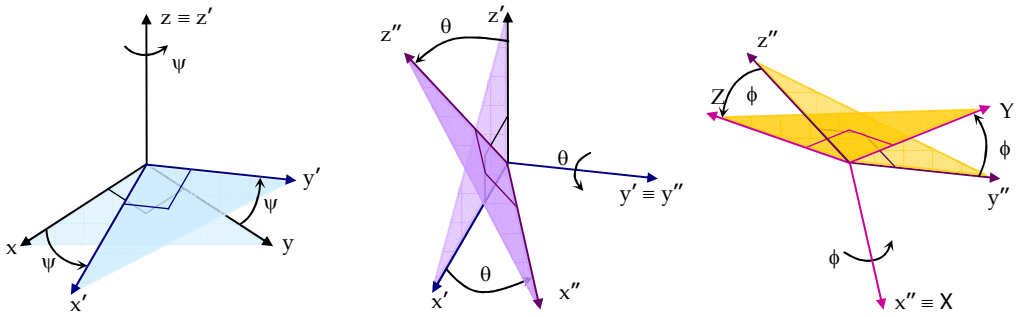


FIGURA 3.2 – Seqüência de rotações 3-2-1 dos ângulos (ψ, θ, ϕ), respectivamente.

que correspondem a partir-se do referencial x, y, z chegar-se ao referencial X, Y, Z através das rotações:

1. Em torno de z, de um ângulo ψ, levando o referencial para x′, y′, z′;
2. Em torno de y′, de um ângulo θ, levando o referencial para x′′, y′′, z′′;
3. Em torno de x′′, de um ângulo ϕ, levando o referencial para X, Y, Z.

Assim,

$$C_{B/R} \equiv C_1(\phi)C_2(\theta)C_3(\psi) = \begin{bmatrix} c\theta \cdot c\psi & c\theta \cdot s\psi & -s\theta \\ (-c\theta \cdot s\psi + s\phi \cdot s\theta \cdot c\psi) & (c\phi \cdot c\psi + s\phi \cdot s\theta \cdot s\psi) & s\phi \cdot c\theta \\ (s\phi \cdot s\psi + c\phi \cdot s\theta \cdot c\psi) & (-s\phi \cdot c\psi + c\phi \cdot s\theta \cdot s\psi) & c\phi \cdot c\theta \end{bmatrix} \quad (3.17)$$

onde $c\phi \equiv \cos\phi$, $s\phi \equiv sen\phi$; e, respectivamente para θ e ψ. Das equações chega-se a:

$$\begin{bmatrix} \vec{b}_1 \\ \vec{b}_2 \\ \vec{b}_3 \end{bmatrix} = \begin{bmatrix} c\theta \cdot c\psi & c\theta \cdot s\psi & -s\theta \\ (-c\theta \cdot s\psi + s\phi \cdot s\theta \cdot c\psi) & (c\phi \cdot c\psi + s\phi \cdot s\theta \cdot s\psi) & s\phi \cdot c\theta \\ (s\phi \cdot s\psi + c\phi \cdot s\theta \cdot c\psi) & (-s\phi \cdot c\psi + c\phi \cdot s\theta \cdot s\psi) & c\phi \cdot c\theta \end{bmatrix} \cdot \begin{bmatrix} \vec{r}_1 \\ \vec{r}_2 \\ \vec{r}_3 \end{bmatrix} \quad (3.18)$$

Os ângulos (ϕ, θ, ψ) de Euler adotados, seguem uma convenção bastante utilizada no meio aeroespacial. Outra convenção bastante utilizada refere-se a "roll", "pitch" e "yaw" ou também rolagem, arfagem e guinada, onde:

- rotação em torno do eixo z (ψ): *yaw*
- rotação em torno do eixo y (θ): *pitch*
- rotação em torno do eixo x (ϕ): *roll*

Para a seqüência de rotações descrita acima, a velocidade angular do referencial B (fixo no corpo) com relação a R (VLHL) fica (ARANTES, 2005; MOREIRA, 2006):

$$\begin{bmatrix} \vec{\omega}_x \\ \vec{\omega}_y \\ \vec{\omega}_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -sen\theta \\ 0 & \cos\phi & sen\phi \cdot \cos\theta \\ 0 & -sen\phi & \cos\phi \cdot \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (3.19)$$

$$\begin{bmatrix} \vec{\omega}_x \\ \vec{\omega}_y \\ \vec{\omega}_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -sen\theta \\ 0 & \cos\phi & sen\phi \cdot \cos\theta \\ 0 & -sen\phi & \cos\phi \cdot \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} - \omega_0 \cdot \begin{bmatrix} \cos\theta \cdot sen\psi \\ sen\phi \cdot sen\theta \cdot sen\psi + \cos\phi \cdot \cos\psi \\ \cos\phi \cdot sen\theta \cdot sen\psi - sen\phi \cdot \cos\psi \end{bmatrix} \quad (3.20)$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \frac{1}{\cos\theta} \begin{bmatrix} \cos\theta & sen\phi sen\theta & \cos\phi\cos\theta \\ 0 & \cos\phi\cos\theta & -sen\phi\cos\theta \\ 0 & sen\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + \frac{\omega_0}{\cos\theta} \begin{bmatrix} sen\psi \\ \cos\theta\cos\psi \\ sen\theta sen\psi \end{bmatrix} \quad (3.21)$$

onde ω_0 é a velocidade orbital do satélite e consequentemente a velocidade angular com que o referencial VLHL gira em relação ao referencial inercial.

3.3.1.2 Linearização da Planta

A técnica de linearizar o modelo de um sistema é frequentemente adotada para facilitar a obtenção de controladores por meio analítico, uma vez que normalmente o modelo não linear de um sistema só pode ser controlado por métodos numéricos.

Uma das formas de linearizar o modelo de um sistema não linear é adotar algumas simplificações em torno de um determinado ponto de operação. Para o caso da PMM, são aplicáveis as simplificações adotadas por Souza (1980) para a dinâmica do satélite:

1. Se $|I_S| >> |I_R|$, então $(I_R + I_S) \cong I_S$;

2. Se $\vec{\omega}_S$ for pequeno, então os torques giroscópicos devidos ao movimento de rotação do conjunto satélite+rodas de reação em relação ao referencial inercial serão desprezíveis;

3. Se $\vec{H}_{RS}$ for pequeno, então os torques giroscópicos devido ao movimento de rotação das rodas de reação, em relação ao referencial móvel serão desprezíveis;

4. Se adotarmos como referencial móvel os três eixos principais de inércia do satélite, então I_S é diagonal, desaparecendo assim os torques gerados pelos produtos de inércia;

Uma vez que:

- De 1): $I_{Sx} = 295{,}71[kg.m^2]$, $I_{Sy} = 501{,}37[kg.m^2]$, $I_{Sz} = 364{,}82[kg.m^2]$ e $I_{Rj} = 0{,}015[kg.m^2]$;

- De 2) e 3): No modo nominal $\omega_{Sj} \leq 0{,}167°/s = \frac{\pi}{1080} rd/s$, $H_{RSj} \leq 12N \cdot m \cdot s$ e considerando I_{Sx}, I_{Sy}, I_{Sz} e I_R, tem-se no pior caso $\dot{h}_{Sy} \cong 0{,}00424N \cdot m$ e $\dot{h}_{RS\_xyz} \cong 0{,}0349N \cdot m$, sendo bem menores que $\dot{h}_{RS\_xyz} \cong 0{,}6N \cdot m$;

- De 4): I_S é diagonal.

Com isso, a equação 3.13, repetida abaixo,

$$\dot{\vec{H}}_S + \vec{\omega}_S \times \left[\left(I_S + I_R \right) \cdot \vec{\omega}_S + I_R \cdot \vec{\omega}_{RS} \right] = \vec{T}_{Ext} + \dot{\vec{H}}_R \tag{3.22}$$

pode ser reduzida a:

$$\begin{aligned} I_{Sx} \cdot \dot{\omega}_{Sx} &= \dot{T}_{Ext\_x} + \dot{h}_{Rx} \\ I_{Sy} \cdot \dot{\omega}_{Sy} &= \dot{T}_{Ext\_y} + \dot{h}_{Ry} \\ I_{Sz} \cdot \dot{\omega}_{Sz} &= \dot{T}_{Ext\_z} + \dot{h}_{Rz} \end{aligned} \tag{3.23}$$

Para o caso da cinemática do satélite, conforme Gobato (2007), supõe-se que as correções dos desvios angulares são feitas em torno da condição de apontamento do satélite, cuja precisão deve ser menor que 0,05°. Então a equação (3.20) repetida abaixo:

$$\begin{bmatrix} \vec{\omega}_x \\ \vec{\omega}_y \\ \vec{\omega}_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -sen\theta \\ 0 & \cos\phi & sen\phi \cdot \cos\theta \\ 0 & -sen\phi & \cos\phi \cdot \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} - \omega_0 \cdot \begin{bmatrix} \cos\theta \cdot sen\psi \\ sen\phi \cdot sen\theta \cdot sen\psi + \cos\phi \cdot \cos\psi \\ \cos\phi \cdot sen\theta \cdot sen\psi - sen\phi \cdot \cos\psi \end{bmatrix} \tag{3.24}$$

pode ser simplificada adotando-se arredondamentos para as relações trigonométricas relacionada a pequenos ângulos:

$$\begin{aligned} \cos(\alpha) &\approx 1 \\ sen(\alpha) &\approx \alpha \end{aligned}$$

$$\begin{bmatrix} \vec{\omega}_x \\ \vec{\omega}_y \\ \vec{\omega}_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\theta \\ 0 & 1 & \phi \\ 0 & -\phi & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} - \omega_0 \cdot \begin{bmatrix} \psi \\ \phi \cdot \theta \cdot \psi + 1 \\ \theta \cdot \psi - \phi \end{bmatrix} \tag{3.25}$$

onde, uma vez que $\phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}, \omega_0$ são grandezas pequenas, e seus respectivos produtos podem ser desprezados, a eq. (3.25) fica:

$$\begin{bmatrix} \vec{\omega}_x \\ \vec{\omega}_y \\ \vec{\omega}_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} - \omega_0 \cdot \begin{bmatrix} \psi \\ 1 \\ -\phi \end{bmatrix} \tag{3.26}$$

Apesar das manobras de atitude neste trabalho partirem de ângulos iniciais de até 30° em "roll", a aproximação linearizando em torno de 0° não interferiu nos resultados. O modelo linearizado da planta pode, então, ser descrito por:

$$\begin{aligned} \ddot{\phi} &= \frac{T_{Ext\_x}}{I_{Sx}} + \frac{\dot{h}_{Rx}}{I_{Sx}} \\ \ddot{\theta} &= \frac{T_{Ext\_y}}{I_{Sy}} + \frac{\dot{h}_{Ry}}{I_{Sy}} \\ \ddot{\psi} &= \frac{T_{Ext\_z}}{I_{Sz}} + \frac{\dot{h}_{Rz}}{I_{Sz}} \end{aligned} \tag{3.27}$$

3.3.1.3 Ambiente

Usando uma terminologia semelhante aos trabalhos de Gobato (2007) e Moreira (2006), no Ambiente estarão definidos os torques externos atuantes sobre o satélite, os campos gravitacional e magnético da Terra, os Tempos Juliano e Gregoriano, a propagação dos elementos orbitais, e outros dados a serem considerados neste trabalho.

O Ambiente recebe como entrada os elementos keplerianos da órbita (semi-eixo maior, excentricidade, inclinação, ascenção reta do nodo ascendente, argumento do perigeu e anomalia média), e entrega como saídas: a posição e a velocidade orbitais do satélite, a velocidade orbital média e os torques externos T_{Ext_x}, T_{Ext_y} e T_{Ext_z}.

Adotando as mesmas considerações que Gobato (2007) para a estimativa dos torques externos, pode-se adotar um torque externo médio constante de $1x10^{-4}$ N.m em cada eixo, uma vez que o torque devido ao gradiente de gravidade é da ordem de $1x10^{-6}$ N.m, e o torque magnético é ainda menor. Portanto, o valor adotado pode ser entendido como conservativo.

3.3.1.4 Sensores

Um giroscópio é um dispositivo de medida baseado no Princípios da Conservação do Momento Angular. Através de uma massa girando em alta velocidade ao redor de um eixo, pode-se detectar mudanças na orientação inercial relativa ao eixo de giro (Figura 3.3).

Existem dois tipos básicos de instrumentos giroscópicos de medição (GOBATO, 2007):

- "Rate Gyros": medem velocidades angulares de um satélite. Com esses valores, e partindo-se de alguma condição inicial, um computador de bordo pode gerar uma estimativa de mudança de atitude do satélite por integração;
- "Rate Integrating Gyros": medem as mudanças de atitude do satélite diretamente, sem a necessidade de um computador de bordo.

FIGURA 3.3 – Giroscópio.
FONTE: Gobato (2007).

Quando três giroscópios, ou giros, são montados ortogonalmente entre si, e alinhados com o referencial do corpo, torna-se possível medir os componentes da velocidade angular de um veículo espacial. Este foi o caso considerado neste trabalho.

Com relação ao modelamento propriamente dito, foi adotado como hipótese simplificadora um ganho unitário, acrescido de um ruído gaussiano com amplitude de $1x10^{-5}$ rad/s, conforme sugerido por Gobato (2007).

3.3.1.5 Atuadores

Para o controle de atitude da PMM, os atuadores utilizados são rodas de reação.

Dentre os atuadores disponíveis na PMM, para o seu controle de atitude em Modo Normal são utilizadas rodas de reação.

Conforme Souza (1980) em Gobato (2007), as rodas de reação são usadas nos modos que necessitam de apontagem fina, por apresentarem a vantagem de produzir torques de baixíssimo valor e proporcionais aos ângulos de erro. Além disso, as elevadas freqüências de comutação desses torques não representam um problema, uma vez que o sistema consome energia, ao invés de massa. Outra vantagem é a possibilidade de serem fabricadas com baixos tempos de resposta.

Uma roda de reação é basicamente uma inércia rotativa, acoplada a um motor brushless DC. As Figuras 3.4 e 3.5 ilustram a montagem de uma roda desse tipo. Quando o motor é alimentado, a roda acelera provocando no corpo do satélite onde está montada uma rotação em direção oposta, produzindo torques e momentos usados para estabilizar o satélite um uma dada atitude. Uma desvantagem das rodas de reação é a existência de um limite máximo para as velocidades angulares – nessa condição a roda perde a capacidade de gerar torques de controle – necessitando periodicamente de um mecanismo de redução dessas velocidades, para restaurar a capacidade de produzir momentos angulares.

Para que a estabilização possa ser obtida nos eixos X, Y e Z, pelo menos três rodas de reação, ortogonais entre si, são necessárias. Normalmente uma quarta roda é adicionada ao sistema, geralmente com ângulos iguais em relação às outras três, com o objetivo de tornar o sistema robusto à falha simples.

FIGURA 3.4 – Roda de reação.
FONTE: CTA Space System.

FIGURA 3.5 – Roda de reação desmontada, mostrando massa de inércia e motor brushless DC.
FONTE: Bradford Engineering.

O modelo da roda de reação, bem como os valores característicos adotados neste trabalho seguem a sugestão de Souza (1980) em Gobato (2007), e são baseados em uma aproximação linear de um servomotor CC, cuja curva característica segue o padrão apresentado na Figura 3.6 ou seja, considerando a força contraeletromotriz na

capacidade de torque a ser transmitida ao corpo do satélite. O correspondente diagrama de blocos também está representado na mesma Figura 3.6.

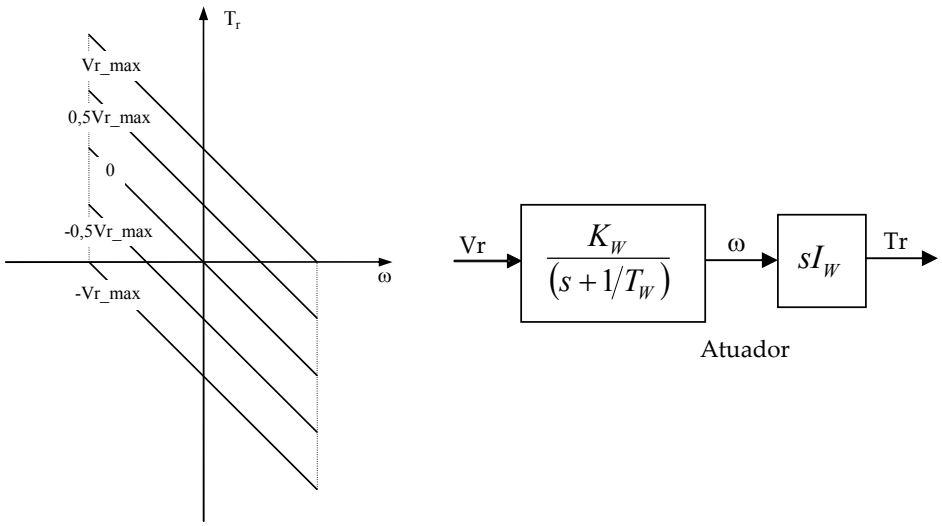


FIGURA 3.6 – Aproximação linear da curva característica do servomotor e diagrama de blocos correspondente.
FONTE: adaptado de Souza (1980).

Os dados correspondem a uma roda alternativa, utilizada por Souza (1980), e diferente do modelo especificado para a PMM, mas que simplifica a obtenção de um ajuste satisfatório para o controlador. São eles:

TABELA 3.1 – Parâmetros da roda de reação adotados neste trabalho.

I_R	0,0015 [kg.m²]	T_W = 20 [s]	K_W = 0.06 [N.m/V.s]
ω_{Rmax}	7500 [rpm]		
T_{Rmax}	0.6 [N.m]		
V_{Rmax}	10 [V]		

3.4 Projeto do Controlador de Atitude

A estratégia de controle adotada neste trabalho está baseada nas estruturas de controle clássico, e em linha com outros trabalhos similares (MOREIRA, 2006; GOBATO, 2007).

O controlador adotado será o PID, bastante popular e conhecido tanto no meio acadêmico como no industrial. Sua grande virtude e também o seu maior defeito, é a simplicidade. Para ajustar um PID padrão são necessários apenas 3 parâmetros, e conseqüentemente, a capacidade de lidar com ruídos e perturbações fica limitada.

Além de ser um controlador com uma extensa folha corrida, com mais de 60 anos de aplicação prática, ele tem merecido novos procedimentos de ajuste recentemente (ASTROM, 2001; COMINOS, 2002), tornando-o uma alternativa cada vez mais válida.

Um controlador PID é descrito pela seguinte função de transferência no domínio contínuo:

$$G_C(s) = P(s) + I(s) + D(s) = K_P + \frac{K_I}{s} + K_D \cdot s \tag{3.28}$$

Ou

$$G_C(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right) \tag{3.29}$$

onde K_p é o ganho proporcional, K_i o coeficiente de integração e K_d o coeficiente derivativo. T_i é conhecido como tempo de ação integral e T_d como tempo de ação derivativa.

Essa forma de implementação do controlador é conhecida como forma paralela, permitindo um razoável desacoplamento entre os parâmetros PID (COMINOS, 2002), e pode ser descrita através do seguinte diagrama de blocos:

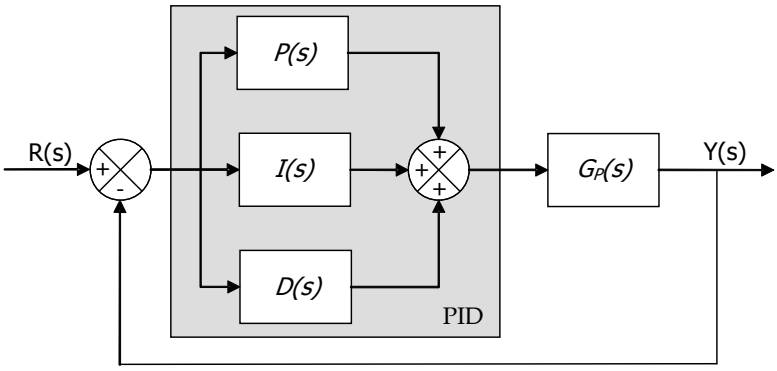


FIGURA 3.7 – Diagrama de blocos de um controlador PID.

Na prática, um PID normalmente é implementado na seguinte forma:

$$G_C(s) = K_p + \frac{K_i}{s} + \frac{K_d s}{1 + T_n s} \tag{3.30}$$

Essa implementação, apesar de ter a mesma resposta da versão original (3.28) em baixas freqüências, inclui um filtro passa-baixas no termo derivativo para reduzir a amplificação a ruídos. T_n representa a constante de tempo do filtro.

Dentre os vários métodos disponíveis para ajustar um PID (COMINOS, 2002), o primeiro e ainda bastante usado como ponto de partida é o método Ziegler-Nichols. O método original (depois houveram outras versões) é baseado na resposta a degrau do sistema em malha aberta.

Uma outra categoria de métodos de ajuste é baseado no estabelecimento de critérios, onde os parâmetros são escolhidos de forma a minimizar um determinado índice de desempenho. Esses índices de desempenho são calculados durante a efetiva atuação do controlador, ou seja, a qualidade do ajuste dos parâmetros é dependente das respostas da planta. Alguns do mais comuns são (COMINOS, 2002):

$$ISE = \int_0^T \left| r(t) - y(t) \right|^2 \cdot dt \tag{3.31}$$

$$IAE = \int_0^T |r(t) - y(t)| \cdot dt \tag{3.32}$$

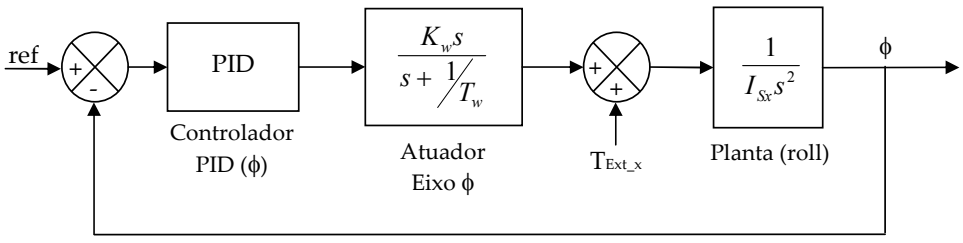
$$ITAE = \int_0^T |r(t) - y(t)| \cdot t \cdot dt \tag{3.33}$$

O ITAE "Integral-of-time-multiplied absolute error" (eq. 3.33) será o índice de desempenho adotado neste trabalho, onde o termo t dentro da integral implica considerar o tempo de rastreamento da referência do sistema como mais um dos fatores, além do erro de rastreamento da referência propriamente dito.

Como também pode ser observado do critério de desempenho adotado, ele não leva em consideração um outro parâmetro relevante na qualidade da resposta que é o valor de pico do sistema em resposta a um degrau na sua referência.

Assim, a estratégia de otimização dos parâmetros do controlador, a partir das respostas do sistema pode ser considerada como semi-automática, onde manualmente ajustam-se os parâmetros do controlador e observa-se a sobreelevação do sistema, e automaticamente é calculado o índice ITAE. Combinando a observação manual com a automática chega-se à conclusão sobre a necessidade de algum refinamento nos ajustes dos parâmetros do PID, ou não.

Para obter os parâmetros iniciais do controlador, procede-se inicialmente a uma linearização da planta, como descrito na Seção 3.2.1.3, chegando-se aos diagramas de blocos para os eixos (*roll*, *pitch*, *yaw*) de controle do satélite



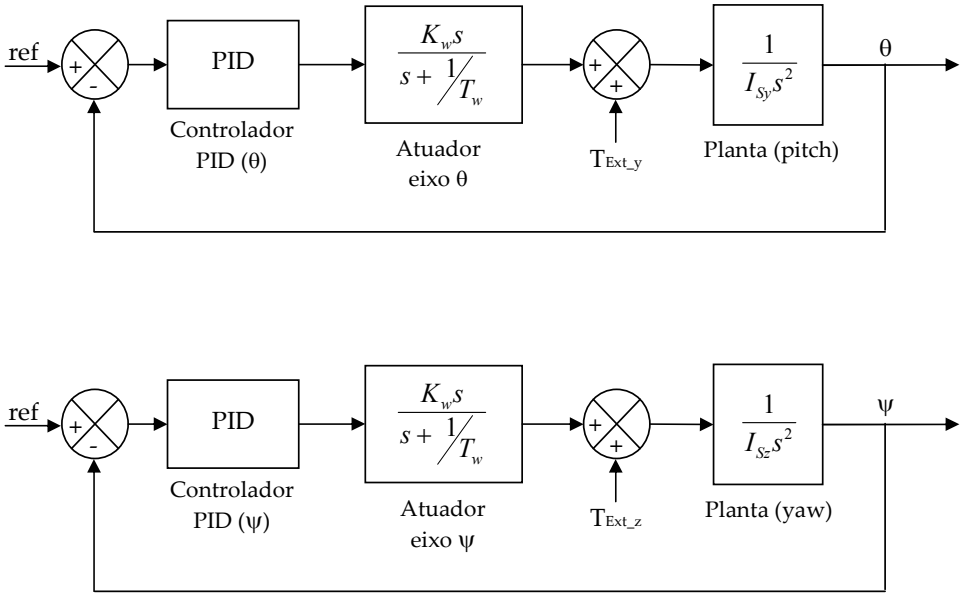


FIGURA 3.8 – Sistema de controle para planta linearizada, nos 3 eixos.

Segundo Ogata (1982), a forma ótima de uma função de transferência de 3º grau de malha fechada, baseada no critério ITAE é:

$$\frac{C(s)}{R(s)} = \frac{\omega_n^3}{s^3 + 1{,}75\omega_n s^2 + 2{,}15\omega_n^2 s + \omega_n^3} \tag{3.34}$$

Para o caso específico do sistema tratado neste trabalho, um valor de ω_n é estimado inicialmente a partir de:

$\zeta = 0{,}71$ (compromisso entre velocidade de resposta e sobreelevação), e t_S = 92s.

Assim chegando-se a $\omega_n = \frac{4}{\zeta \cdot t_S} = \frac{4}{0{,}71 \cdot 92} = 0{,}0612$ rad/s.

Com isso, a equação 3.34 fica:

$$\frac{C(s)}{R(s)} = \frac{0{,}0002296}{s^3 + 0{,}1072 s^2 + 0{,}008062 s + 0{,}0002296} \tag{3.35}$$

Para esses ajustes, obtém-se a resposta a degrau conforme a Figura 3.9, e o diagrama de "root locus" apresentado na Figura 3.10.

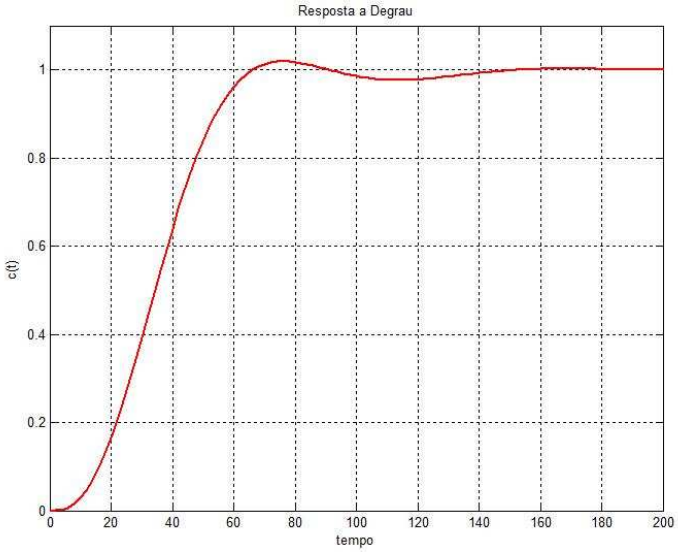


FIGURA 3.9 – Resposta a degrau da forma ótima da FTMF de 3ª ordem.

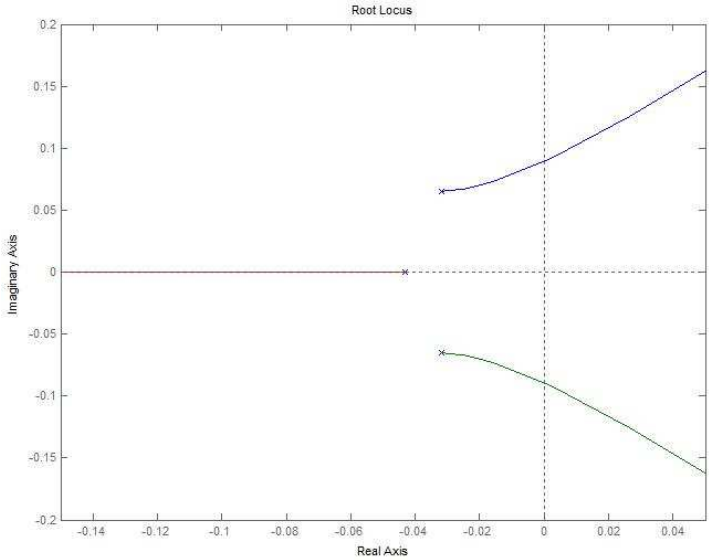


FIGURA 3.10 – *Root Locus* da forma ótima da FTMF de 3ª ordem.

Da Figura 3.8, a função de transferência de malha fechada para o eixo *roll* (ϕ) é:

$$\frac{\phi(s)}{ref(s)} = \frac{K_w}{I_{Sx}} \left[\frac{K_{d\_x} \cdot s^2 + K_{p\_x} \cdot s + K_{i\_x}}{s^3 + \left(\frac{K_w \cdot K_{d\_x}}{I_{Sx}} + \frac{1}{T_w} \right) \cdot s^2 + \frac{K_w \cdot K_{p\_x}}{I_{Sx}} \cdot s + \frac{K_w \cdot K_{i\_x}}{I_{Sx}}} \right] \tag{3.36}$$

onde o controlador PID é considerado como o representado pela equação 3.28, cujos ganhos P, I e D estão representados por K_{p_x}, K_{i_x} e K_{d_x}, respectivamente.

Para os eixos *pitch* e *yaw*, a forma da FTMF é a mesma, considerando que nesses casos I_{sx} é substituído por I_{sy} e I_{sz}, e também K_{p_x}, K_{i_x}, K_{d_z} por K_{p_y}, K_{i_y}, K_{d_y} e K_{p_z}, K_{i_z}, K_{d_z}, respectivamente.

Como a equação 3.36 possui zeros na função de transferência, inexistentes na equação 3.35, é esperado que a resposta a degrau das duas equações, ajustadas com o mesmo polinômio no denominador, sejam diferentes. De qualquer forma, os valores de 3.35 podem ser usados como ponto de partida. Assim, comparando ambos os denominadores, chega-se a parâmetros iniciais do PID para o controle do ângulo ϕ:

$$\begin{aligned} K_{p\_x} &= 0{,}008062 \cdot \frac{I_{Sx}}{K_w} = 39{,}7357 \\ K_{i\_x} &= 0{,}0002296 \cdot \frac{I_{Sx}}{K_w} = 1{,}1318 \\ K_{d\_x} &= \frac{I_{Sx}}{K_w} \left(0{,}1072 - \frac{1}{T_w} \right) = 281{,}9102 \end{aligned} \tag{3.37}$$

E portanto, substituindo a eq. 3.37 na eq. 3.36, chega-se a:

$$\frac{\phi(s)}{ref(s)} = \frac{0{,}0572 \cdot s^2 + 0{,}008062 \cdot s + 0{,}0002296}{s^3 + 0{,}1072 \cdot s^2 + 0{,}008062 \cdot s + 0{,}0002296} \tag{3.38}$$

com sua resposta a degrau conforme a Figura 3.11.

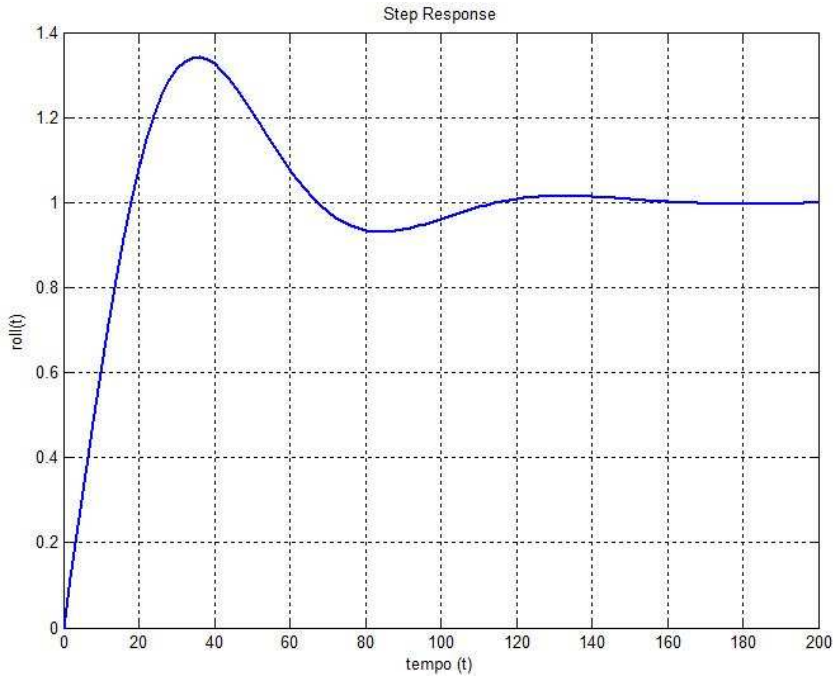


FIGURA 3.11 – Resposta a degrau de $\phi(s)/ref(s)$ com ganhos PID dados por 3.37.

De forma análoga, os parâmetros iniciais para o controle dos ângulos θ e ψ são obtidos e estão condensados na Tabela 3.2, abaixo:

TABELA 3.2 – Parâmetros dos controladores PID.

ParâmetroPID / *ângulo*	K_p	K_i	K_d
ϕ	K_{p_x} = 39,7357	K_{i_x} = 1,1318	K_{d_x} = 281,9102
θ	K_{p_y} = 67,367	K_{i_y} = 1,9186	K_{d_y} = 477,9727
ψ	K_{p_z} = 49,0196	K_{i_z} = 1,396	K_{d_z} = 347,795

Devido à sobreelevação observada, será necessário refinar os ganhos do PID. Esse ajuste será realizado com um modelo mais detalhado do satélite, apresentado no

Capítulo 4. O gráfico de *Root Locus* da FTMF de *roll* (eq. 3.38), para os ganhos dados por 3.37 pode ser visto na Figura 3.12 a seguir.

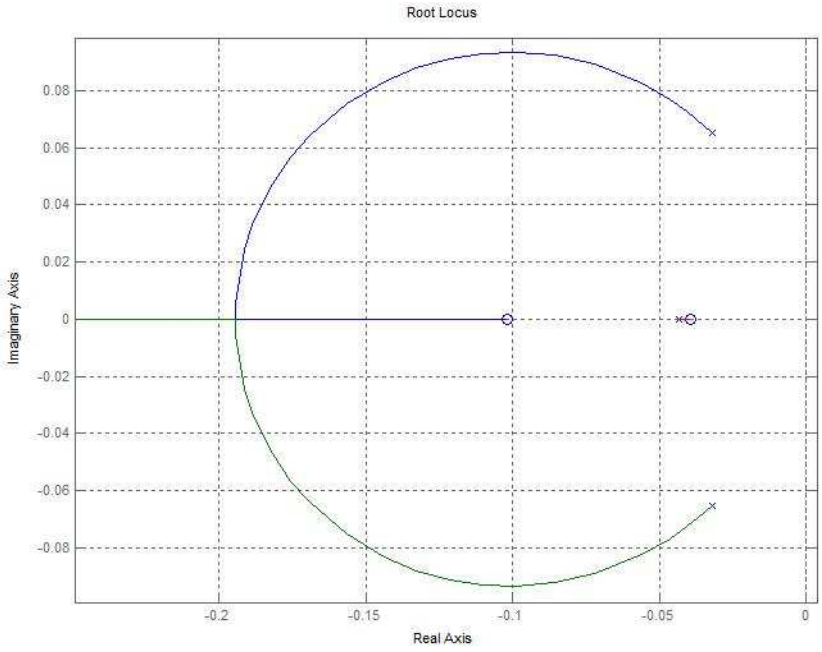


FIGURA 3.12 – *Root Locus* de $\phi(s)/ref(s)$ com ganhos PID dados por 3.37.

CAPÍTULO 4

SIMULAÇÃO EM TEMPO VIRTUAL

Neste capítulo, na seqüência do processo de desenvolvimento proposto, os modelos matemáticos desenvolvidos no Capítulo 3 são desdobrados em algoritmos integrados e simulados, quando necessário dentro de um ambiente de simulação em tempo virtual, gerando-se os resultados correspondentes. A organização do modelo integrado busca minimizar alterações na etapa final, onde esse mesmo sistema será executado em um ambiente de simulação distribuída, em tempo real, com uma parte sendo executada em um FPGA.

4.1 Ambiente de Simulação

Conforme antecipado no Capítulo 3, o ambiente de simulação utilizado para a implementação da modelagem do problema proposto é o Matlab/Simulink. Através dele, pode-se construir modelos usando interfaces gráficas em um formato parecido com um diagrama de blocos – parte Simulink – integrada com recursos de programação em uma linguagem de script, através de linhas de comando – parte Matlab.

Coordenando esses dois recursos, é possível inicializar parâmetros da simulação através do Matlab; e elaborar, executar e testar o modelo no Simulink. Durante a execução da simulação está-se em um modo de simulação conhecido como "as fast as possible" (o mais rápido possível), onde a preocupação com o tempo de simulação corresponder ao tempo real, ou físico, ainda não existe, sendo apenas uma referência virtual dentro do ambiente.

4.2 Modelos de Simulação

O modelo da aplicação espacial pode ser visto na Figura 4.1, onde a maioria dos blocos existentes corresponde a implementações das equações obtidas no Capítulo 3. Por exemplo, no bloco *Sensors* (Figura 4.10), está implementado um sensor de ganho unitário acrescido de um ruído gaussiano, conforme definido na Seção 3.3.1.5.

Também estão presentes na Figura 4.1 alguns recursos utilizados para monitoramento on-line da resposta do sistema, bem como registro dos dados no ambiente Matlab, para posterior geração dos gráficos de respostas, apresentados na Seção 4.3.

Para a geração desses gráficos, os blocos *ToWorkspace_1* a *ToWorkspace_5*, transferem as suas entradas para variáveis definidas no ambiente Matlab.

Para monitorar o critério de desempenho ITAE, o bloco *ITAE_Calculation* implementa o critério apresentado na equação 3.33, conforme Figura 4.11. Os resultados numéricos podem ser visualizados, durante a execução da simulação, no bloco *PerformanceIndex*.

O integrador utilizado foi o Runge-Kutta, 4ª ordem, e o passo da simulação foi ajustado para ser fixo, com período em 10ms.

4.2.1 Bloco Ambiente

A Figura 4.2 apresenta o diagrama do bloco *Ambiente*, onde na sua parte superior está a propagação da órbita do satélite, e na inferior a geração dos torques ambientais.

Para a propagação de órbita foram utilizadas as equações apresentadas por Kuga (1995).

4.2.2 Bloco BodyDynamics

Neste bloco, a dinâmica do satélite é calculada. Ela é baseada na equação 3.13, mas é desdobrada nos seus componentes escalares.

Mais exatamente, a implementação dessa equação está detalhada nos blocos *Eq_Euler* e *ReactionWheelMoments*, das Figuras 4.4 e 4.5, respectivamente. Na analogia com os termos da equação 3.13, no bloco *Eq_Euler* temos:

- Para $\dot{\vec{H}}_R$: T_rw;
- Para $\vec{T}_{Ext}$: Tamb;
- Para $\vec{\omega}_S \times (I_R \cdot \vec{\omega}_{RS})$: w_rw. No bloco *ReactionWheelMoments*, são calculados os termos escalares da operação vetorial;

Os valores das constantes de inércia do satélite utilizadas no modelo, nos seus eixos principais de inércia, são:

$$I_{Sx} = 295{,}71[kg.m^2],\ I_{Sy} = 501{,}37[kg.m^2] \text{ e } I_{Sz} = 364{,}82[kg.m^2];$$

4.2.3 Bloco BodyCinematics

A cinemática do satélite é calculada no bloco *BodyCinematics*, apresentado na Figura 4.6, em ângulos de Euler, a partir da equação 3.21. Nesse bloco, a matriz de rotação é construída, sendo efetuadas as operações matriciais para a solução da equação diferencial, a cada passo de execução.

Para a construção da matriz foi usado um bloco existente em um dos pacotes adicionais do Simulink: o *Aerospace Blockset*; os outros blocos fazem parte do pacote padrão.

4.2.4 Bloco ReactionWheels

As dinâmicas das três rodas de reação estão modeladas no bloco *ReactionWheels*, conforme a Figura 4.7, seguindo a estrutura apresentada na Figura 3.6, com valores numéricos da Tabela 3.1. Ainda, representando a fcem, o ganho K é definido como a

tensão máxima de saída do controlador pela velocidade máxima (em RPM) da roda de reação.

O modelo de cada roda está discriminado em paralelo porque, em princípio, o bloco inteiro ou pelo menos uma parte está previsto para ser executado em FPGA quando o sistema estiver sendo executado em tempo real.

4.2.5 Bloco AttitudeController

Neste bloco está implementado o controlador de atitude, correspondendo às Figuras 4.8 e 4.9. A estrutura segue o padrão já comentado para as rodas de reação.

O algoritmo em si, Figura 4.9, segue a estrutura paralela, conforme apresentado no Capítulo 3, equação 3.28 e Figura 3.7.

A diferença é a implementação, já na forma digital. Nela foram usados blocos do pacote padrão de funcionalidades do Simulink. No termo integral, optou-se por adotar a forma trapezoidal, com saturador específico. A integração trapezoidal também pode ser obtida pela transformação de Tustin, ou bilinear, do termo 1/s do mundo contínuo, com o mesmo resultado (ISERMAN, 1989). O saturador no integrador é um mecanismo para impedir que, ao sair da saturação, esse termo ainda permaneça limitado durante um tempo excessivo, comprometendo a capacidade de resposta do controlador. Ambos os saturadores usados no PID, tanto o da saída como o do termo I foram ajustados para ±10 (V), representando os valores numéricos de uma saída D/A conectada à entrada de uma roda de reação.

Para o termo derivativo, usou-se o algoritmo mais simples. De fato, ele é o equivalente a aplicar-se a transformada Z a um termo s do mundo contínuo.

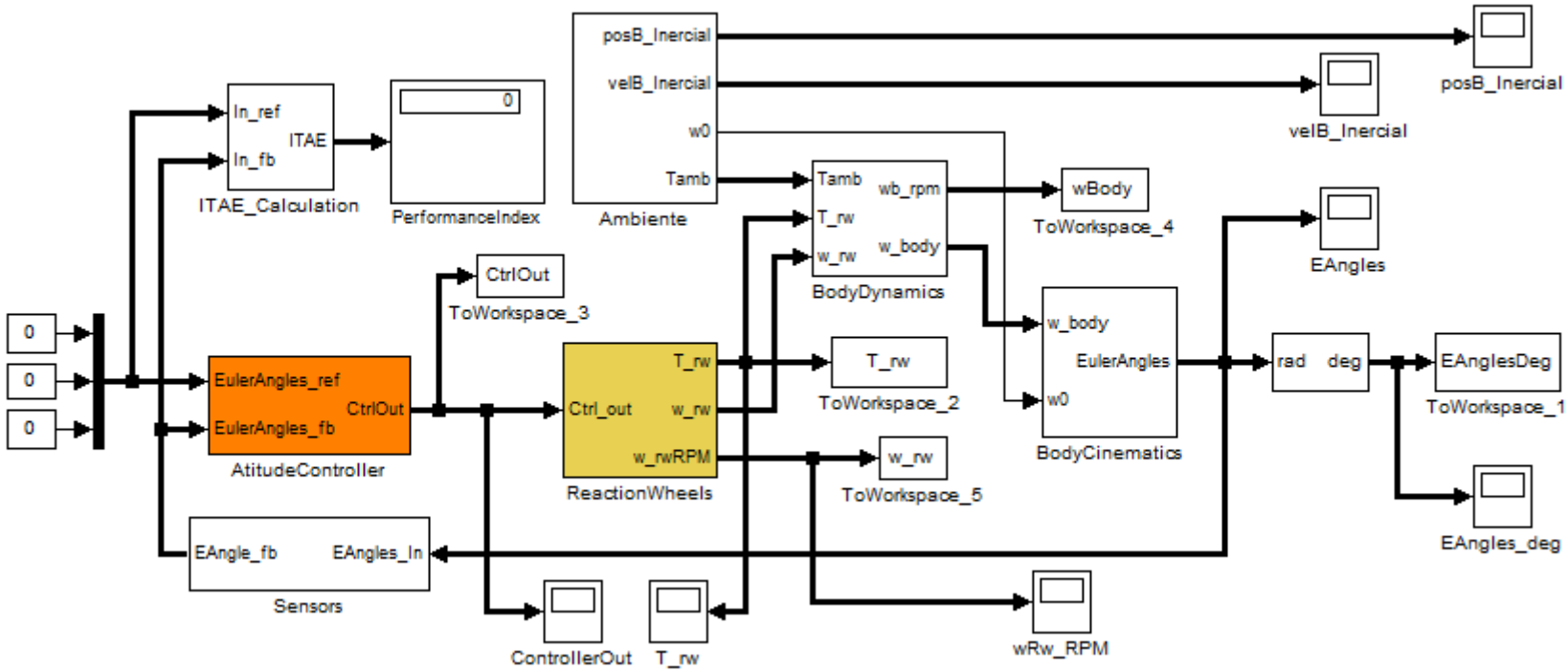


FIGURA 4.1 – Modelo da aplicação espacial: diagrama principal.

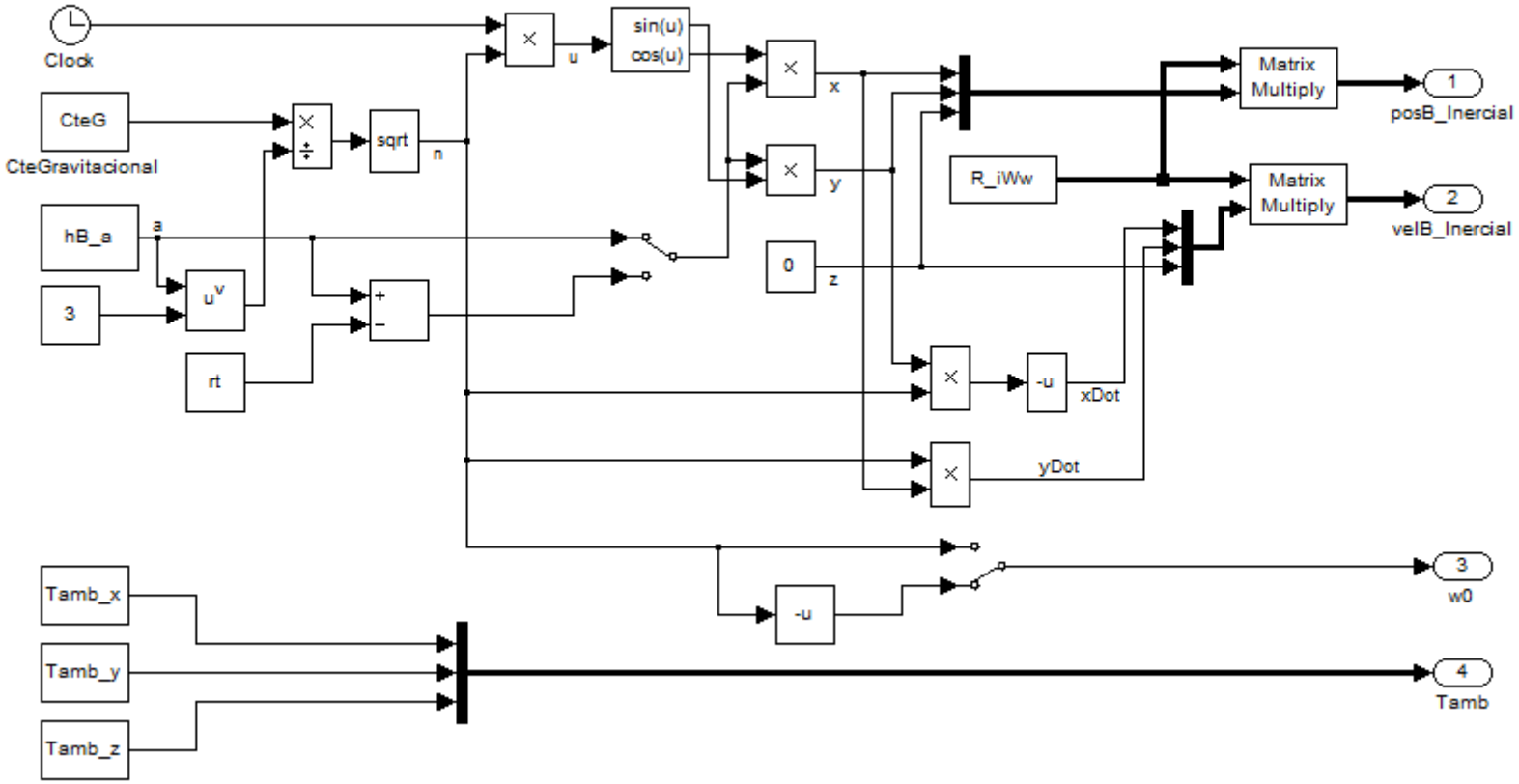


FIGURA 4.2 – Modelo da aplicação espacial: bloco Ambiente.

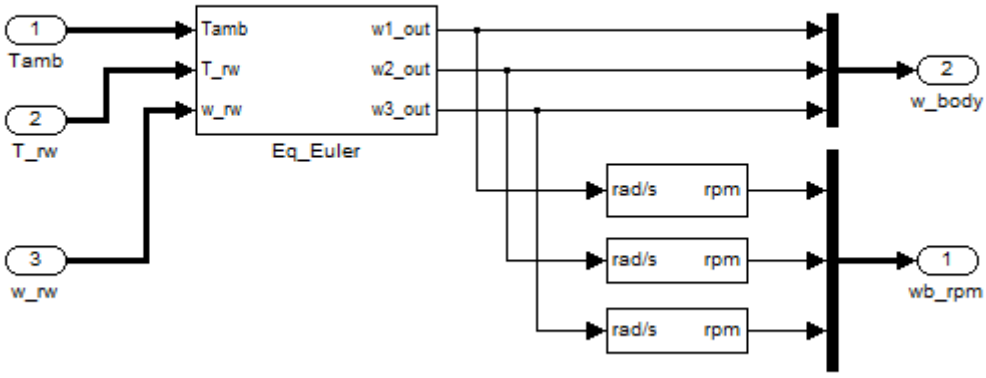


FIGURA 4.3 – Modelo da aplicação espacial: bloco BodyDynamics.

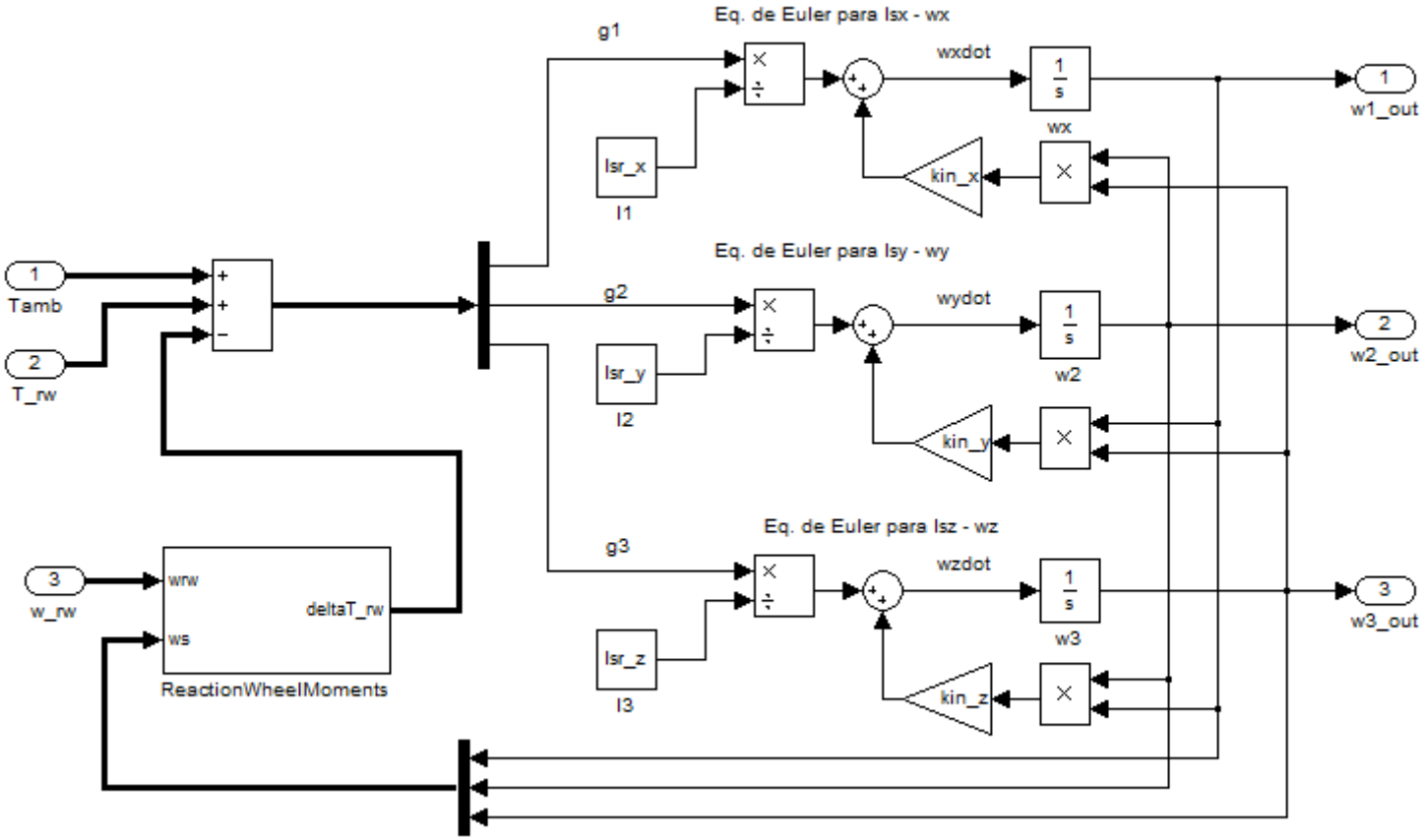


FIGURA 4.4 – Modelo da aplicação espacial: bloco Eq_Euler, dentro de BodyDynamics.

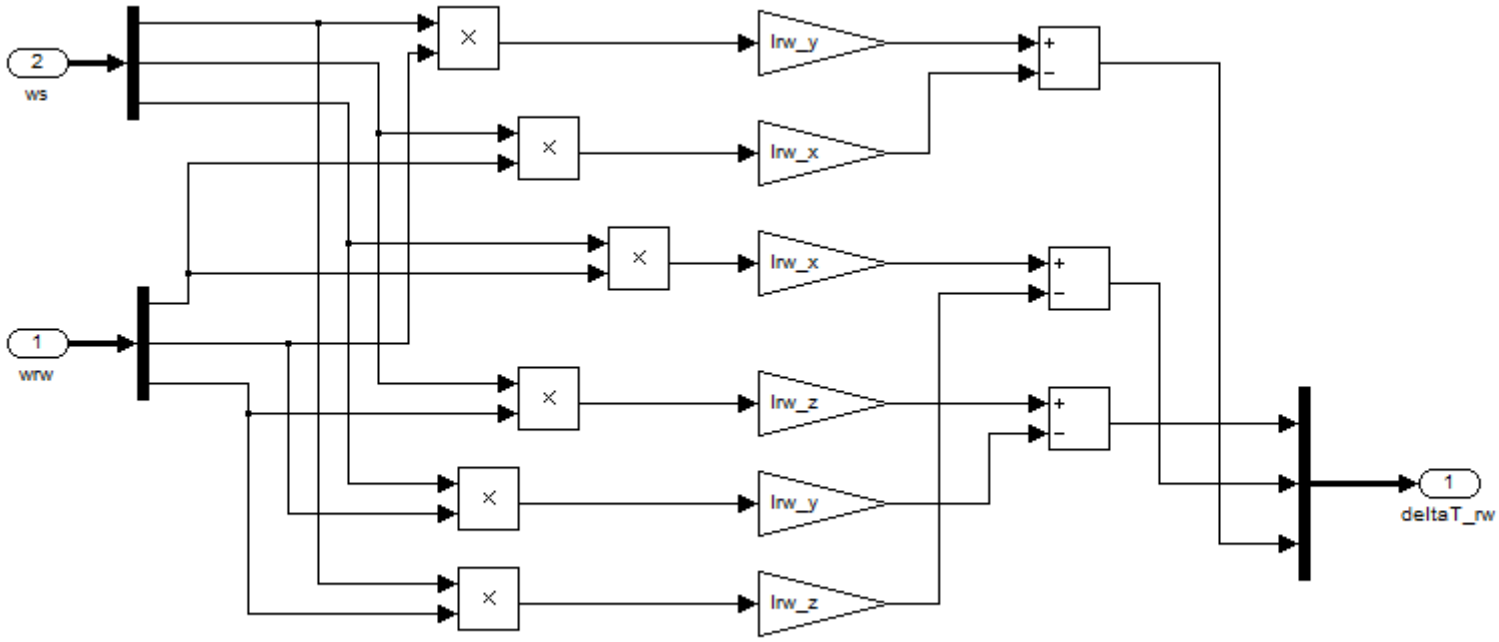


FIGURA 4.5 – Modelo da aplicação espacial: bloco ReactionWheelMoments, dentro de Eq_Euler.

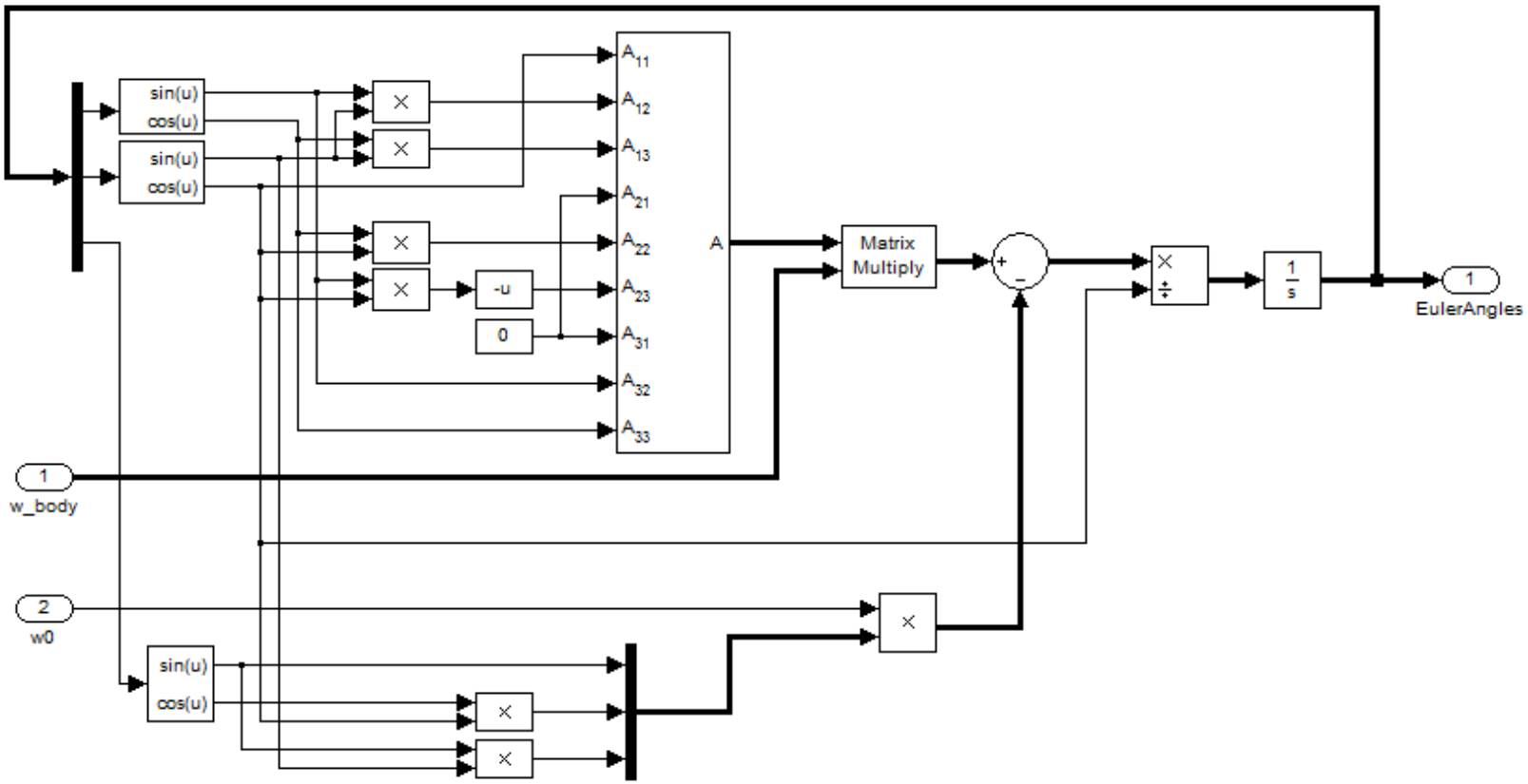


FIGURA 4.6 – Modelo da aplicação espacial: bloco BodyCinematics.

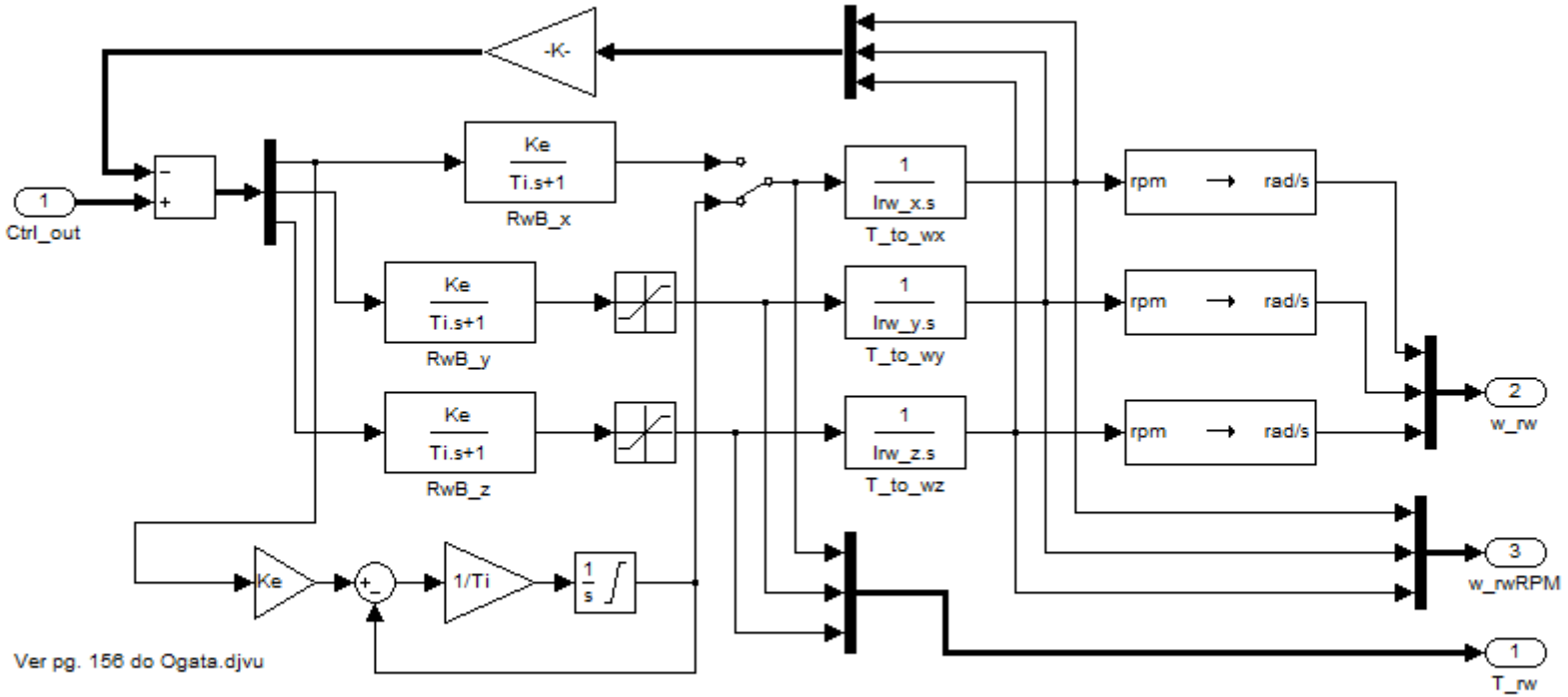


FIGURA 4.7 – Modelo da aplicação espacial: bloco ReactionWheels.

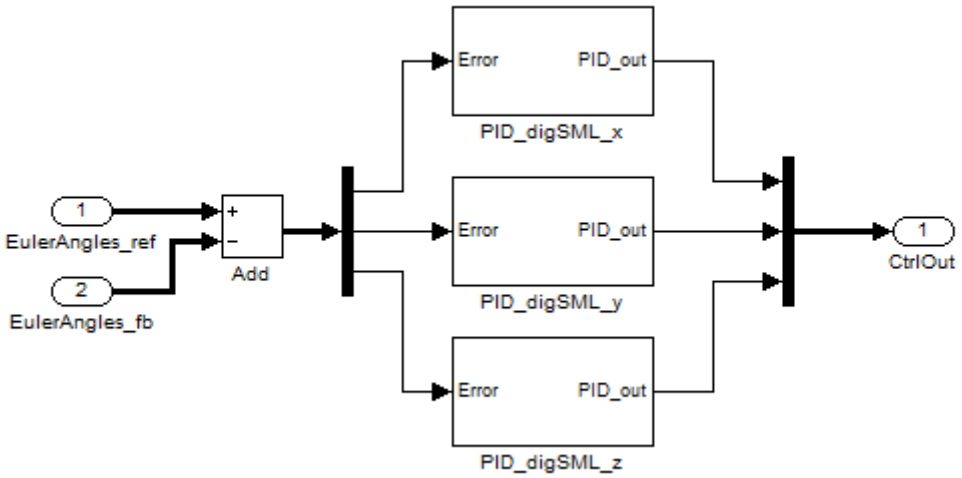


FIGURA 4.8 – Modelo da aplicação espacial: bloco AttitudeController.

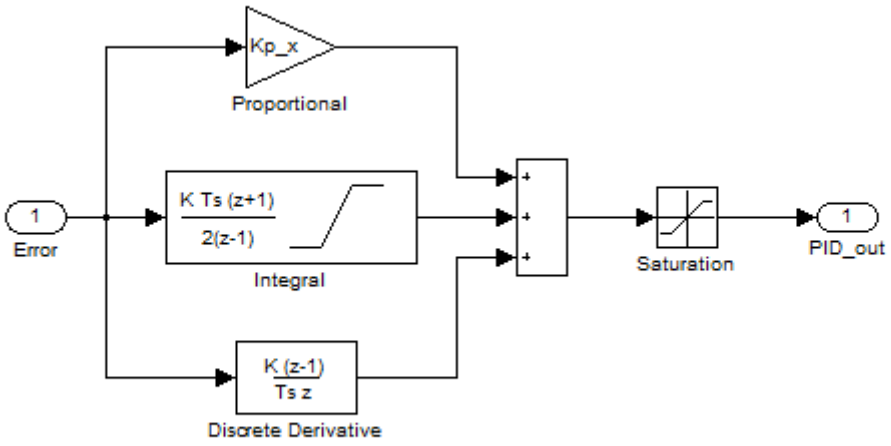


FIGURA 4.9 – Modelo da aplicação espacial: bloco PID_digDML_x, dentro de AttitudeController.

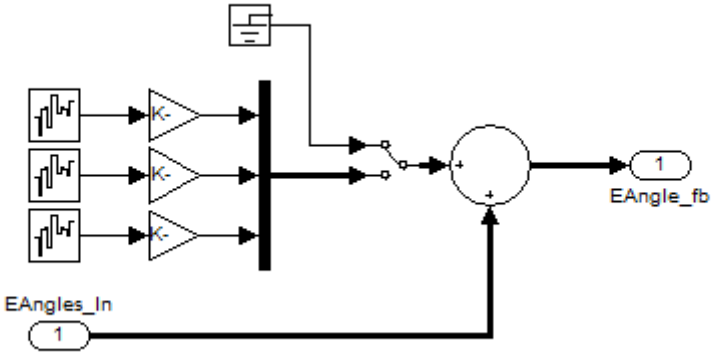


FIGURA 4.10 – Modelo da aplicação espacial: bloco Sensors.

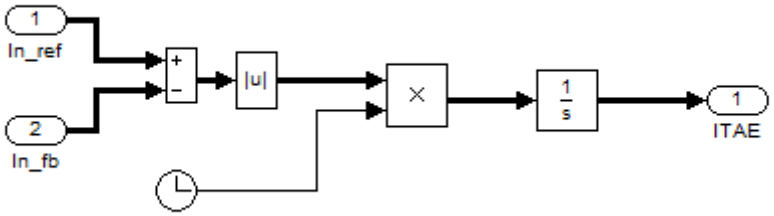


FIGURA 4.11 – Modelo da aplicação espacial: bloco ITAE_Calculation.

4.3 Resultados da Simulação

As simulações realizadas procuram demonstrar o cumprimento dos requisitos enunciados na Seção 3.2. Assim, a manobra adotada supõe que, inicialmente, o satélite está desalinhado em 30° com relação ao seu eixo de rolagem (ϕ = 30°; ψ = θ = 0°), e deseja-se realinhá-lo com o referencial VHLH (ϕ = ψ = θ = 0°).

O passo de integração adotado foi de T=10ms para o sistema inteiro, ou seja, os modelos dos torques externos, satélite, rodas de reação e controlador digital dos três

eixos, são executados com o mesmo passo, utilizando o integrador Runge-Kutta de 4ª ordem, quando aplicável.

Com os valores iniciais dos parâmetros dos controladores de atitude obtidos no Capítulo 3 (Tabela 3.2), mas agora usando um modelo de maior complexidade, observa-se a resposta do sistema nas Figuras 4.12 e 4.13. Na Figura 4.14 pode-se observar ainda o comportamento do torque da roda de reação para o eixo de *roll*. Conclui-se, portanto, ser desejável refinar os ganhos do controlador, de forma a reduzir a sobreelevação, que ficou em torno de 50%, e melhorar o tempo de acomodação, uma vez que o requisito para o valor de posição ficar dentro de ±0,05° em 180s não foi atendido. A Tabela 4.1 apresenta o índice de desempenho ITAE para os três eixos, calculados (conforme Eq. 3.33) durante toda a execução da simulação, ou seja, 400s.

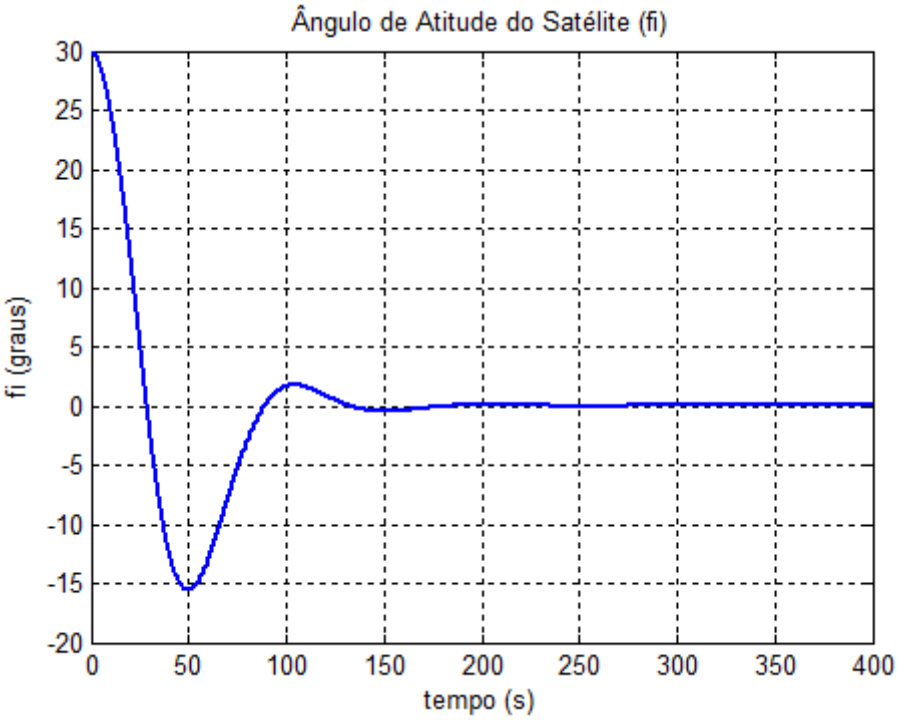


FIGURA 4.12 – Realinhamento do satélite, ângulo ϕ, com parâmetros do controlador calculados a partir do sistema linearizado.

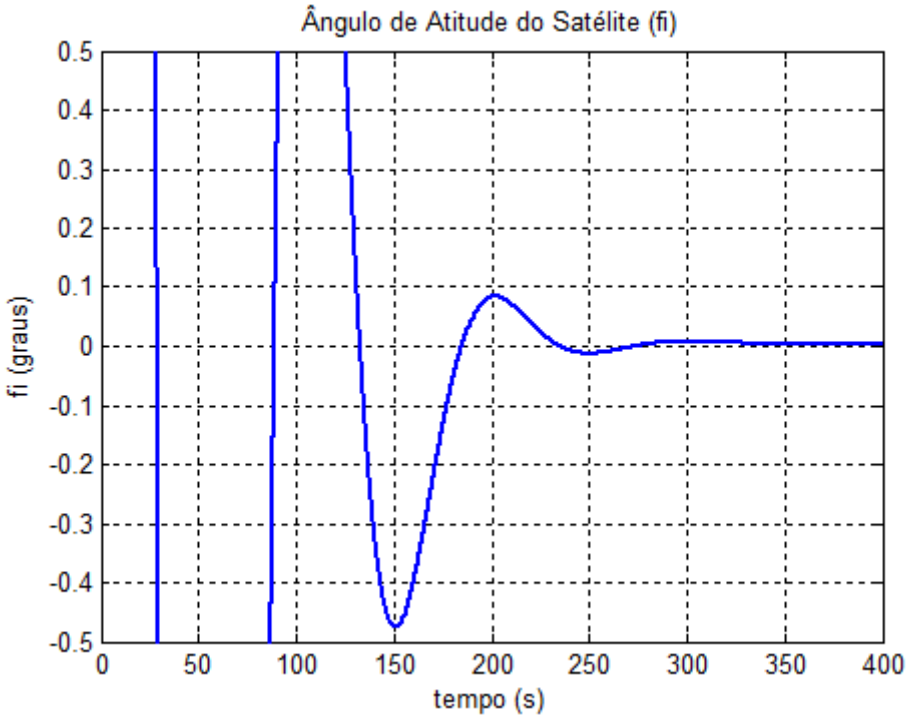


FIGURA 4.13 – Realinhamento do satélite, ângulo ϕ, detalhe da Figura 4.12.

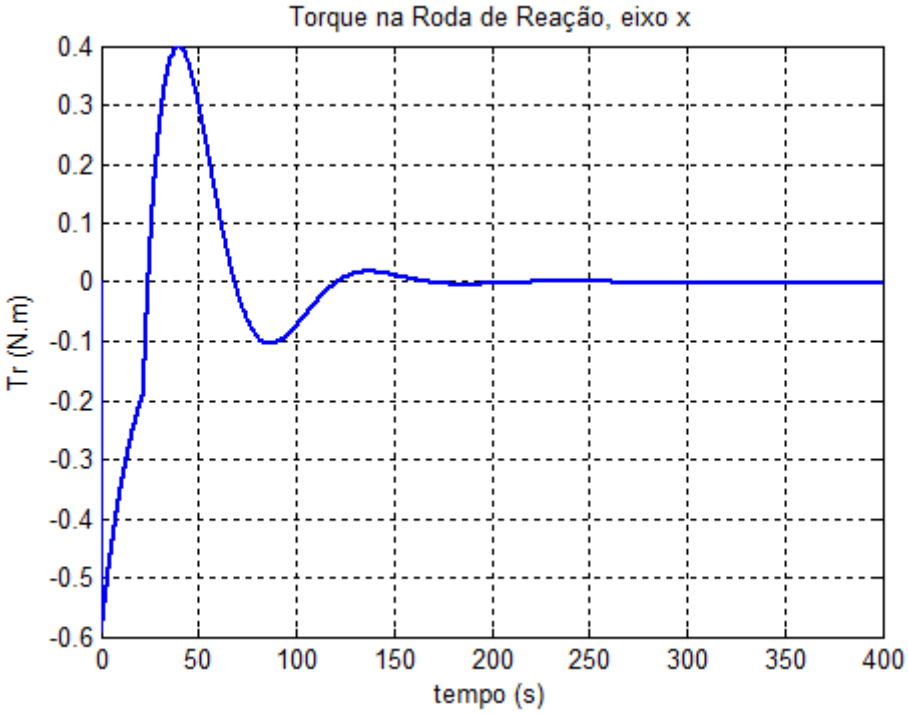


FIGURA 4.14 – Torque na roda de reação, eixo X.

TABELA 4.1 – Índice ITAE do sistema simulado com PID conforme tabela 3.2.

	ϕ	θ	ψ
ITAE	747.1	5.223	16.6

4.3.1 Refinamento na Resposta do Controlador de Atitude

Usando como referência os ganhos PID obtidos no Capítulo 3, chegou-se a novos ganhos, apresentados na Tabela 4.2. Esses ganhos foram obtidos através de estimativas, usando o critério ITAE para auxiliar na decisão sobre refinamentos adicionais.

Durante o processo de ajuste, os ganhos chegaram a ser 200 vezes maiores que os da Tabela 3.2, com a resposta do sistema atendendo com folga às especificações, o índice ITAE indicando uma melhora significativa (ITAE[ϕ] = 112.1), mas a roda de reação apresentando saturação. Assim, mantendo os valores dos ganhos proporcional e integral constantes (aumento de 200 vezes), mas aumentando ainda mais o ganho derivativo, a roda foi saindo da saturação, até atingir os resultados apresentados nas figuras a seguir. Nessa condição, o ganho derivativo ficou 500 vezes maior que o obtido no Capítulo 3, conforme a Tabela 4.2, e o índice ITAE correspondente na Tabela 4.3.

Nas Figuras 4.15 a 4.29 pode-se observar o resultados obtidos com os novos ganhos PID.

As Figuras 4.15 e 4.16 apresentam o comportamento da atitude do satélite no eixo de rolagem, através do ângulo ϕ , com as condições iniciais (30°, 0°, 0°). Especialmente na Figura 4.16, pode-se observar o atendimento aos requisitos de precisão de apontamento de ±0,05° em menos de 180s. De fato, a partir de 105.4 s essa precisão foi atingida.

As Figuras 4.17 e 4.18 mostram o comportamento do satélite para os outros dois eixos: θ e ψ.

TABELA 4.2 – Ganhos PID com sistema modelado completo.

ParâmetroPID / *ângulo*	K_p	K_i	K_d
ϕ	K_{p_x} = 7947,1	K_{i_x} = 226,36	K_{d_x} = 140960
θ	K_{p_y} = 13473	K_{i_y} = 383,72	K_{d_y} = 238990
ψ	K_{p_z} = 9803,9	K_{i_z} = 279,2	K_{d_z} = 173900

TABELA 4.3 – Índice ITAE do sistema simulado com PID conforme tabela 4.2.

	ϕ	θ	ψ
ITAE	247.4	0.02374	0.8256

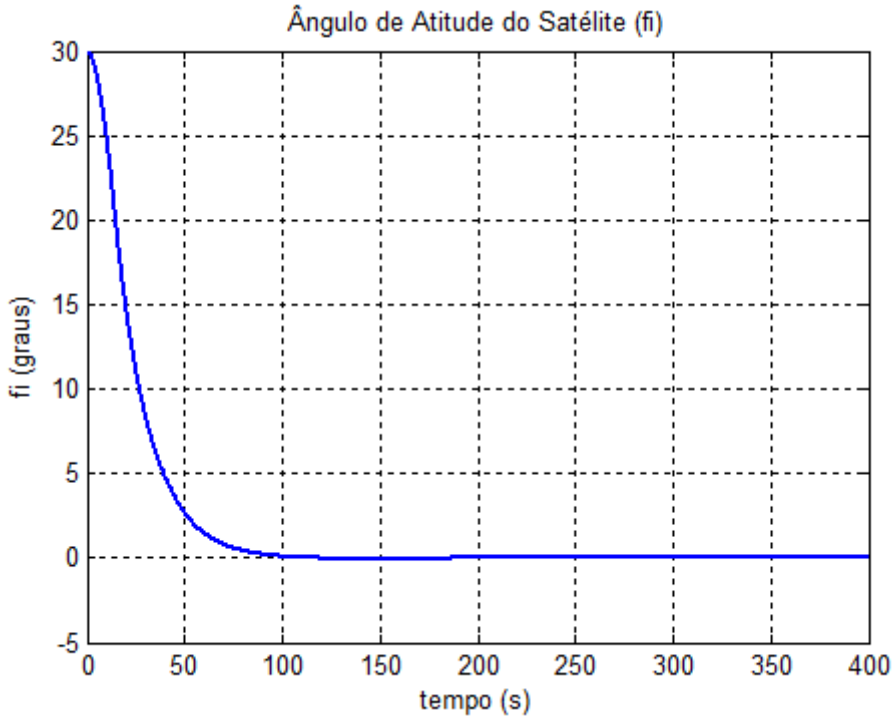


FIGURA 4.15 – Realinhamento do satélite, ângulo ϕ, com parâmetros do controlador conforme a Tabela 4.2.

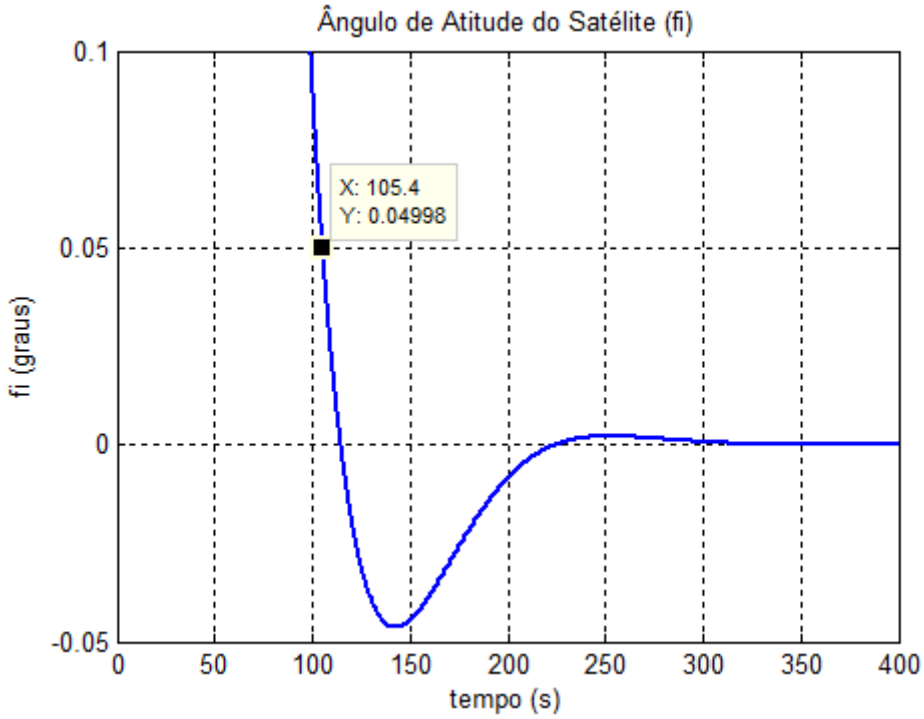


FIGURA 4.16 – Realinhamento do satélite, ângulo ϕ, detalhe da Figura 4.15.

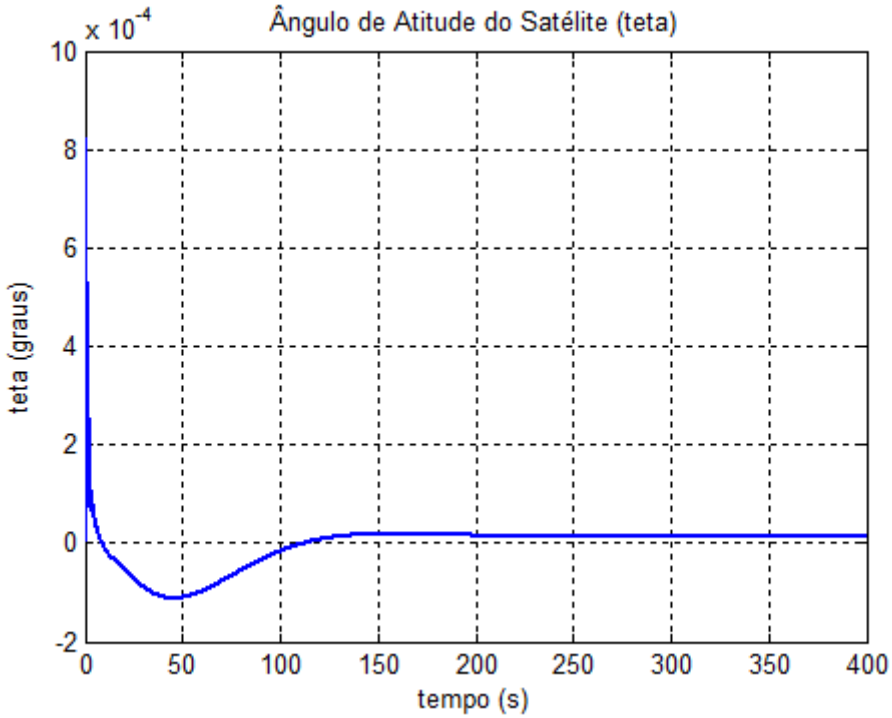


FIGURA 4.17 – Realinhamento do satélite, ângulo θ.

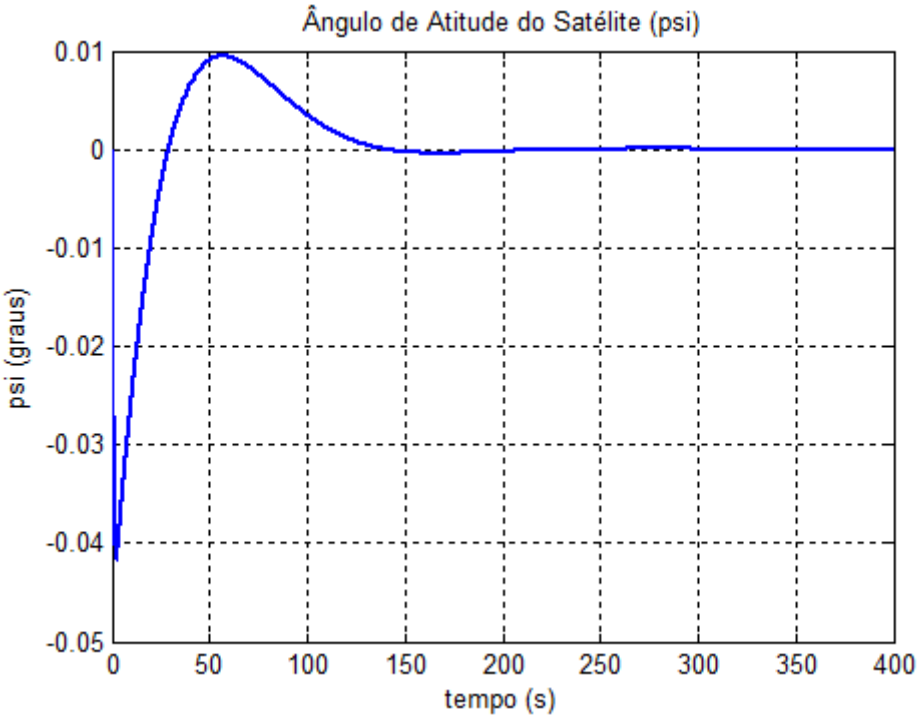


FIGURA 4.18 – Realinhamento do satélite, ângulo ψ.

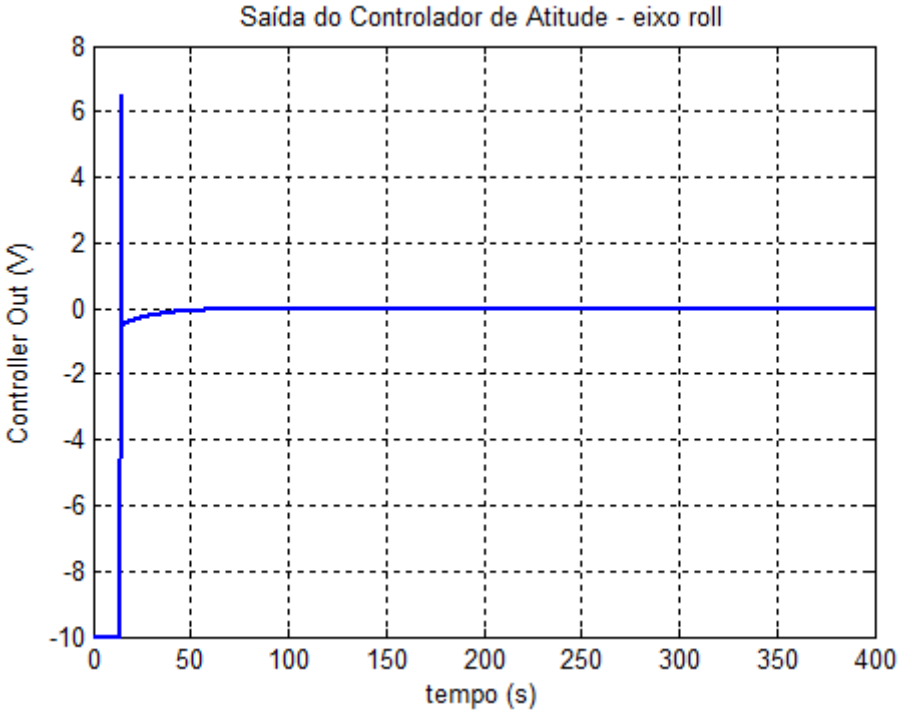


FIGURA 4.19 – Saída do controlador de atitude, eixo ϕ.

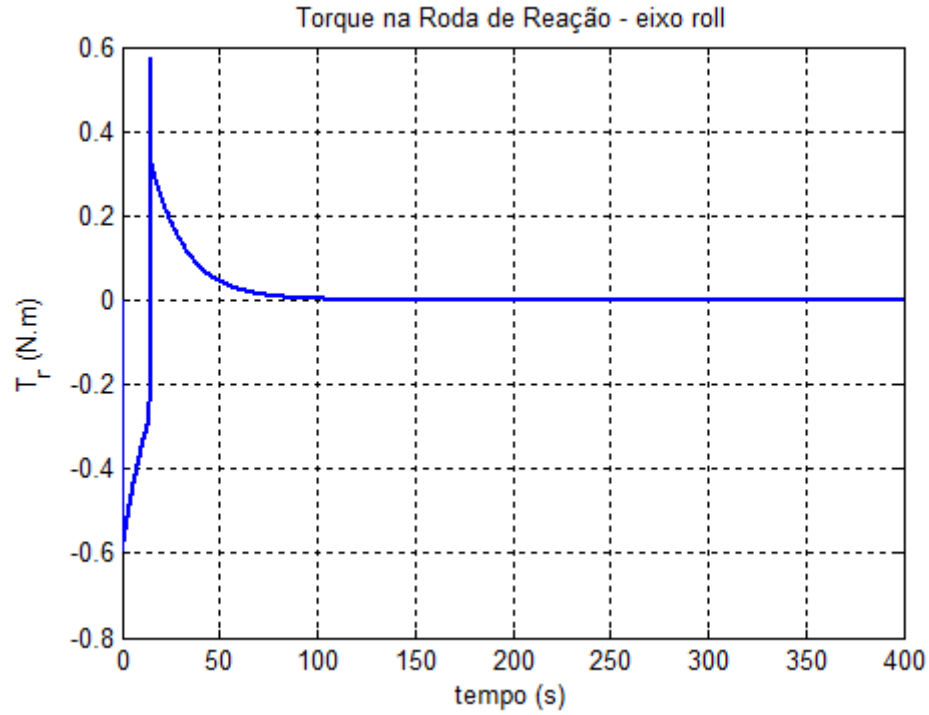


FIGURA 4.20 – Torque na roda de reação, eixo X.

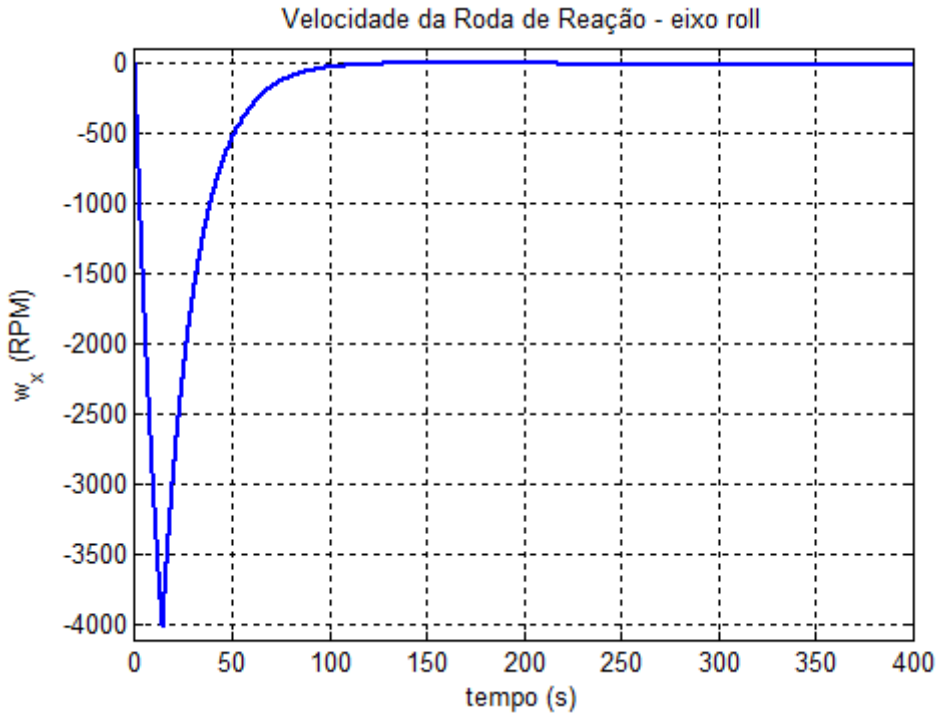


FIGURA 4.21 – Velocidade da roda de reação, eixo X.

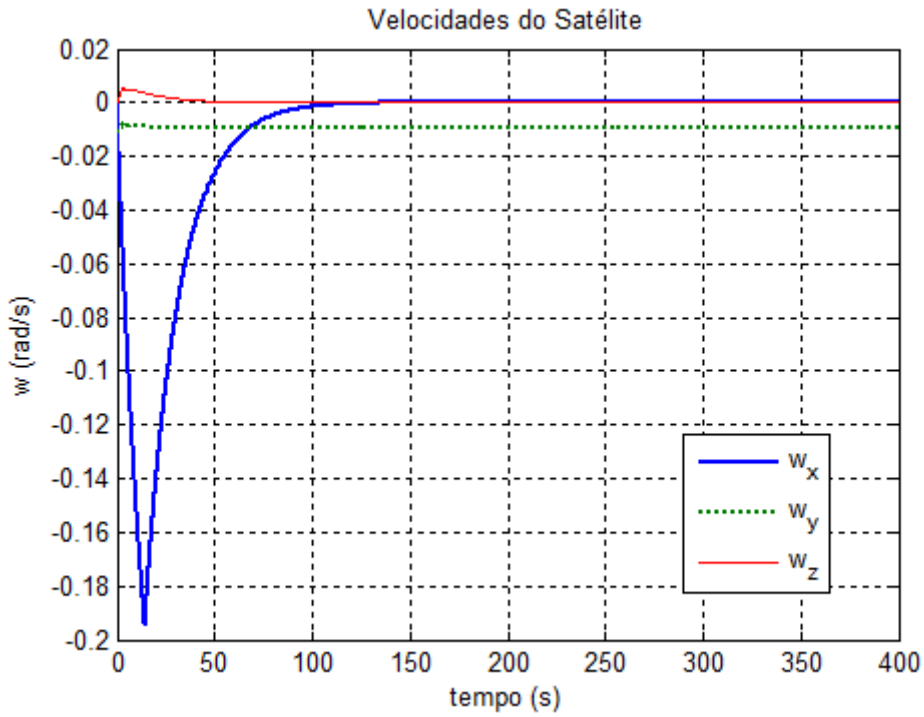


FIGURA 4.22 – Realinhamento do satélite, velocidades do corpo, em rad/s.

Nas Figuras 4.19 a 4.21 pode-se observar o comportamento de outras variáveis de interesse para o eixo de "roll", como a saída do controlador de atitude, o torque produzido pela roda de reação e a sua velocidade, respectivamente.

Ainda na Figura 4.22 estão apresentadas as velocidades angulares do satélite durante a manobra, com as condições iniciais de (0, -0.011, 0) rad/s.

As Figuras 4.23 a 4.26 mostram a mesma manobra apresentada nas figuras anteriores, mas com os sensores de posição sob a influência do ruído gaussiano, através da mudança na chave de seleção existente no modelo dos sensores, contido na Figura 4.10. A Tabela 4.4 mostra o índice ITAE nessa configuração.

TABELA 4.4 – Índice ITAE, PID conforme Tabela 4.2 e sensores com ruído gaussiano.

	ϕ	θ	ψ
ITAE	262.2	6.304	7.055

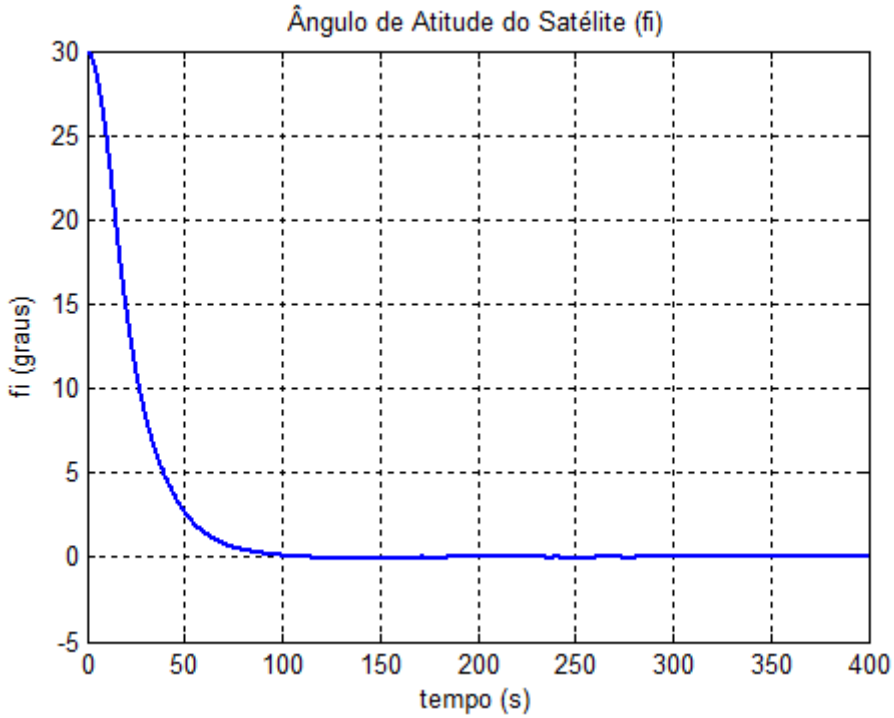


FIGURA 4.23 – Realinhamento do satélite, ângulo ϕ, com parâmetros do controlador conforme Tabela 4.2, e sensor com ruído gaussiano.

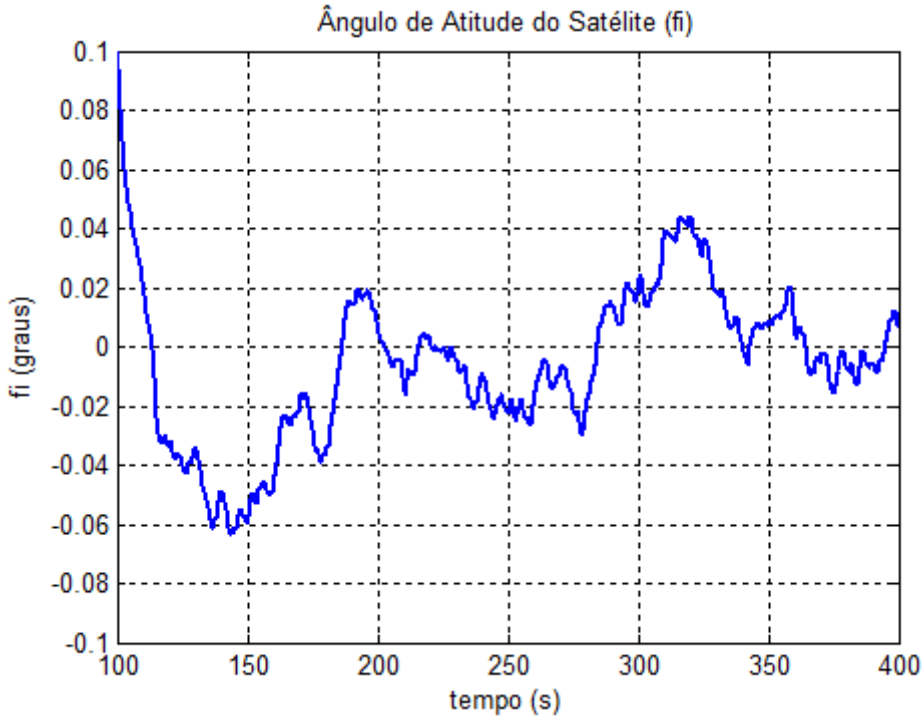


FIGURA 4.24 – Realinhamento do satélite, ângulo ϕ, detalhe da Figura 4.23.

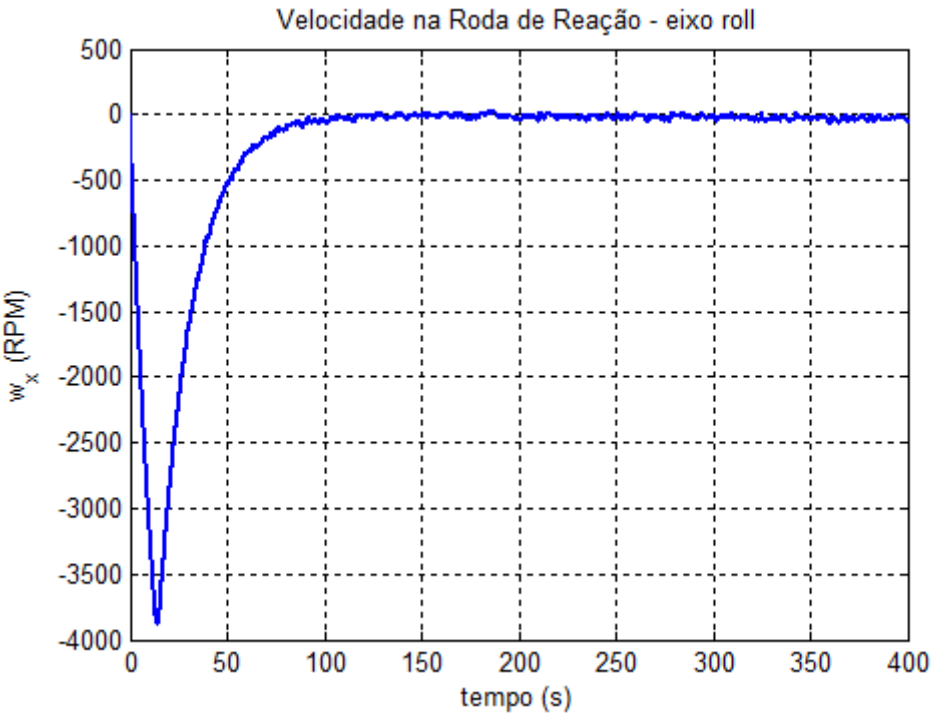


FIGURA 4.25 – Velocidade da roda de reação, eixo X.

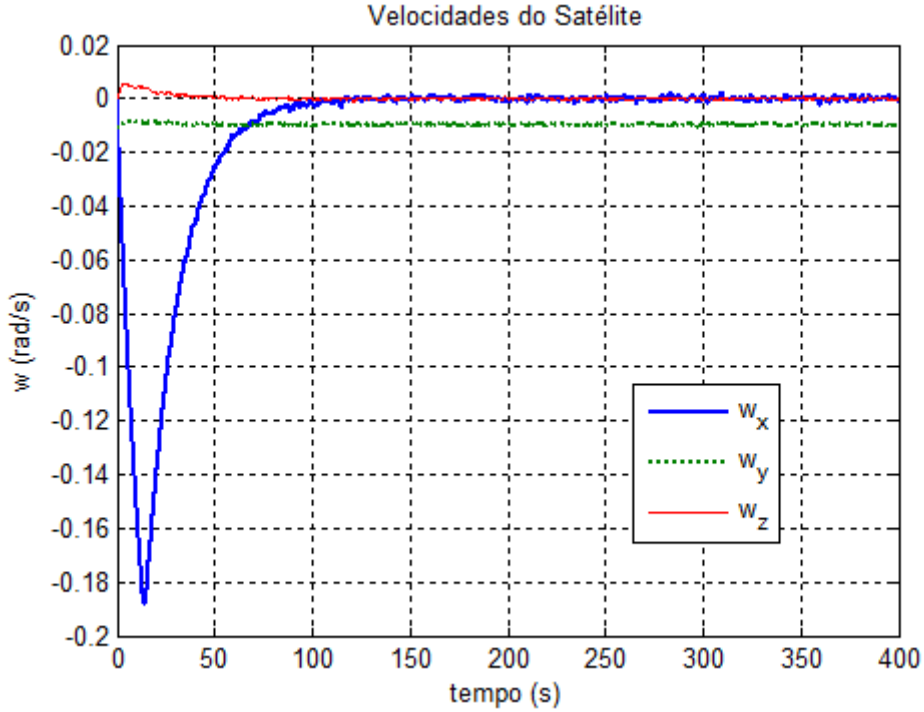


FIGURA 4.26 – Realinhamento do satélites, velocidades do corpo, em rad/s.

Apesar da esperada degradação na resposta da atitude do satélite, ela permaneceu dentro do requerido.

Nas Figuras 4.27 a 4.29, são apresentadas a resposta do sistema a uma manobra bastante pequena, partindo de uma condição desalinhada em apenas 0,07°. Nessa manobra, pode-se perceber melhor a resposta do controlador PID na sua região linear.

Essa condição pode ser percebida na Figura 4.28 onde, a partir do segundo passo de integração, o controlador está fora da condição de saturação. Nessa figura pode-se observar em detalhes o comportamento do controlador em regime de curta duração. Já na Figura 4.29, pode-se ver o mesmo controlador respondendo ao regime de longa duração.

A amplitude dessa manobra pode ser entendida como a faixa de atuação linear do controlador; a partir desse valor, o comportamento do PID precisa ser avaliado junto com o seu mecanismo de saturação.

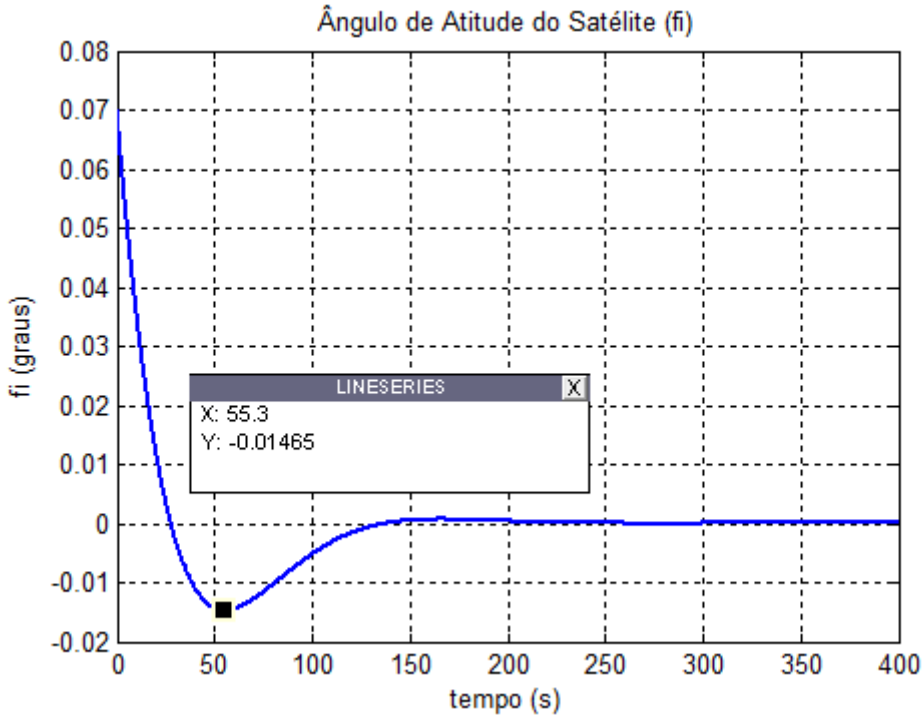


FIGURA 4.27 – Realinhamento da atitude do satélite, eixo de “roll”.

FIGURA 4.28 – Saída do controlador de atitude, eixo de "roll", para uma manobra pequena. Detalhe realçando a resposta de curta duração.

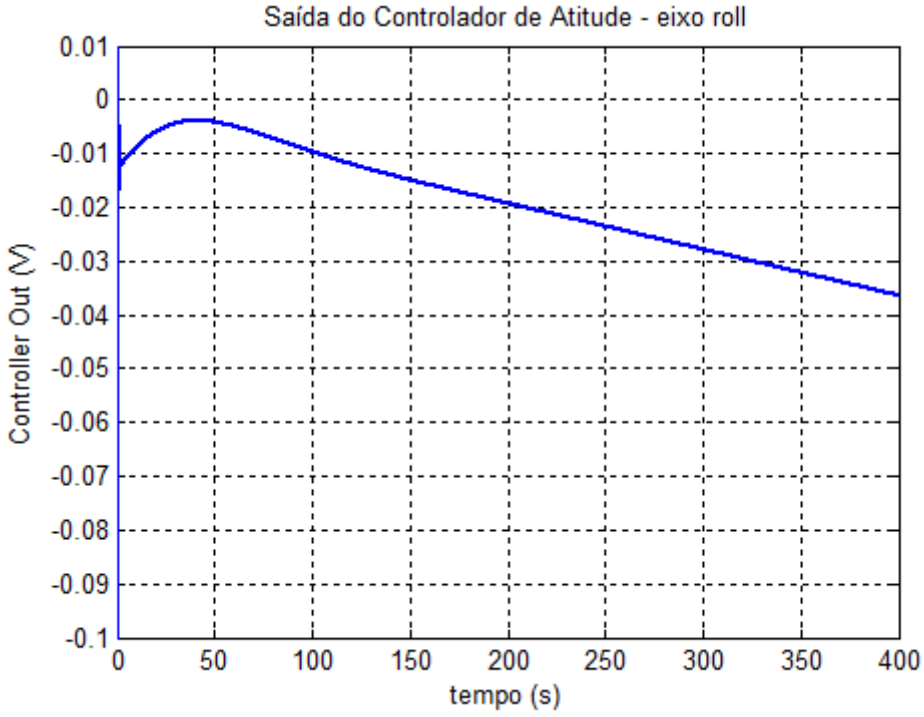


FIGURA 4.29 – Saída do controlador de atitude, eixo de "roll".

CAPÍTULO 5

SIMULAÇÃO EM TEMPO REAL

Neste capítulo descrevem-se os ambientes de co-simulação e, a partir dos resultados do Capítulo 4, o modelo desenvolvido em Simulink é particionado, sendo uma parte migrada para um processador tradicional executando dentro de requisitos de tempo real, e outra parte migrada para um FPGA. Finalmente, o ambiente é integrado e os resultados correspondentes são gerados.

5.1 Ambientes de Simulação Distribuída

Os ambientes de co-simulação utilizados neste trabalho estão diretamente relacionados à disponibilidade de equipamentos baseados em FPGA reconfigurável. Dessa forma, pôde-se adotar 2 abordagens: 1) um equipamento cuja arquitetura é baseada em FPGA possui um módulo processador executando LabVIEW RT; e 2) usando um computador padrão executando Windows XP acrescido de um cartão utilizando um FPGA reconfigurável. Para a primeira abordagem, o ambiente será considerado como baseado em uma arquitetura proprietária; já para a segunda abordagem, o ambiente será considerado como baseado em uma arquitetura padrão.

5.1.1 Ambiente Baseado em Arquitetura Proprietária

Nessa alternativa, o ambiente de co-simulação baseia-se em um equipamento desenvolvido pela National Instruments e chamado de CompactRIO, ou cRIO, considerado pelo fabricante como um Controlador de Automação Programável, e orientado para sistemas de controle embarcado e aquisição de sinais.

Ele é projetado para suportar condições ambientais hostis, tais como temperatura operacional (-40° a 70°C), tensão (até 2300 Vrms), vibração (até 5g), impacto (até 50g).

A sua arquitetura combina um processador embarcado com sistema de tempo real, módulos de entrada e saída que podem ser inseridos e extraídos energizados, e um chassis incorporando um FPGA reconfigurável. O FPGA é o elemento de ligação entre a Entrada/Saída E/S e o processador, e é conectado a este último através de um barramento PCI, permitindo também a personalização na temporização e tratamento de E/S em um nível próximo ao do hardware. A Figura 5.1 mostra como esses elementos se interligam para uma aplicação típica.

Fisicamente, um dos modelos da linha cRIO pode ser visualizado na Figura 5.2. No caso o chassis possui capacidade para 8 módulos de E/S. Construtivamente, o módulo controlador, baseado em um processador Pentium 200MHz, e os módulos de E/S, são fixados a conectores no fundo do chassis do cRIO. Dentro deste chassi está o FPGA reconfigurável, que pode ter duas versões: com capacidade de 1M "gates" ou 3M "gates".

No caso deste trabalho foi utilizada uma configuração de cRIO com chassis para quatro módulo de E/S, um FPGA com capacidade de 3M gates, e o modelo de controlador é cRIO-9004, com 512MBytes de memória não volátil, 64MBytes de RAM e uma porta Ethernet.

As ferramentas de desenvolvimento disponíveis baseiam-se em LabVIEW. Para o processador, LabVIEW RT, e para o FPGA, LabVIEW FPGA. Ambos podem ser considerados como ferramentas adicionais ao LabVIEW, especificamente para disponibilizar recursos de tempo real, no caso do RT; e blocos específicos, caso do FPGA.

Ainda como parte do conjunto de ferramentas disponíveis para esse ambiente, existe um pacote chamado "Simulation Interface Toolkit" – SIT – que simplifica e automatiza a tarefa de embarcar e integrar um modelo definido em Simulink ao conjunto de ferramentas LabVIEW. SIT também instrumenta o modelo, de forma que um usuário

possa avaliar o comportamento do sistema durante a sua execução, em uma interface que não interfere nas características de tempo real do processamento.

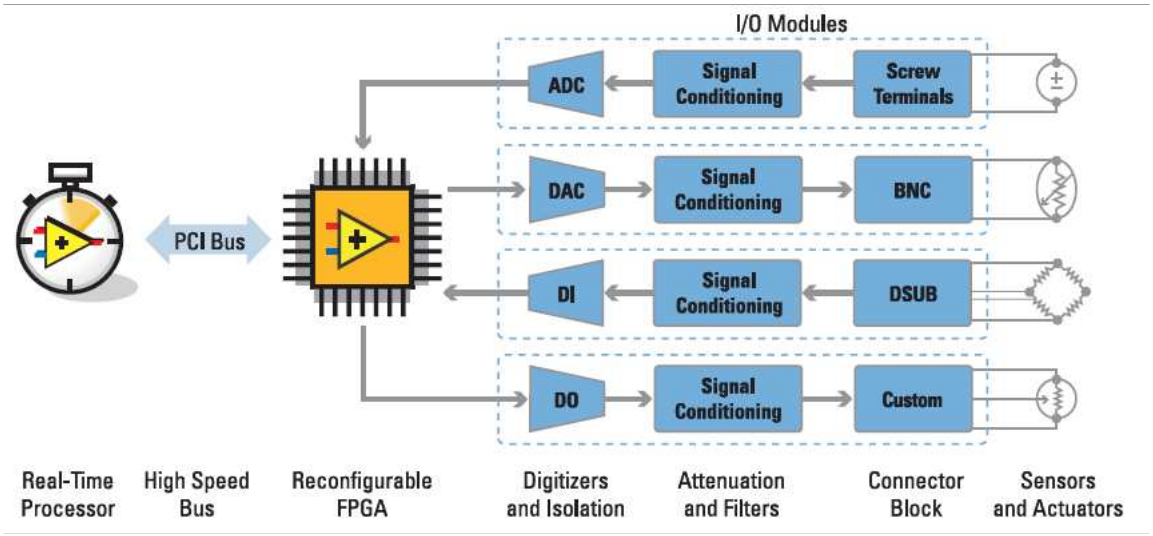


FIGURA 5.1 – Arquitetura do CompactRIO.
FONTE: National Instruments (2009).

FIGURA 5.2 – CompactRIO com capacidade para 8 módulos de E/S.
FONTE: National Instruments (2009).

No processo de criação do ambiente de simulação usando o SIT, uma vez gerada a DLL do modelo, o usuário cria a sua interface e, através de geração automática, é criado o resto dos aplicativos necessários para execução do modelo. Em termos arquiteturais, a estrutura de simulação tem o formato apresentado na Figura 5.3.

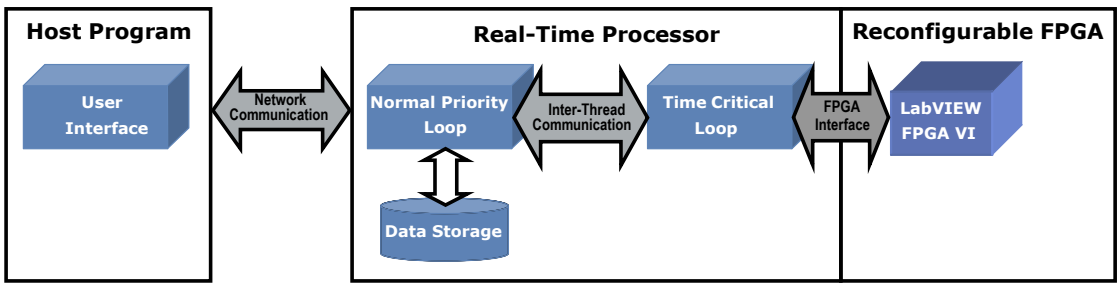


FIGURA 5.3 – Estrutura do ambiente de simulação usando o SIT.

Em termos dos diferentes aplicativos gerados, a Figura 5.4 mostra a interação entre eles, bem como a seqüência das atividades para a execução do modelo dentro do ambiente. Nessa seqüência, o usuário envia o comando para iniciar a execução do modelo [1]. A seguir, a cada passo, a seqüência de atividades [2] a [5] é executada, onde inicialmente são lidos os dados vindos do FPGA. Esses dados são enviados para a área de dados de entrada da DLL, ela é executada, suas saídas são enviadas ao FPGA, e o processo se repete até o final da execução do modelo. Ao final [6], é informado ao usuário que a execução terminou. Implicitamente, a execução da parte do modelo no FPGA está entre o evento [5] e [2].

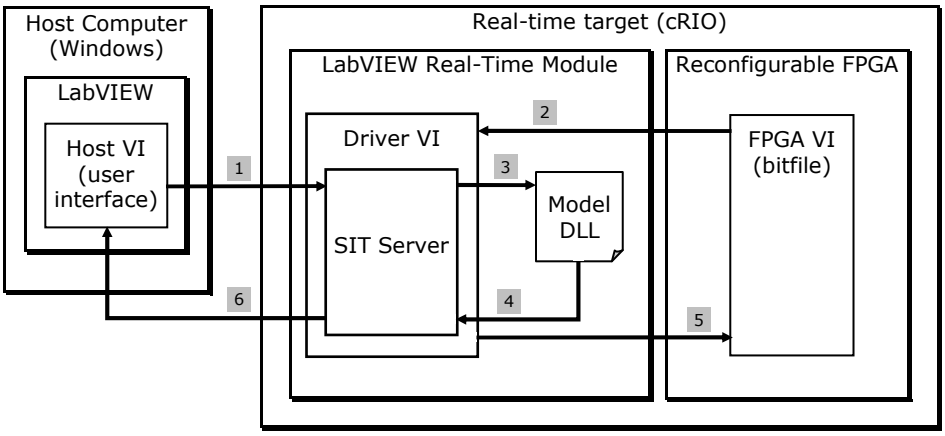


FIGURA 5.4 – Estrutura para execução da co-simulação no cRIO.

5.1.2 Ambiente Baseado em Arquitetura Padrão

Nesta alternativa, o ambiente de co-simulação é baseado em um computador normal, no caso deste trabalho com um processador Athlon XP 1.6 GHz, 640MB RAM e *slots* no padrão PCI para conexão com o cartão reconfigurável.

O cartão com FPGA reconfigurável embarcado corresponde fisicamente ao apresentado na Figura 5.5. Ele foi desenvolvido pela National Instruments, é baseado em um FPGA da Xilinx, Virtex II, na versão com capacidade de 1M gates, possui um oscilador de 40MHz, e também os seguintes elementos de E/S:

- 8 canais de entrada analógica (conversão A/D), 16 bits, com taxa máxima de 200kHz;
- 8 canais de saída analógica (conversão D/A), 16 bits, com taxa máxima de 1MHz;
- 96 linhas de entrada e saída digitais.

FIGURA 5.5 – Cartão PCI-7831, com Xilinx Virtex II.
FONTE: National Instruments (2009).

Para os propósitos deste trabalho não foram utilizados os elementos de E/S, sendo usados apenas o FPGA e seu oscilador.

Com relação ao barramento interno, este cartão possui ainda uma linha de interrupção, permitindo a implementação de mecanismos de sincronização entre o código executado no processador do computador e do FPGA.

A Figura 5.6 mostra a seqüência de atividades para os aplicativos envolvidos na simulação do modelo, no ambiente padrão. A partir do início da execução de Host VI (ativação não mostrada), são realizadas leituras dos dados vindos do FPGA, estes são enviados para a DLL, ela é executada, e suas saídas são enviadas para o FPGA.

Nesse caso, antes da leitura dos dados vindos do FPGA é gerada uma interrupção no processador, e o envio das saídas da DLL ao FPGA é realizada junto com um sinal de reconhecimento da interrupção (ACK) para liberar o FPGA para a execução do próximo passo.

No FPGA, existe um temporizador para garantir a execução no tempo especificado. Na ocorrência do tempo definido, a parte do modelo embarcada é executada usando as entradas recebidas pelo processador e são geradas as saídas. A seguir, uma interrupção é gerada no processador e o FPGA fica esperando pelo sinal de reconhecimento (ACK). Com o ACK recebido, esse passo é considerado terminado, e o processamento do FPGA fica esperando pela próxima ocorrência, definida pelo temporizador.

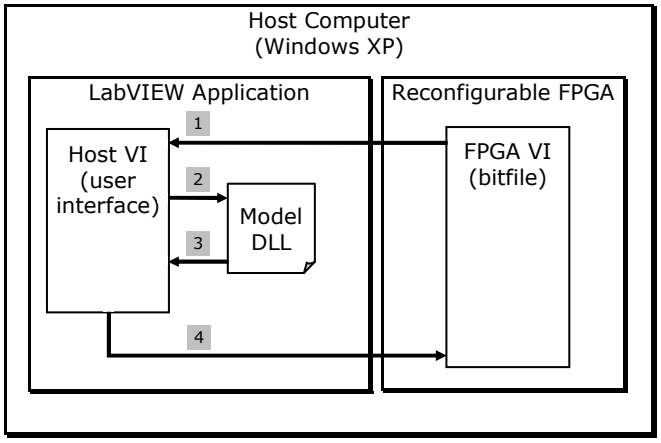


FIGURA 5.6 – Estrutura para execução da co-simulação no PC com FPGA.

5.2 Migração para um Sistema Operacional com Características de Tempo Real

Até o Capítulo 4, todos os resultados obtidos da simulação do modelo foram gerados inteiramente dentro do ambiente Matlab/Simulink, com passo de execução definido como uma grandeza virtual, desacoplada do tempo físico, ou real. Tudo isso foi implementado de uma forma bastante transparente ao usuário.

Agora, deseja-se migrar esse modelo para um ambiente onde o Matlab/Simulink não está presente, e a execução do modelo é baseada em critérios de tempo físico, ou do mundo real.

A parte do modelo que será executada por um FPGA será abordada na Seção 5.3; esta seção refere-se à parte do modelo sendo executada em um processador tradicional, usando algum sistema operacional que forneça as condições mínimas de execução, respeitando os requisitos de tempo.

5.2.1 Migração para o Windows

Conforme já mencionado anteriormente, apesar do Windows não ser considerado um sistema operacional de tempo real, dependendo da especificação de tempo requerida, ele pode ser uma alternativa válida. No caso deste trabalho, a especificação é executar uma simulação durante aproximadamente 400s, utilizando como passo de integração do modelo, bem como o período de amostragem do controlador, o valor de 10ms.

Tomando alguns cuidados no desenvolvimento do aplicativo, é possível executá-lo no Windows e ter o requisito atendido. Para tal, adotou-se uma estratégia que consiste em encapsular o código de uma parte do modelo por um outro código, cujo objetivo será permitir a temporização da execução e integração com os dados vindos da outra parte do modelo (FPGA). A esse código será definida uma prioridade de execução alta, de forma a reduzir a probabilidade de ser interrompido. Os dados gerados só podem ser

visualizados após a execução da simulação. Também foi desabilitada a rede do computador, bem como o seu descanso de tela.

No tocante ao código que encapsula o modelo, um fluxograma simplificado é apresentado na Figura 5.7, ilustrando a estratégia adotada.

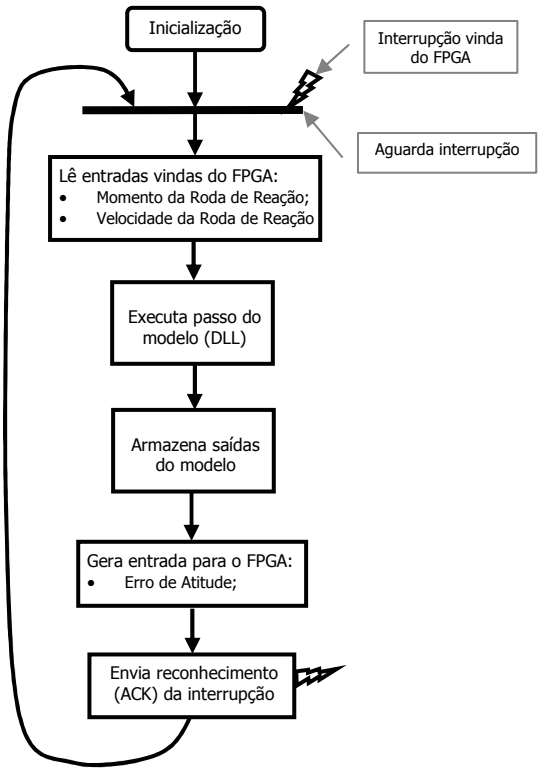


FIGURA 5.7 – Encapsulamento do modelo no SOTR.

Com relação ao código da parte do modelo a ser encapsulada, a seção 5.2.3 a seguir detalha o procedimento.

5.2.2 Migração para o LabVIEW RT

Como no caso do LabVIEW RT o código é gerado automaticamente pelo SIT, com as características já apresentadas na Seção 5.1.1, pouco resta a ser adicionado.

Como informação adicional ao código gerado automaticamente, pequenos ajustes são inseridos em lugares reservados para tal, de forma a implementar a troca de dados

entre o FPGA e a DLL, e vice-versa. Essa personalização foi necessária no caso deste trabalho, em função da forma como os dados no FPGA foram organizados. Dependendo dessa organização, em outras aplicações isso poderia ser resolvido automaticamente pelo SIT.

5.2.3 Geração Automática de Código

Diversas ferramentas estão integradas ao ambiente Matlab/Simulink, em especial o *Real Time Workshop* (RTW), onde o sistema modelado pelo Simulink pode ser convertido automaticamente para linguagem C. Essa ferramenta é parte fundamental no processo de desenvolvimento, especialmente para migração desde um ambiente de simulação em tempo virtual para outro em tempo real, uma vez que o C é uma linguagem de padrão aberto onde, com compiladores apropriados, gera-se aplicativos para o sistema operacional desejado. Para aumentar a flexibilidade na geração de código do RTW, com uso de scripts em uma linguagem própria pode-se adequar o código para um padrão desejado, facilitando o processo de geração para a plataforma desejada.

Para o ambiente baseado na arquitetura padrão, o fluxo de geração de código está apresentado na Figura 5.8. Nesse caso o script foi criado com objetivo de compilar o modelo e gerar uma Biblioteca de Ligação Dinâmica – DLL – facilmente integrável em aplicativos executados na plataforma Windows.

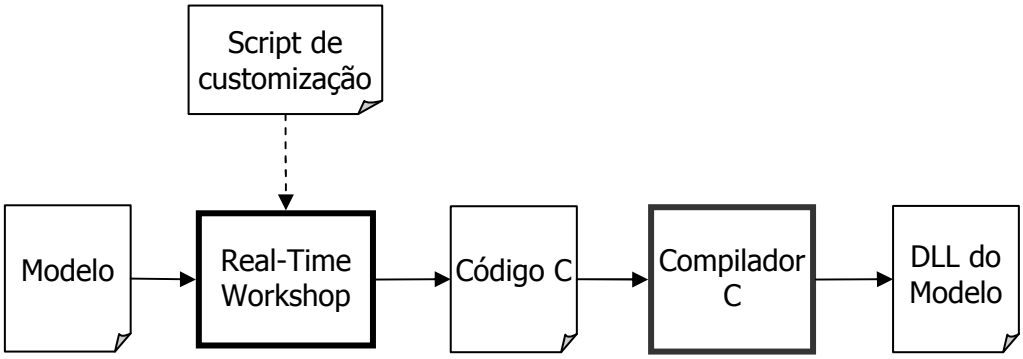


FIGURA 5.8 – Fluxo de geração de código para Windows usando o RTW.

No caso do ambiente usando o LabVIEW RT como sistema operacional, uma parte do pacote SIT é integrado ao Matlab/Simulink. Nesse caso, é necessário incorporar um bloco (SignalProbe, instalado com o SIT) ao modelo, e selecionar como script *nidll.tlc*, para "NI Real-Time Target". A Figura 5.9 mostra o fluxo de geração da DLL do modelo, compatível com a estrutura apresentada na Seção 5.2.2

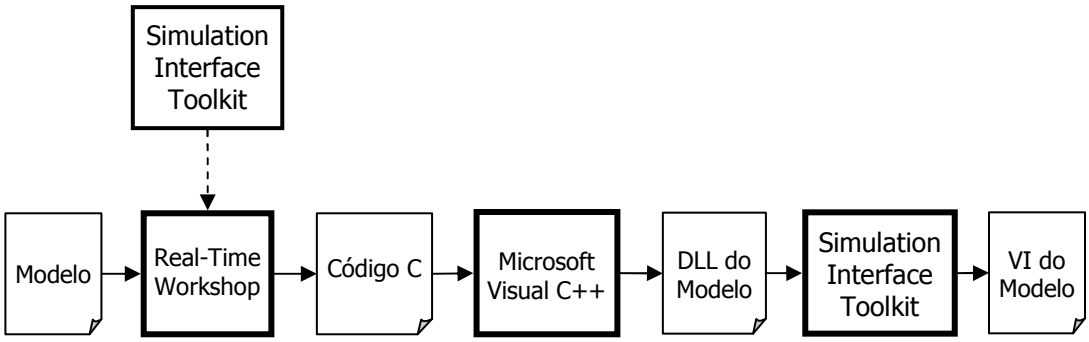


FIGURA 5.9 – Fluxo de geração de código para LabVIEW RT, usando o RTW e o SIT.

5.3 Migração para um FPGA

Usando como referência o modelo apresentado no Capítulo 4, o objetivo será migrar os blocos *PID_digML_* – Figura 4.9 – e *ReactionWheels* – Figura 4.7 –respectivamente modelos do controlador de atitude em um eixo e seu atuador, de forma a serem executados por um FPGA.

Como já mencionado anteriormente, os produtos intermediários dessa migração serão diagramas em LabVIEW mapeando os diagramas Simulink dos modelos correspondentes.

Para o controlador de atitude, observando em maiores detalhes o diagrama Simulink correspondente, nota-se que ele está modelado em um formato bastante próximo de sua implementação digital. Ou seja, reescrevendo os termos P, I e D, chega-se a:

- $P[n] = K_p \cdot Error[n-1]$
- $I[n] = (K_i \cdot T_s/2) \cdot (Error[n] + Error[n-1]) + I[n-1]$
- $D[n] = (K_d / T_s) \cdot (Error[n] - Error[n-1])$

onde: [n] corresponde ao n-ésimo passo de execução para cada um dos termos PID; Error[n] é a diferença entre a referência do controlador e o valor real no n-ésimo passo; K_p, K_i e K_d correspondem aos ganhos do controlador; e T_s é o tempo de amostragem.

Já para o atuador a migração não é tão direta uma vez que, implicitamente, estão implementados integradores em cada uma das funções de transferência existentes no diagrama, e os integradores similares do LabVIEW são bastante simplificados. Portanto, primeiramente foi gerado automaticamente o código em C especificamente para um dos atuadores, usando como referência o diagrama da Figura 5.10; e, a seguir, esse código foi convertido manualmente para o formato LabVIEW para FPGA.

Para essa geração foi utilizado o script *grt.tlc* do RTW, incluído na ferramenta para obtenção de um código embarcável em uma plataforma genérica.

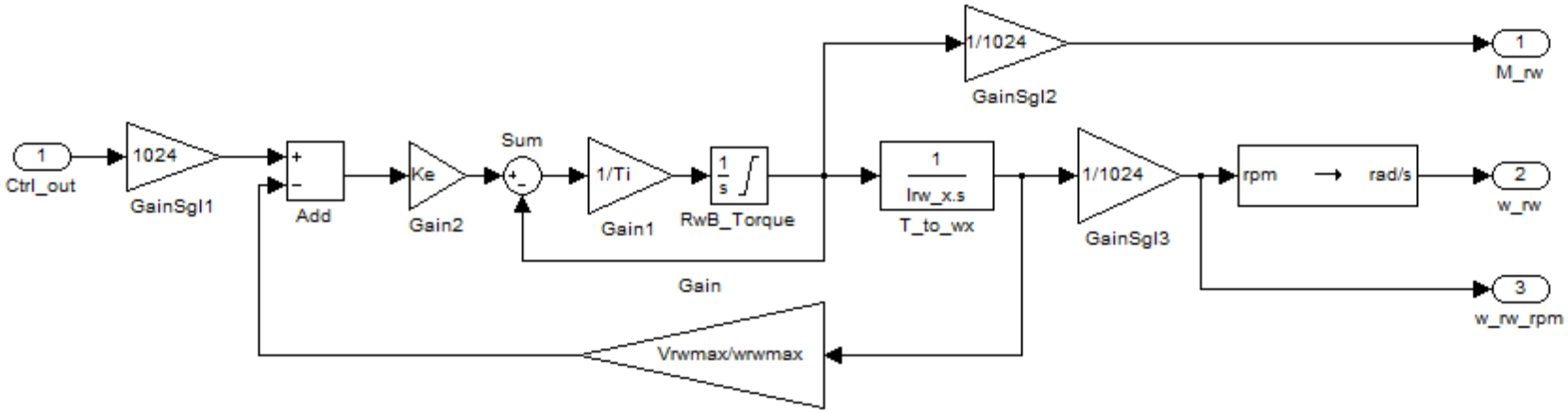


FIGURA 5.10 – Modelo do atuador: detalhe da Figura 4.7 para geração de código.

5.4 Distribuição da Simulação no Ambiente Proprietário

Usando a Figura 5.4 como referência, a parte relativa a Driver VI, gerada automaticamente pelo SIT, está apresentada na Figura 5.11. Na Figura 5.12 pode-se ver a parte principal do conteúdo do bloco 5a, da Figura 5.11 onde é chamada a DLL do modelo (cujo diagrama principal, com a sua interface está na Figura 5.13), usando os valores de torque e velocidade das rodas de reação (*M_rwout* e *w_rwout*, respectivamente, para os eixos x, y e z) vindos do FPGA; e para onde são enviados os erros de posição dos três eixos (*Erro_n_x*, *Erro_n_y* e *Erro_n_z*).

Ou seja, no FPGA são executados o controlador de atitude e seu atuador para os três eixos do satélite. O diagrama principal dessa implementação, a menos da sua inicialização, está na Figura 5.14. O controlador PID está mostrado na Figura 5.15 e o integrador, resolvendo o atuador, está mostrado na Figura 5.16.

5.5 Distribuição da Simulação no Ambiente Padrão

Na definição do Ambiente Padrão, na Seção 5.1.2, pode-se observar que a capacidade do FPGA utilizado é 1/3 daquela existente na Arquitetura Proprietária. Com isso, algumas adaptações tornam-se necessárias, uma vez que os três controladores de atitude e seus atuadores não cabem nessa versão de FPGA com 1M gates.

Nesse caso, apenas o controlador de atitude do eixo de "roll" ficou dentro do FPGA; os outros eixos voltaram para dentro do modelo executado pelo processador tradicional.

As Figuras 5.17 a 5.19 mostram os diagramas principais do ponto de vista da distribuição do modelo entre os componentes da co-simulação.

Tanto o controlador de atitude como o seu atuador possuem a mesma implementação usada no caso da Arquitetura Proprietária, sendo válidas as Figuras 5.15 e 5.16.

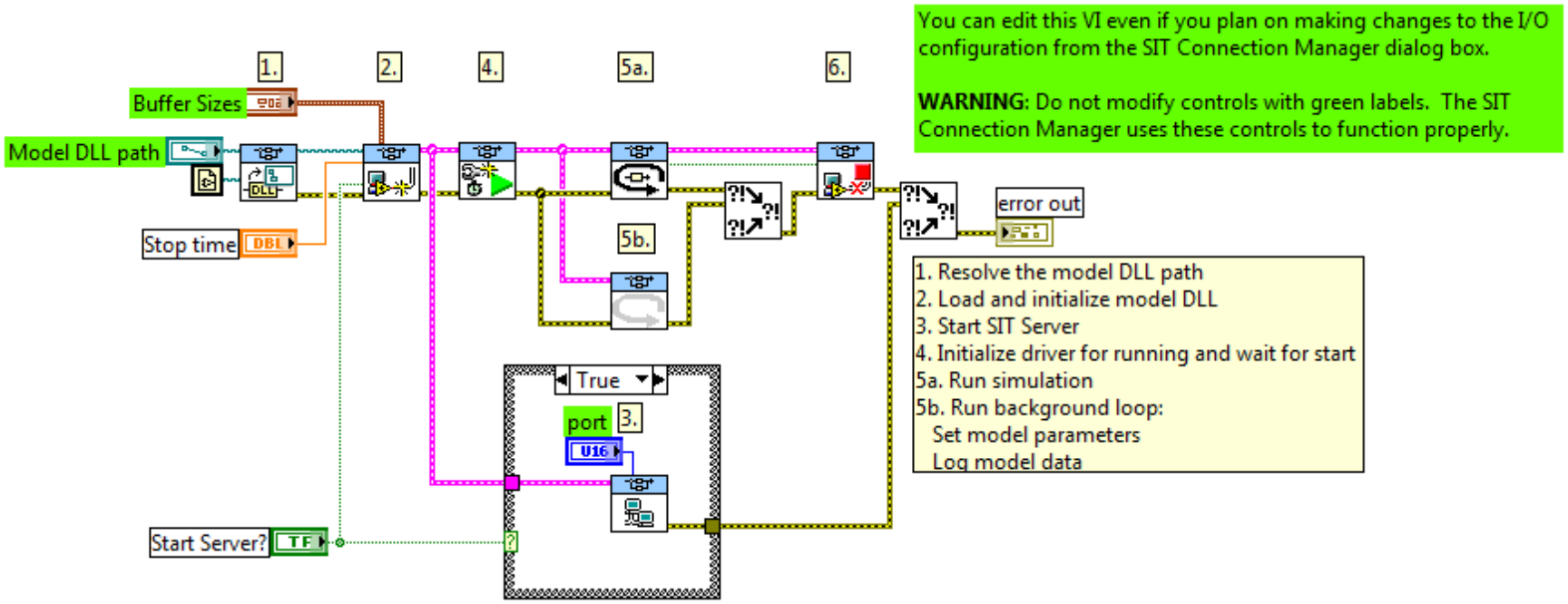


FIGURA 5.11 – Estrutura gerada automaticamente pelo SIT para execução em LabVIEW RT, na arquitetura proprietária.

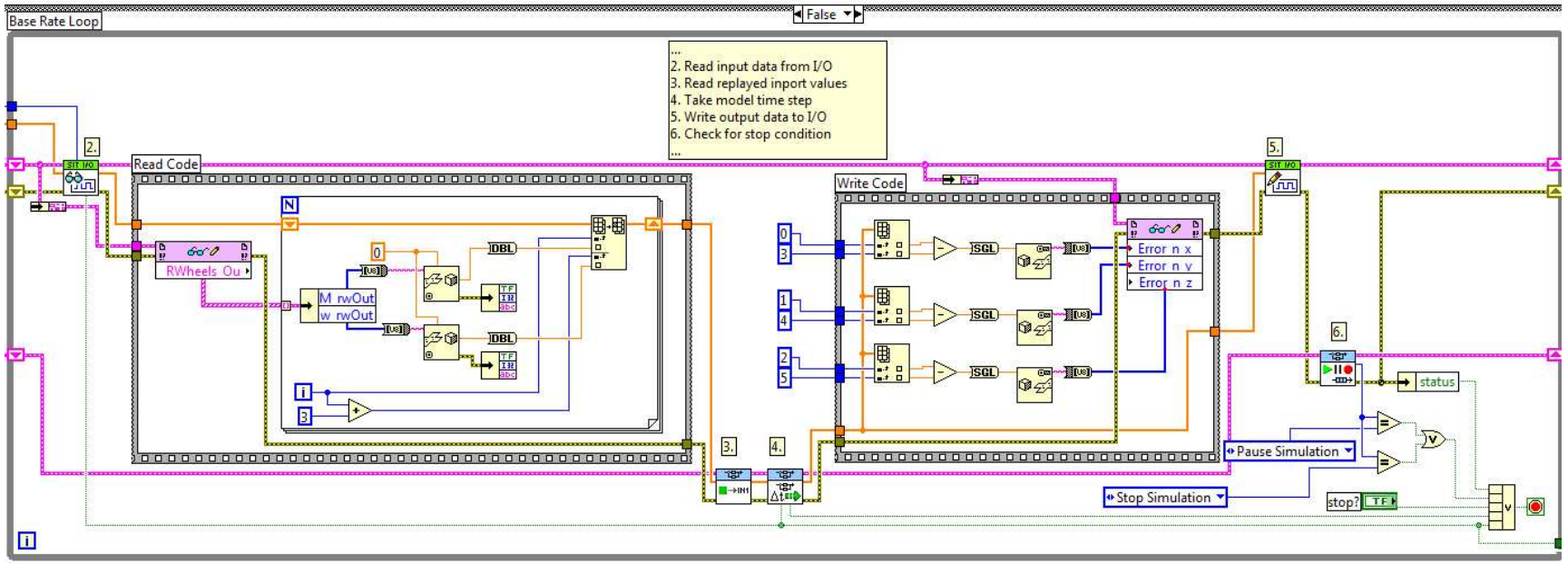


FIGURA 5.12 – Detalhe do bloco 5a, Figura 5.11, para execução do modelo e sincronismo de dados com FPGA. Parcialmente gerado automaticamente por SIT.

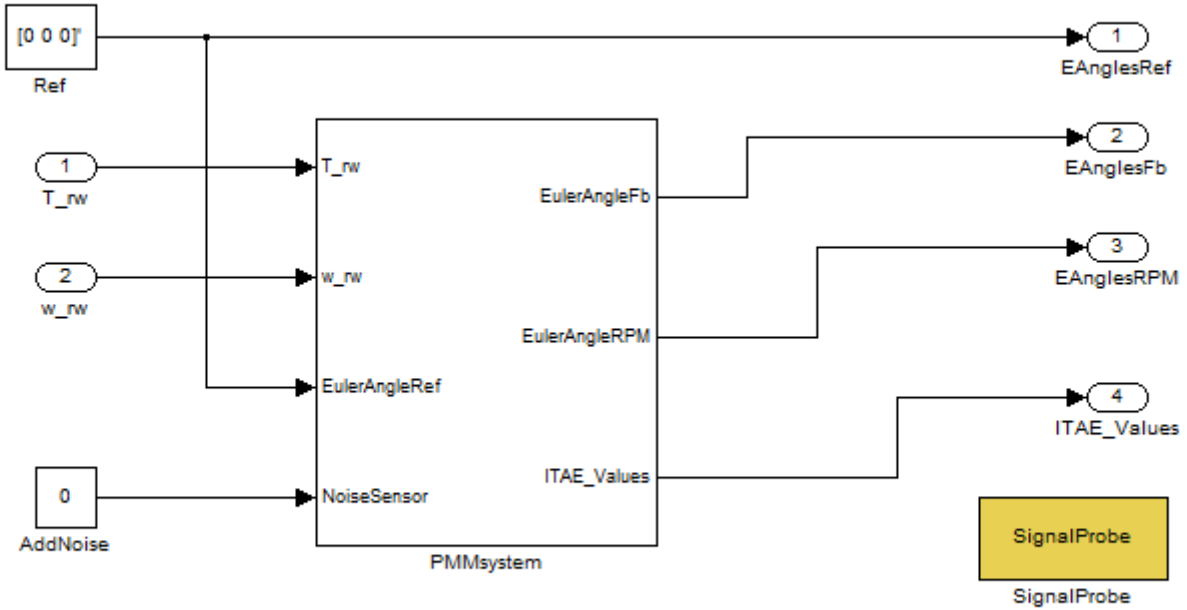


FIGURA 5.13 – Diagrama principal no Simulink do modelo integrado pelo SIT para ser executado em LabVIEW RT (dentro do bloco 4, Figura 5.12), na arquitetura proprietária.

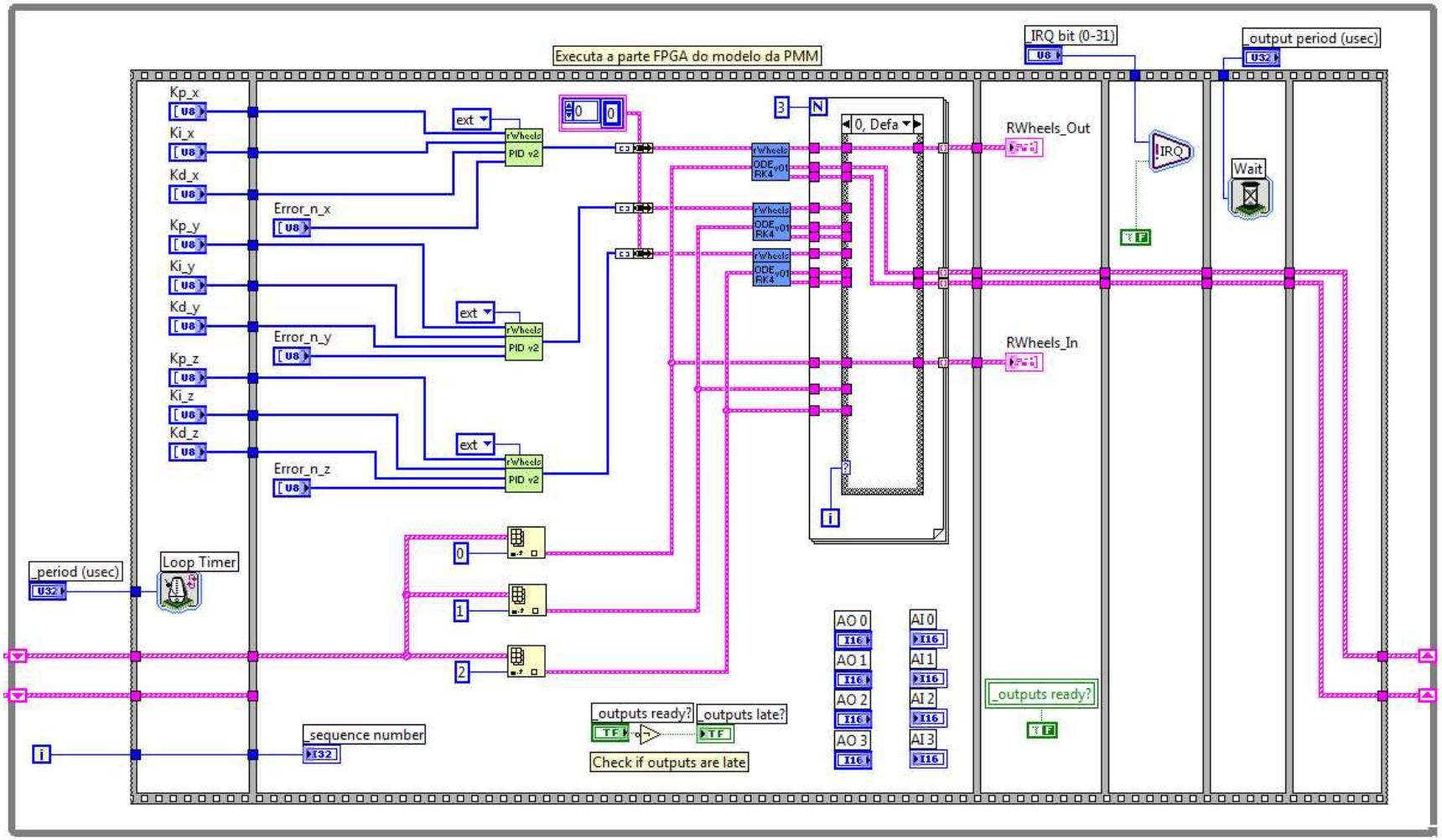


FIGURA 5.14 – Diagrama principal da parte FPGA do sistema, arquitetura proprietária: controladores de atitude e seus atuadores.

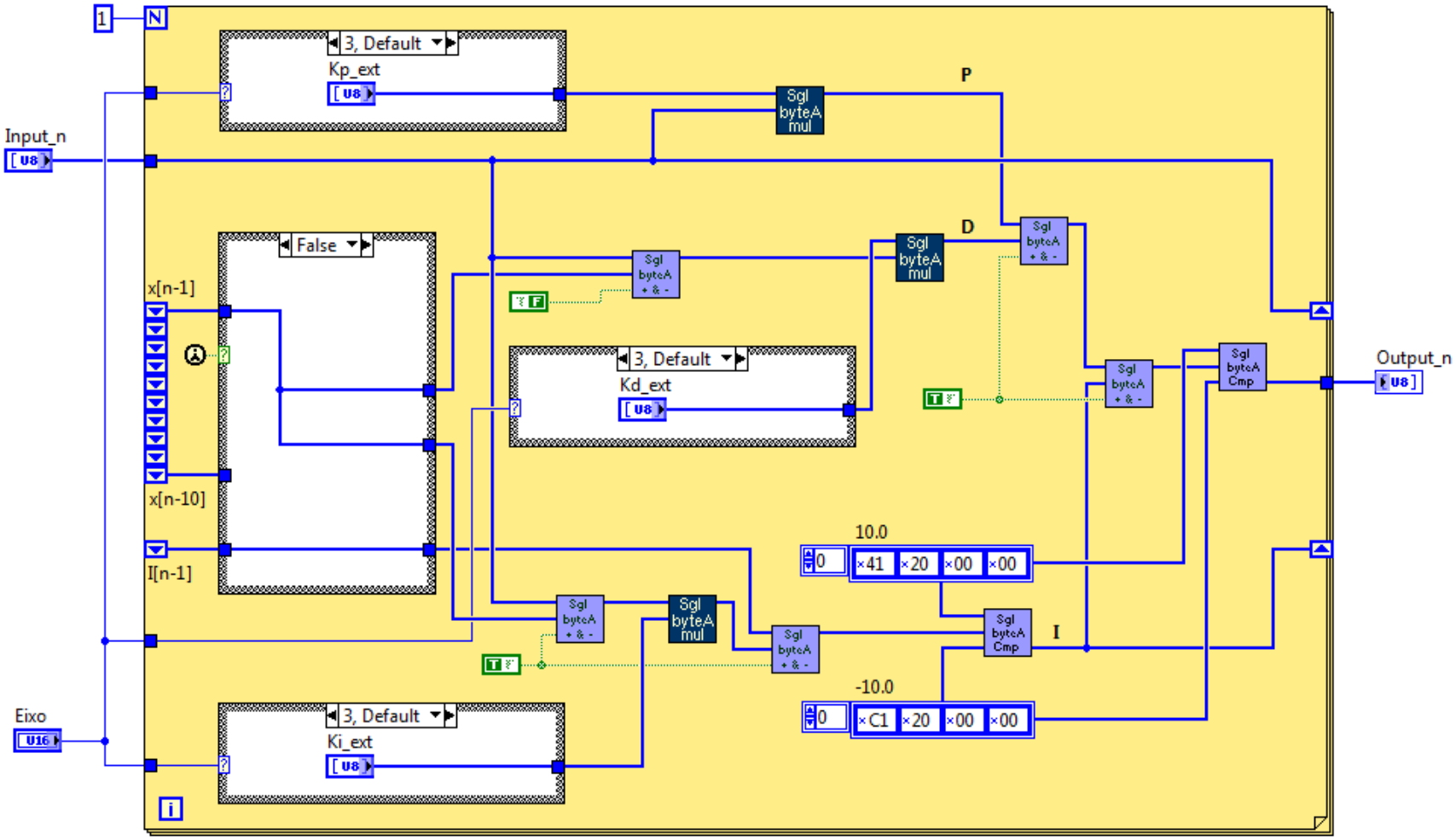


FIGURA 5.15 – Controlador de atitude (PID) executado em FPGA.

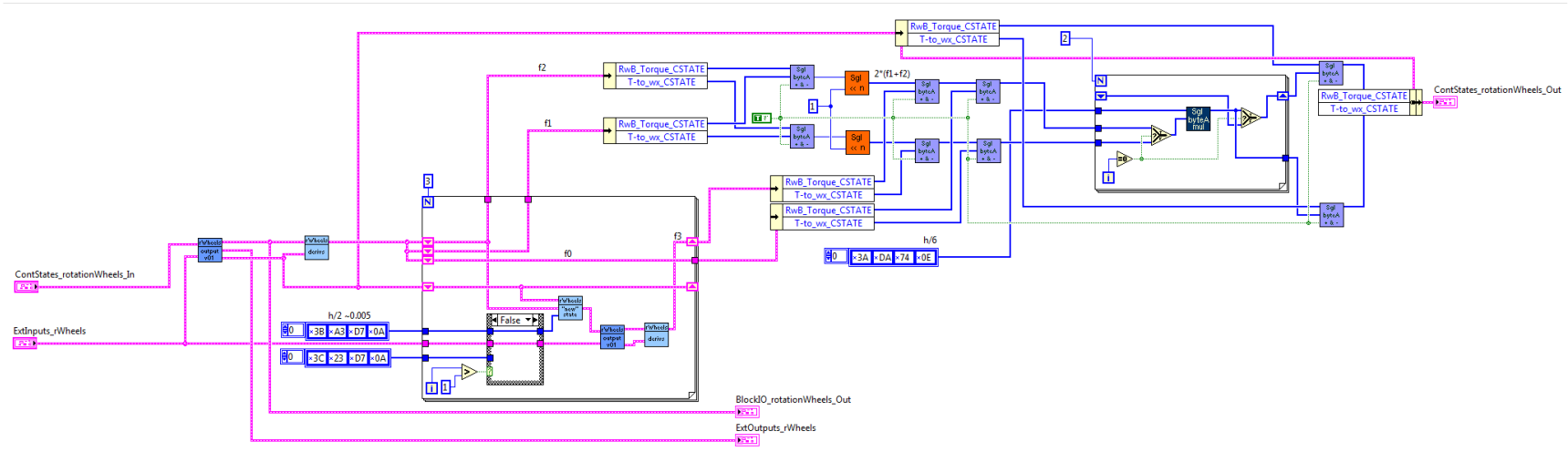


FIGURA 5.16 – Modelo da roda de reação, executado em FPGA, implementado por integrador Runge-Kutta de 4ª ordem.

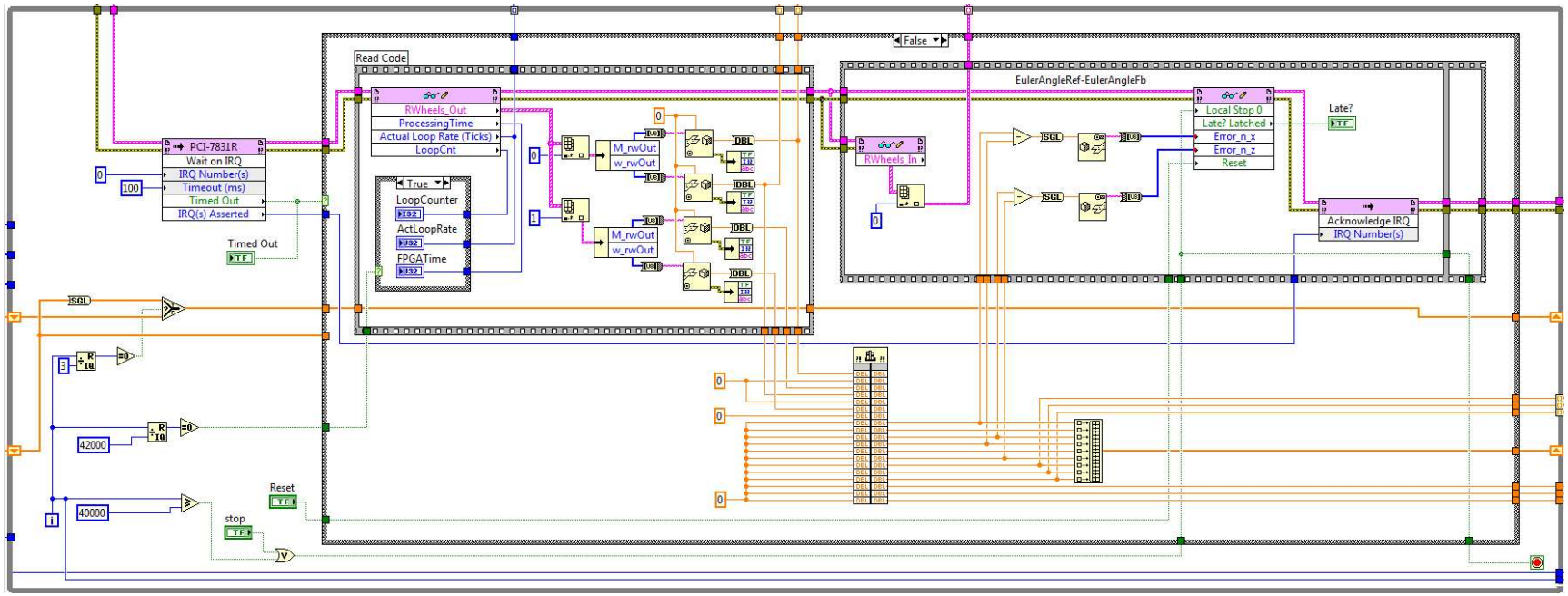


FIGURA 5.17 – Diagrama principal, arquitetura padrão.

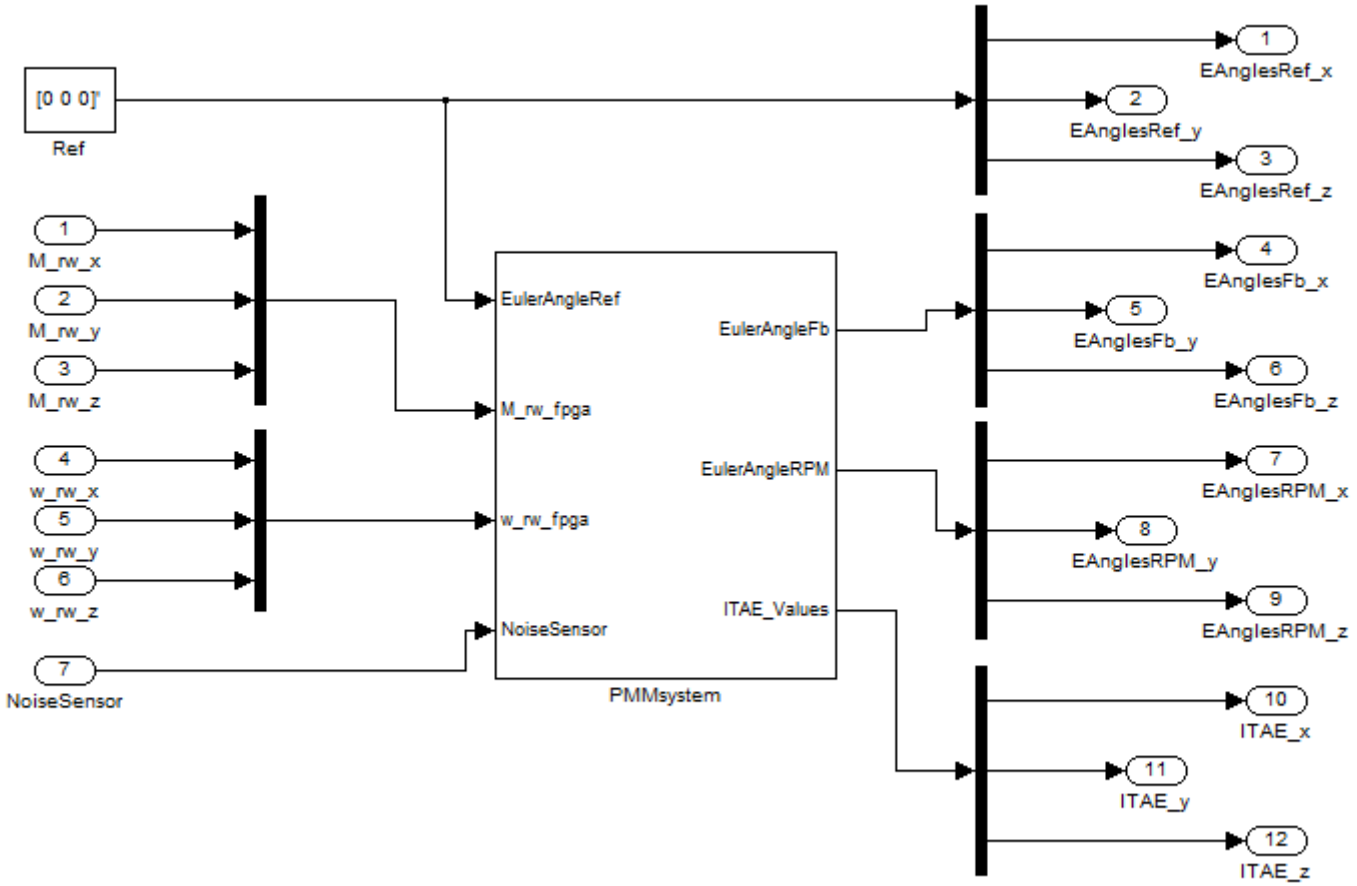


FIGURA 5.18 – Diagrama de topo Simulink, para a arquitetura padrão.

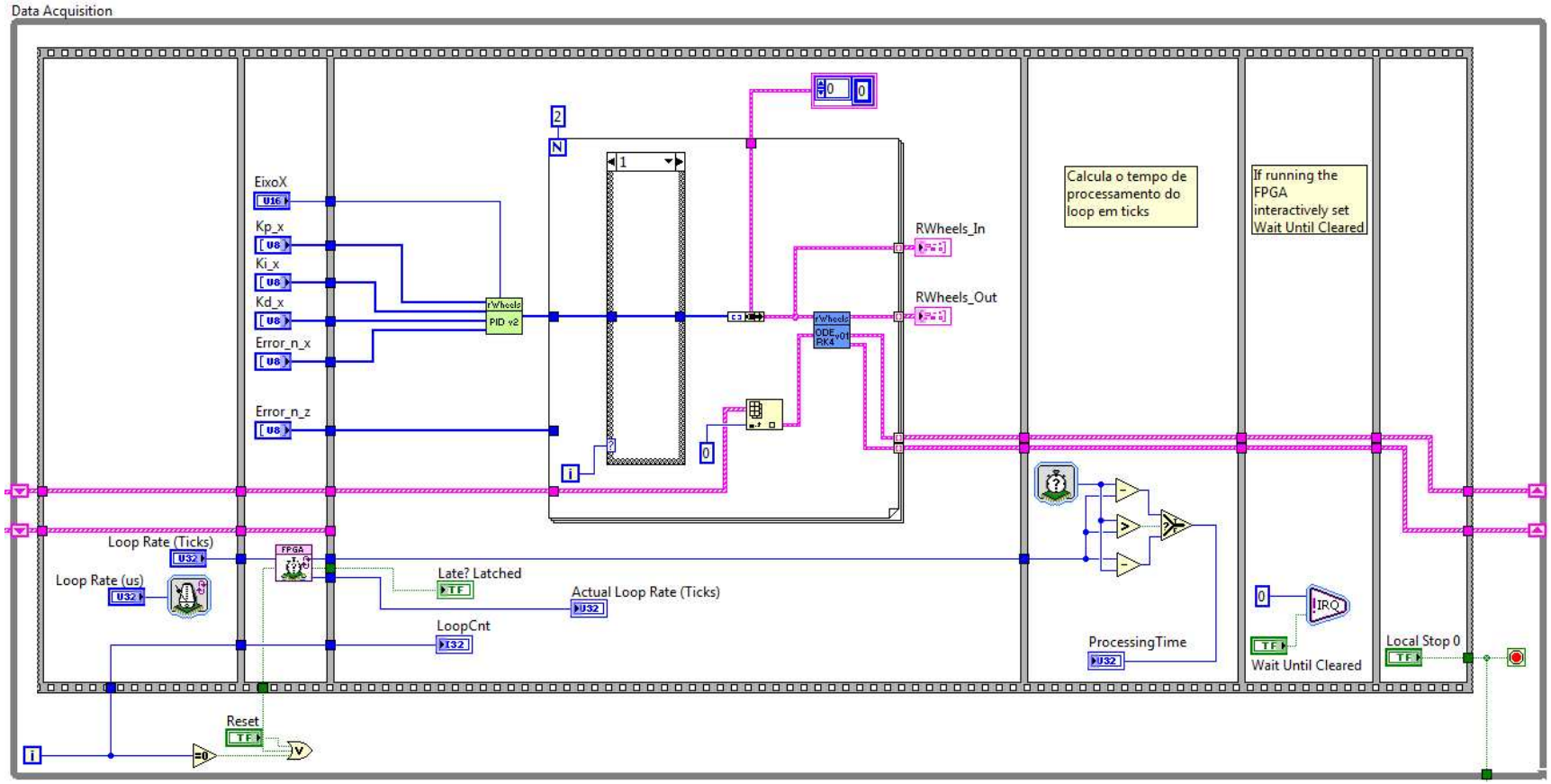


FIGURA 5.19 – Diagrama principal da parte FPGA do sistema, na arquitetura padrão.

5.6 Resultados das Simulações Distribuídas

Os resultados obtidos, a partir dos dois ambientes de co-simulação são apresentados nos itens a seguir. Com o objetivo de facilitar a comparação com a resposta gerada na etapa de projeto, no Capítulo 4, procurou-se reproduzir as mesmas manobras utilizadas anteriormente.

Deve-se atentar para o fato de que foi utilizado o Ambiente Padrão para correção de erros e refinamento dos ganhos, em função da sua disponibilidade e maior rapidez na geração de resultados – foi possível executar o sistema com uma taxa de 1ms – reduzindo o tempo da simulação em 6 minutos (de 400 para 40s), durante a etapa de depuração do código.

Nessas condições, o Ambiente Proprietário foi utilizado apenas como um segundo verificador do processo de desenvolvimento.

5.6.1 Resultados no Ambiente Padrão

Usando os ganhos obtidos no Capítulo 4, Tabela 4.2, e partindo de um desalinhamento de 30° no eixo de rolagem, procura-se realinhar o satélite com relação ao seu referencial VLHL. No que se refere ao ângulo do eixo de rolagem (ϕ), as Figuras 5.20 e 5.21 mostram resultados praticamente idênticos aos obtidos anteriormente (Figuras 4.15 e 4.16) durante a simulação em ambiente virtual.

Já na Figura 5.22, e em maior detalhe na Figura 5.23, pode-se observar uma ligeira saturação na roda de reação, durante o seu processo de desaceleração. Para eliminar esse efeito, foi feito um pequeno ajuste no ganho proporcional do controlador, cujos valores finais de ganho são apresentados na Tabela 5.1. Os novos índices ITAE, calculados da mesma forma que em 4.3.1 (Tabela 4.3) estão na Tabela 5.2.

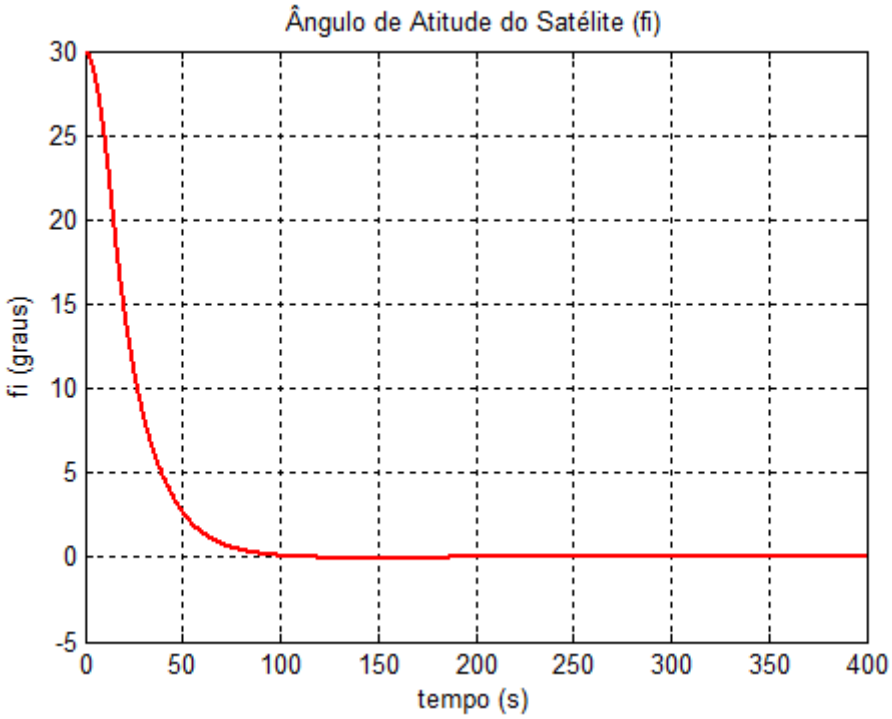


FIGURA 5.20 – Realinhamento do satélite, ângulo ϕ, com PID executado em FPGA e ganhos conforme a Tabela 4.2.

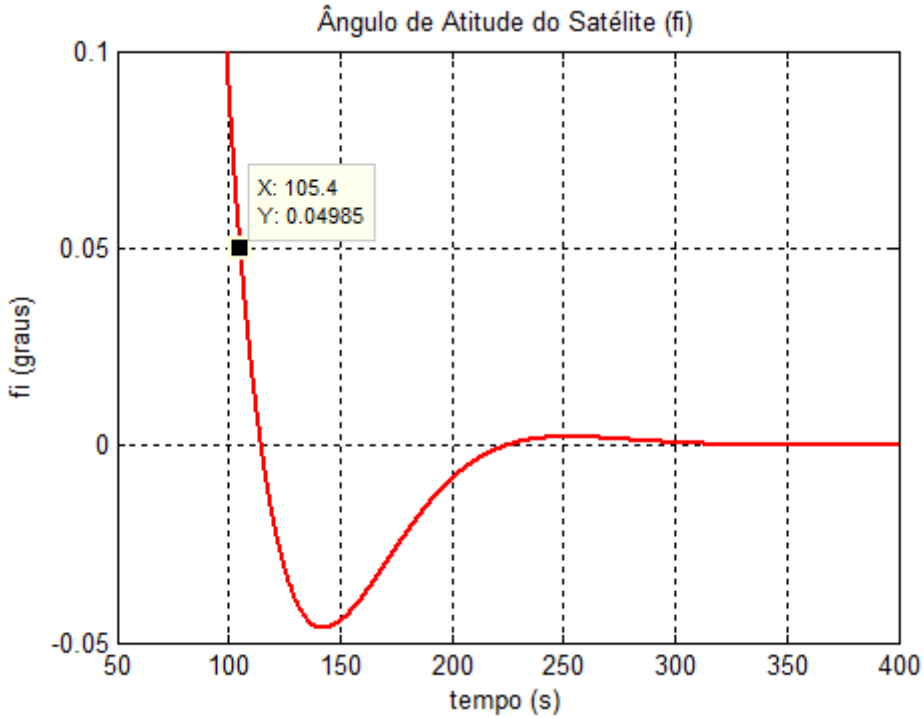


FIGURA 5.21 – Realinhamento do satélite, ângulo ϕ, detalhe da Figura 5.20.

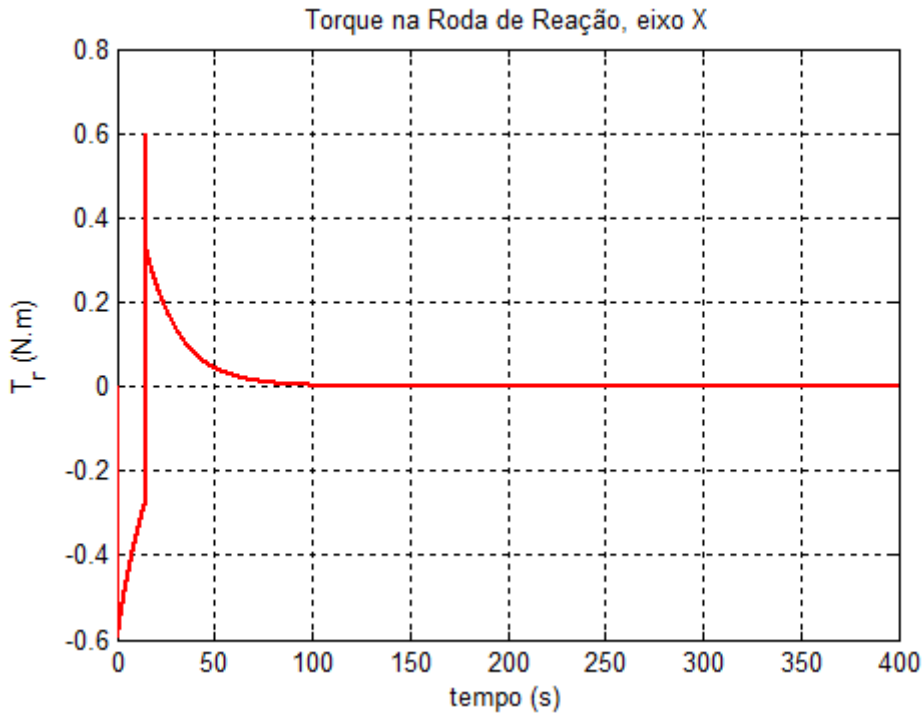


FIGURA 5.22 – Torque na roda de reação, eixo X, com modelo executado em FPGA.

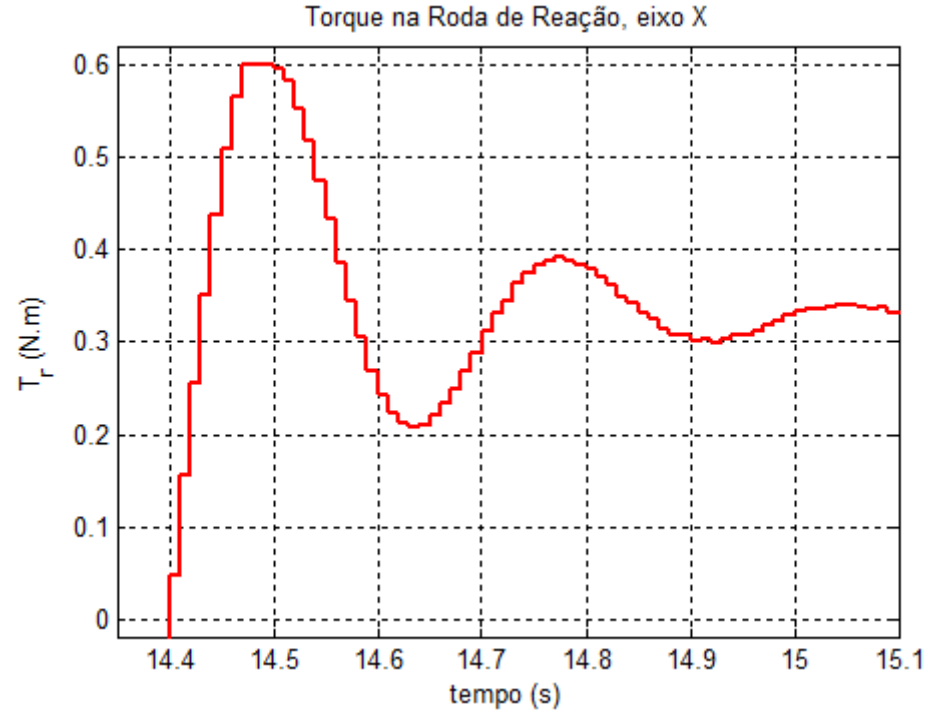


FIGURA 5.23 – Torque na roda de reação, eixo X, detalhe da Figura 5.22.

TABELA 5.1 – Ganhos PID após execução usando FPGA.

ParâmetroPID / *ângulo*	K_p	K_i	K_d
ϕ	K_{p_x} = 7400	K_{i_x} = 226,36	K_{d_x} = 140960
θ	K_{p_y} = 12546	K_{i_y} = 383,72	K_{d_y} = 238990
ψ	K_{p_z} = 9129	K_{i_z} = 279,2	K_{d_z} = 173900

TABELA 5.2 – Índice ITAE do sistema em Simulink, com PID conforme Tabela 5.1.

	ϕ	**θ**	**ψ**
ITAE	270.5	0.02432	0.9097

Assim, as Figuras 5.24 a 5.33 mostram a resposta do sistema com os controladores adotando os ganhos da Tabela 5.1. Da mesma forma que no Capítulo 4, as Figuras 5.24 a 5.26 apresentam o comportamento da atitude do satélite no eixo de rolagem, através do ângulo ϕ, com as condições iniciais (30°, 0°, 0°). Na Figura 5.26, está realçado o momento (111,4s) a partir do qual é atendido o requisito de precisão de apontamento de ±0,05° em menos de 180s.

Ainda na Figura 5.26, deve-se notar que a medida foi feita no sinal obtido pelo sistema co-simulado. A linha próxima refere-se ao mesmo sinal, executado no ambiente utilizado no Capítulo 4, mas com os ganhos da Tabela 5.1. De fato, nas três figuras a resposta do sistema sendo executado em tempo real com FPGA e em tempo virtual no Simulink estão sobrepostas, mas essa respostas estão tão próximas que só na Figura 5.26 é possível identificar que tratam-se de duas formas de onda.

Com isso, os resultados apresentados nas Figuras 5.25 a 5.28 comprovam o atendimento às especificações, bem como a semelhança das respostas onde se utiliza o FPGA, e onde se utiliza a sua simulação em tempo virtual.

O torque na roda de reação do eixo de "roll" pode ser visto nas Figuras 5.27 e 5.28, nas mesmas condições que para o ângulo ϕ de atitude. Também nessas figuras, apenas para pequenas resoluções de tempo é possível perceber que existem duas formas de onda no mesmo gráfico (em tempo real com FPGA e em tempo virtual com Simulink). Como comentário final, na Figura 5.28 está realçado o valor máximo de torque (0.587N.m em 13,8s) durante a desaceleração da roda de reação, mostrando que ela não chegou a saturar durante essa inversão.

De forma similar ao Capítulo 4, as Figuras 5.29 a 5.31 mostram outras variáveis de interesse. As Figuras 5.32 e 5.33 mostram a resposta do sistema quando o sensor de atitude está sujeito a ruído gaussiano, com as mesmas características do Capítulo 4.

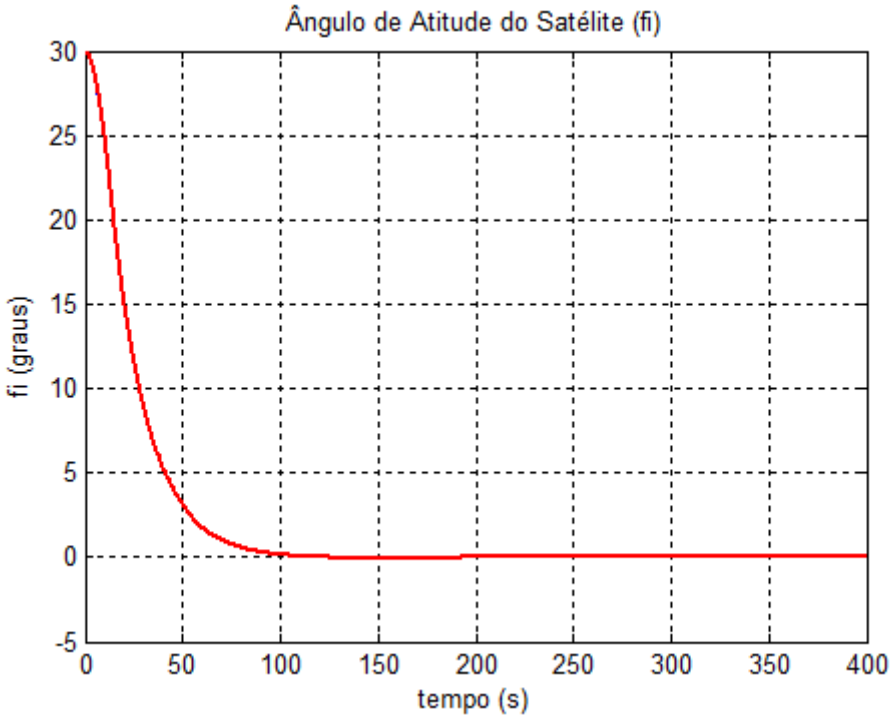


FIGURA 5.24 – Realinhamento do satélite, ângulo ϕ, com PID executado em FPGA e ganhos conforme Tabela 5.1.

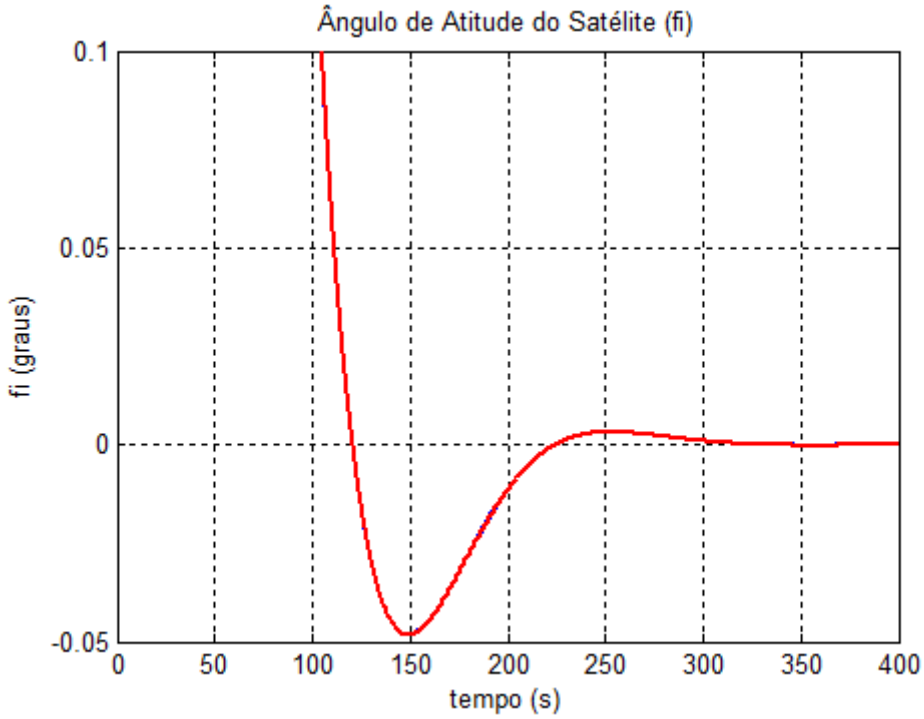


FIGURA 5.25 – Realinhamento do satélite, ângulo ϕ, detalhe da Figura 5.24.

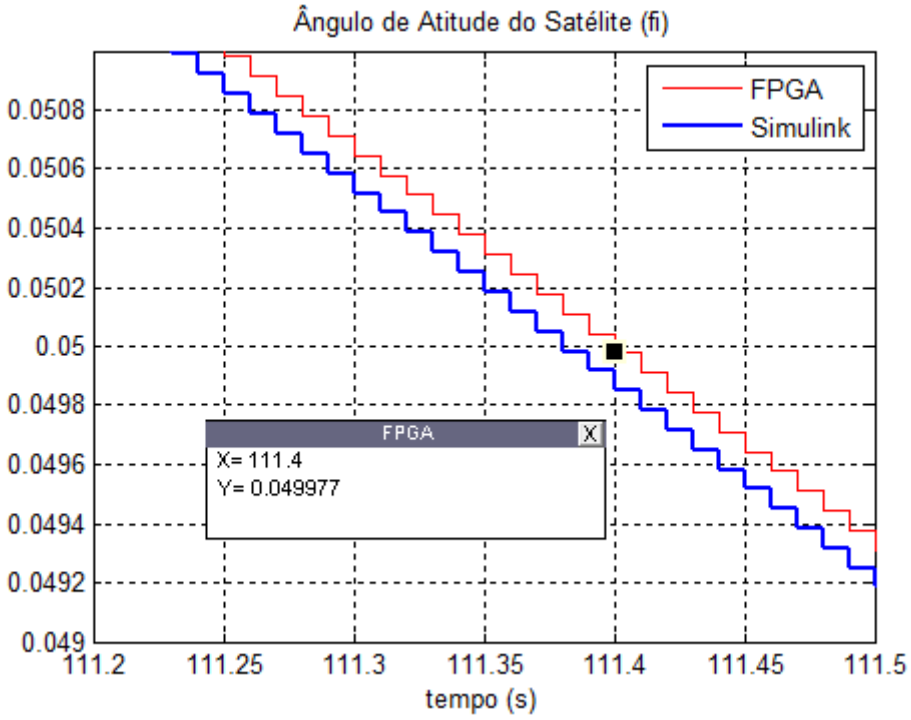


FIGURA 5.26 – Realinhamento do satélite, ângulo ϕ, detalhe da Figura 5.25.

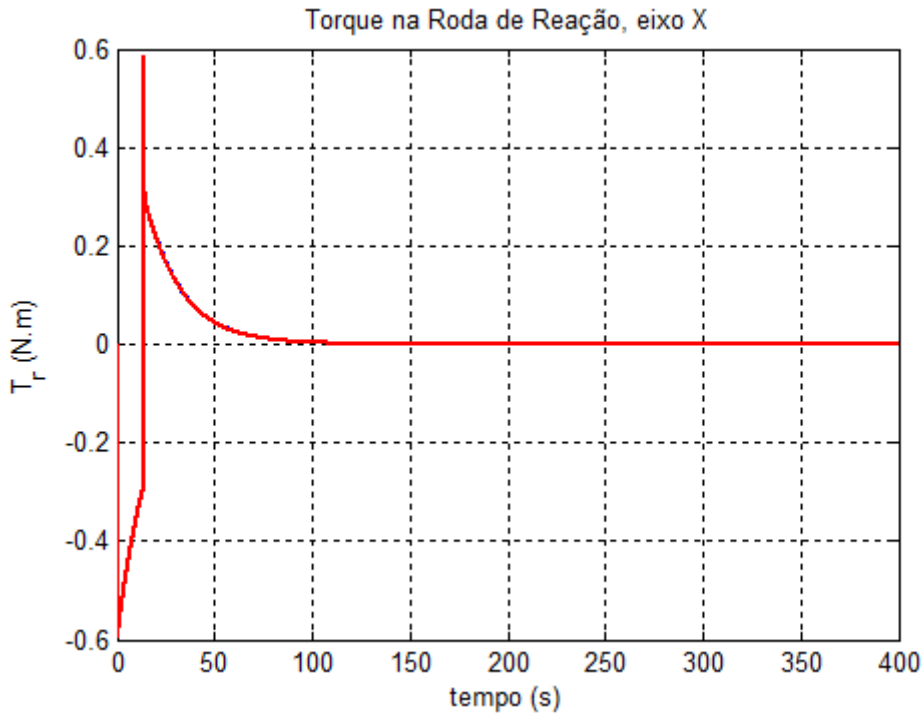


FIGURA 5.27 – Torque na roda de reação, eixo X, com modelo executado em FPGA e usando os ganhos da Tabela 5.1.

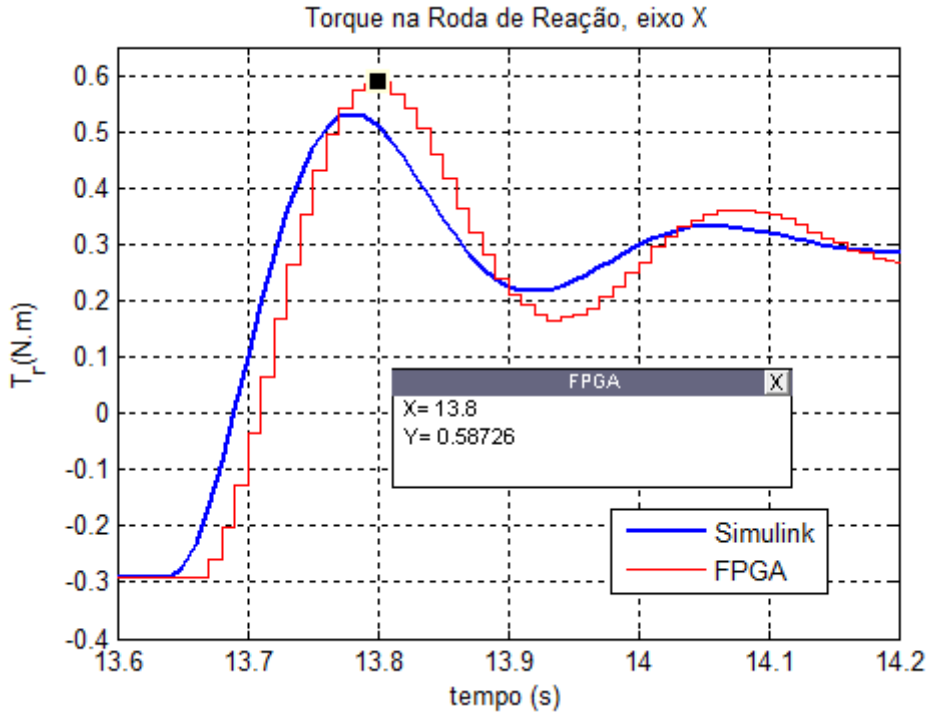


FIGURA 5.28 – Torque na roda de reação, eixo X, detalhe da Figura 5.27.

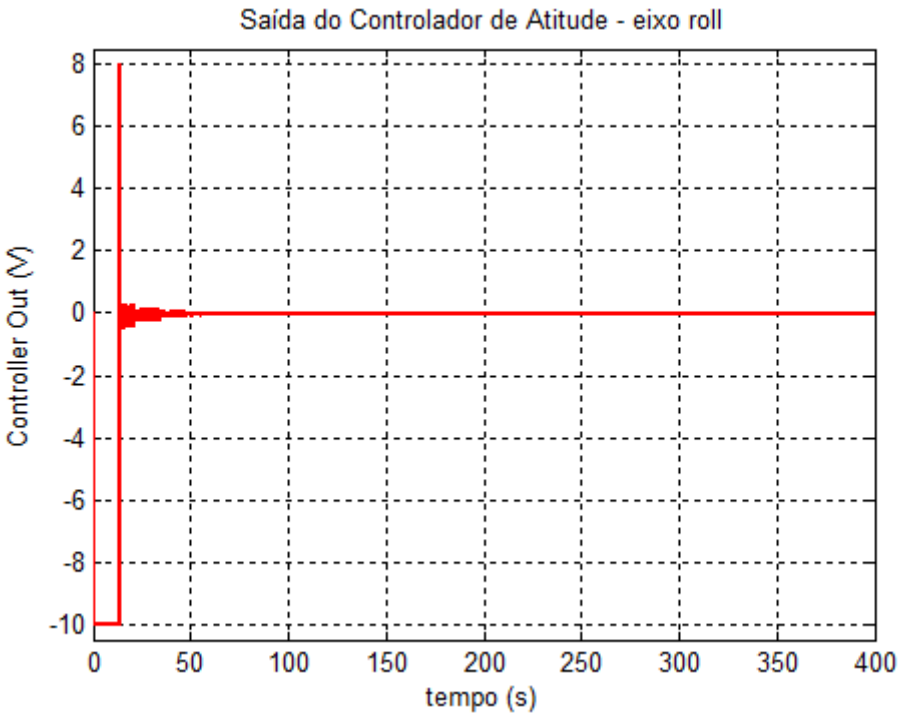


FIGURA 5.29 – Saída do controlador de atitude, eixo ϕ.

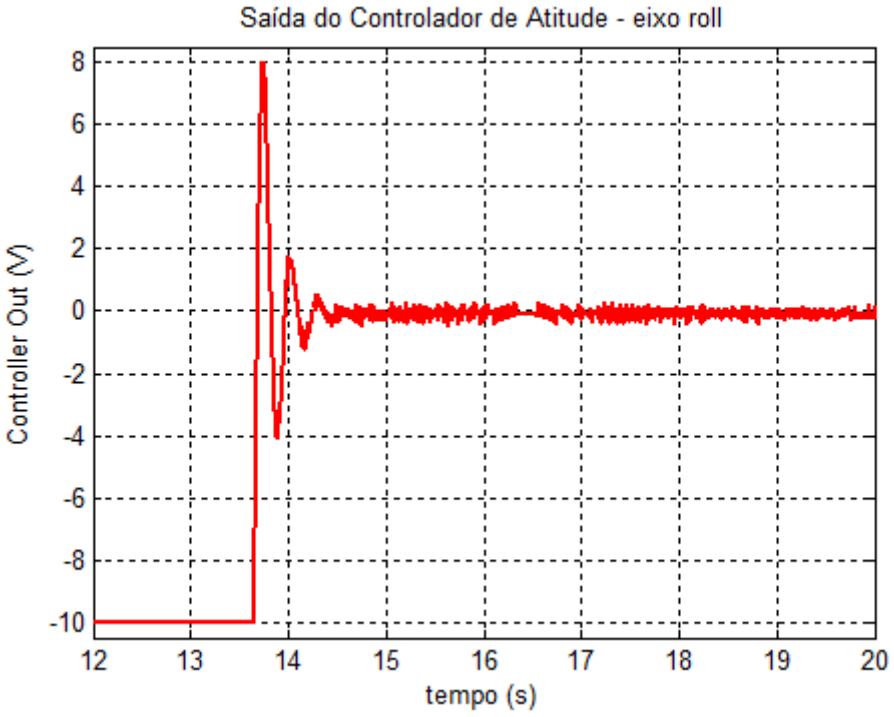


FIGURA 5.30 – Saída do controlador de atitude, detalhe da Figura 5.29.

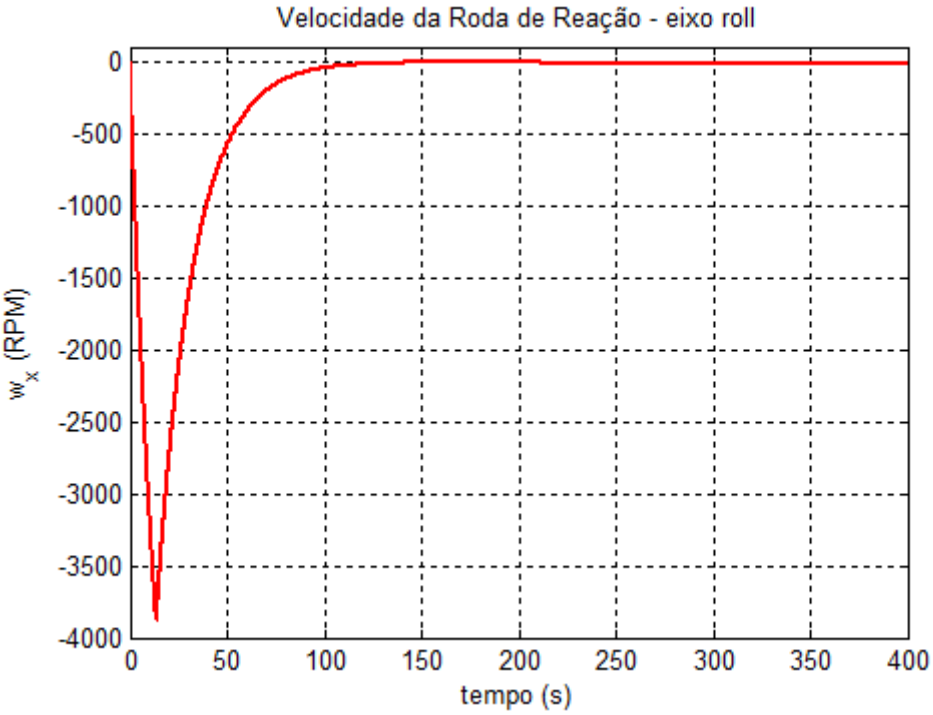


FIGURA 5.31 – Velocidade da roda de reação, eixo X.

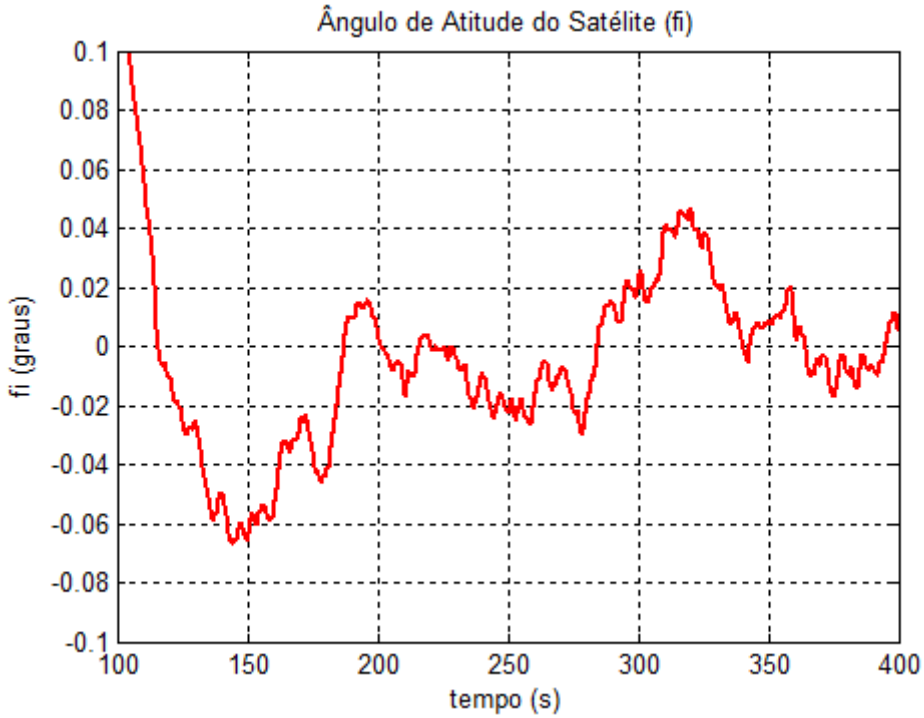


FIGURA 5.32 – Detalhe do realinhamento do satélite, ângulo ϕ, parâmetros do controlador conforme a Tabela 5.1, sensor com ruído gaussiano.

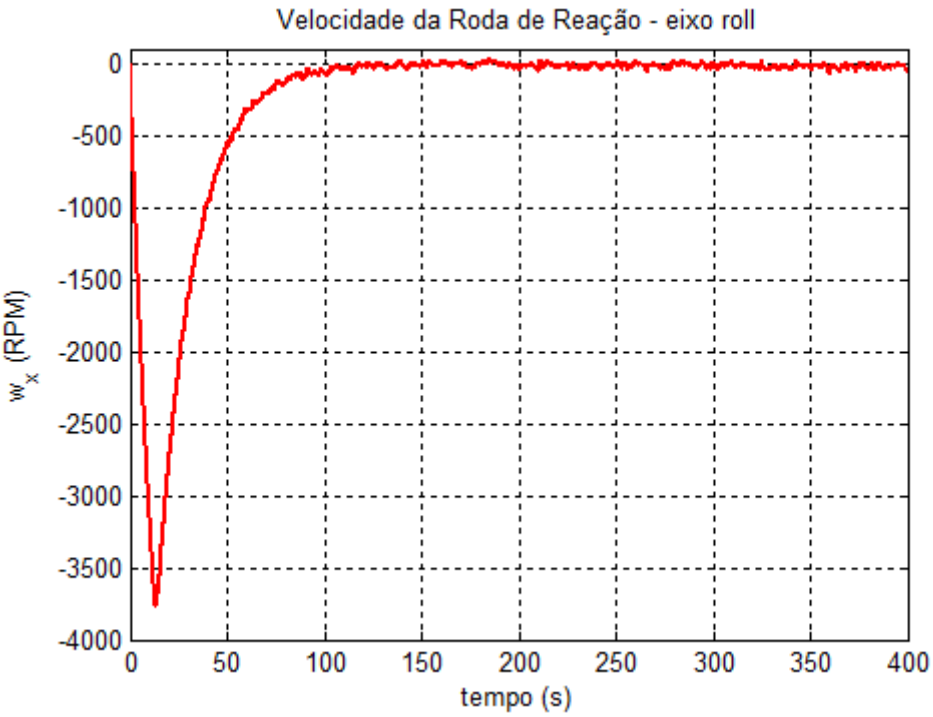


FIGURA 5.33 – Velocidade da roda de reação, eixo x, nas mesmas condições da Figura 5.32.

Assim, para o modelo executado no sistema co-simulado, e baseado na arquitetura padrão, os índices ITAE para os três eixos de controle estão apresentados na Tabela 5.3. Nela, estão mostrados também os valores para quando os sensores do sistema estão sob efeito de ruído gaussiano, na mesma amplitude utilizada no Capítulo 4.

TABELA 5.3 – Índice ITAE do sistema em simulação distribuída, com PID executado em FPGA e ganhos conforme Tabela 5.1.

ITAE	**ϕ**	**θ**	**ψ**
sensor sem ruído	270.573	0.0201	0.9273
sensor c/ ruído gaussiano	286.163	6.6086	7.8234

A Figura 5.34 procura registrar o comportamento da parte implementada no FPGA e seu atendimento do requisito de tempo. Da estratégia apresentada na Figura 5.7, deve-se lembrar que existe uma interdependência entre a execução no FPGA e no SOTR,

através do sinal de ACK, pelo processador tradicional, com o sinal de interrupção gerado pelo FPGA. Em caso de atendimento da interrupção e processamento do modelo no tempo acima do esperado, a resposta do processador (ACK) irá provocar um atraso na liberação do FPGA, fazendo com que o valor medido do contador fique acima do especificado. Para gerar um sinal a cada 10ms, a partir de um oscilador de 40MHz, o contador deve estar ajustado para 400000, como mostra a Figura 5.34.

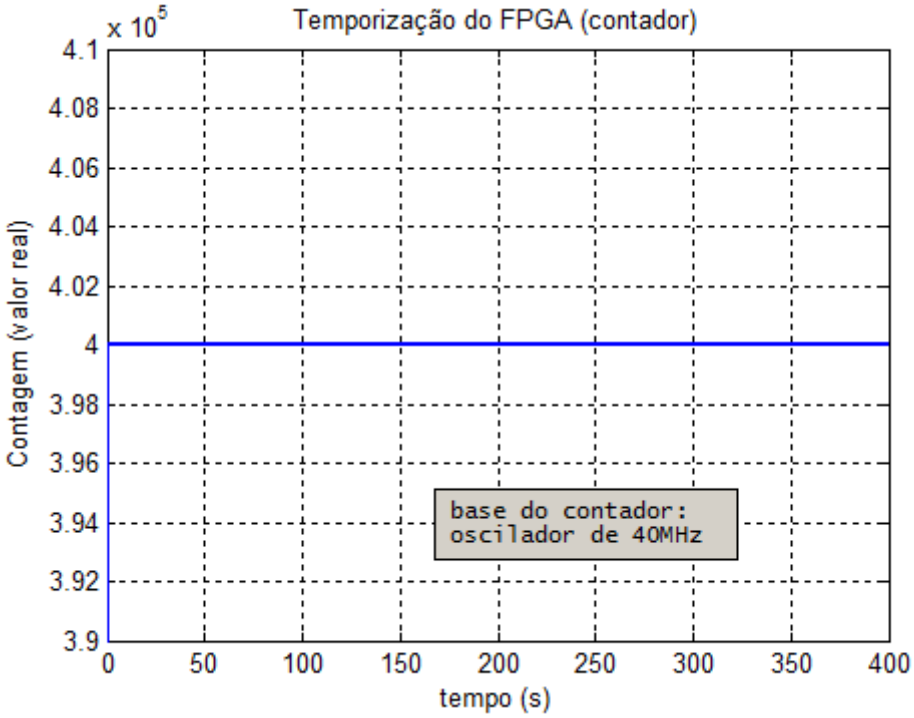


FIGURA 5.34 – Valor real do contador de base de tempo do FPGA.

Para fazer justiça a um sistema operacional (Windows) que não foi projetado para executar tarefas de tempo real, é necessário registrar ainda as condições nas quais foi possível obter o desempenho desejado: além das condições definidas na Seção 5.2.1 serem mandatórias, o comportamento apresentado na Figura 5.34 é atingido em torno de 75% dos casos, mas sempre precisando executar mais de uma vez a simulação.

5.6.2 Resultados no Ambiente Proprietário

No Ambiente Proprietário, buscou-se manter as mesmas condições de ajuste obtidos no Ambiente Padrão, na Seção 5.6.1, incluindo os ganhos dos controladores PID, conforme a Tabela 5.1.

Assim, as Figuras 5.35 a 5.39 mostram os resultados obtidos para a manobra partindo de um desalinhamento de 30° no eixo de rolagem, procura-se realinhar o satélite com relação ao seu referencial VLHL. Nas Figuras 5.35 e 5.36 observa-se o comportamento da atitude do satélite no eixo de rolagem, através do ângulo ϕ, com as condições iniciais (30°, 0°, 0°). Na Figura 5.36, está realçado o momento (111,38s) a partir do qual é atendido o requisito de precisão de apontamento de ±0,05° em menos de 180s. Como esperado, o resultado é idêntico ao da Figura 5.26.

O torque na roda de reação do eixo de "roll" pode ser visto nas Figuras 5.27 e 5.28, nas mesmas condições que para o ângulo ϕ de atitude. De forma similar à Figura 5.28 apenas em menores resoluções de tempo é possível perceber que existe mais de uma forma de onda no mesmo gráfico. Desta vez, porém, pode-se notar (Figura 5.38) que o valor máximo de torque foi atingido por um curto período de tempo. Finalmente, na Figura 5.39 observa-se o comportamento da velocidade da roda de reação do eixo de "roll".

Como se pode notar, ambos os ambientes, o Padrão e o Proprietário, apresentaram resultados semelhantes, conforme esperado. Talvez a maior diferença seja decorrente da arquitetura do Ambiente Proprietário, já apresentado na Figura 5.4 e relacionado à usabilidade do ambiente. Como "Host Computer" e "Target" são equipamentos distintos, o usuário pode acompanhar a resposta do sistema em tempo de execução, não sendo necessário aguardar o final da simulação para verificar o resultado. Existe um desacoplamento entre os processos executados em tempo real e a interface com o usuário. Para simulações com tempos relativamente longos como os relacionados ao satélite, esse pode ser um aspecto relevante a considerar.

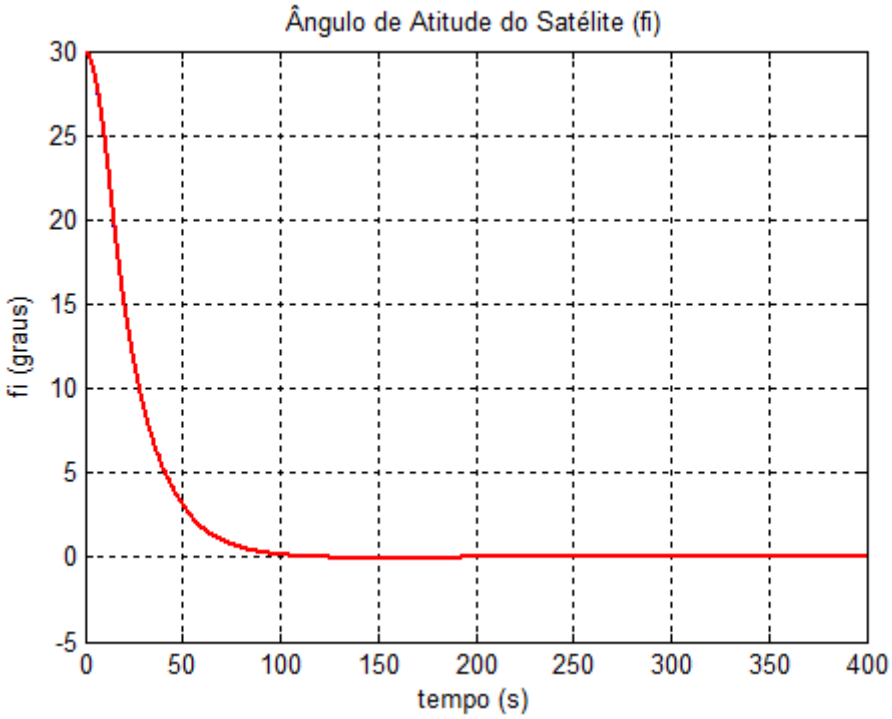


FIGURA 5.35 – Realinhamento do satélite, ângulo ϕ, com PID executado em FPGA no ambiente proprietário, e ganhos conforme Tabela 5.1.

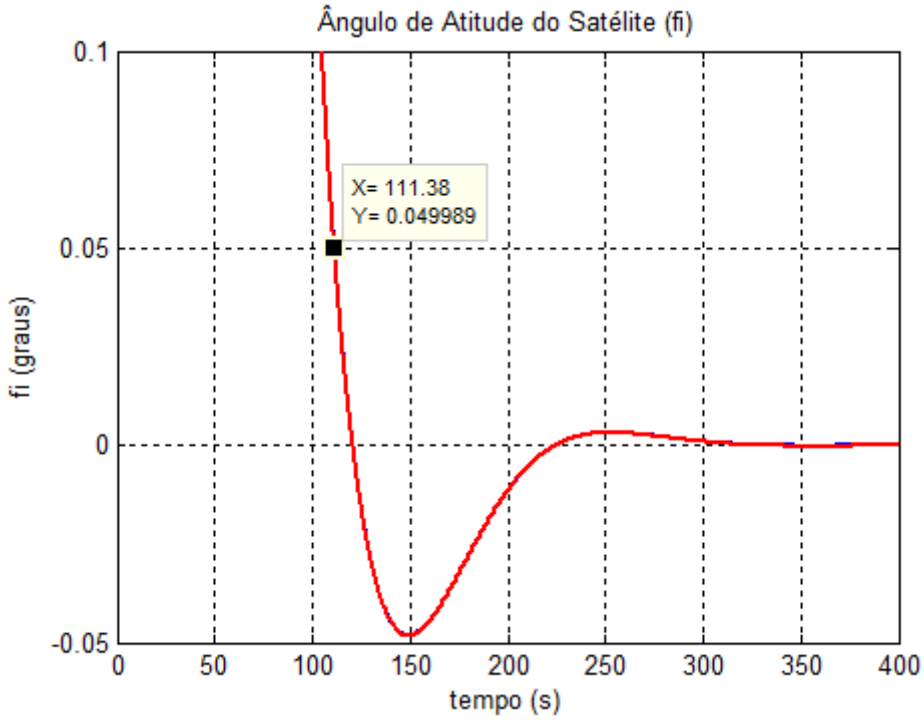


FIGURA 5.36 – Realinhamento do satélite, ângulo ϕ, detalhe da Figura 5.35.

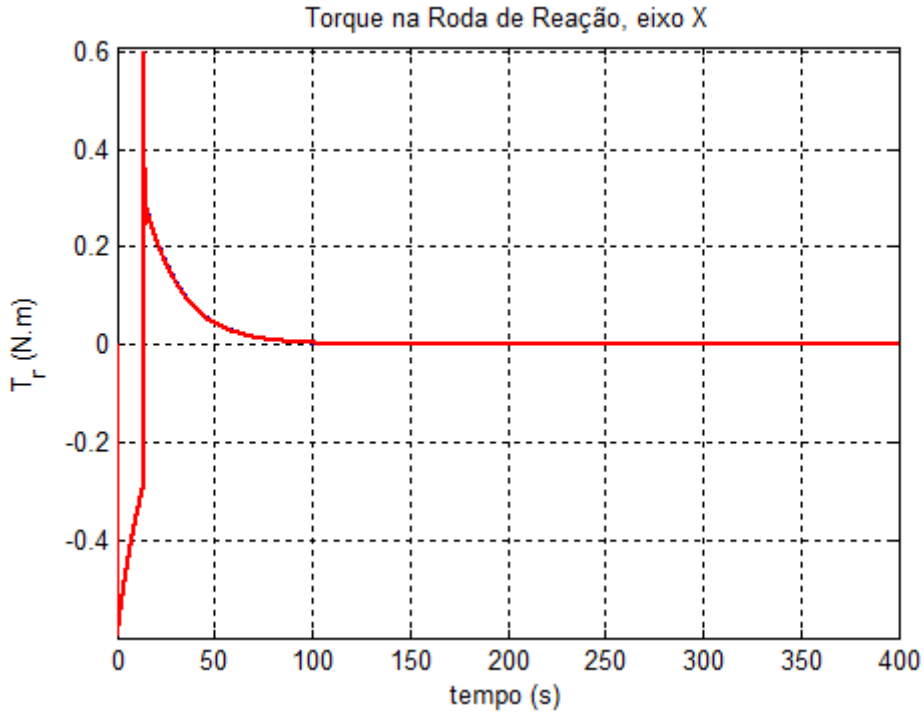


FIGURA 5.37 – Torque na roda de reação, eixo X, com modelo executado em FPGA no ambiente proprietário, e usando os ganhos da Tabela 5.1.

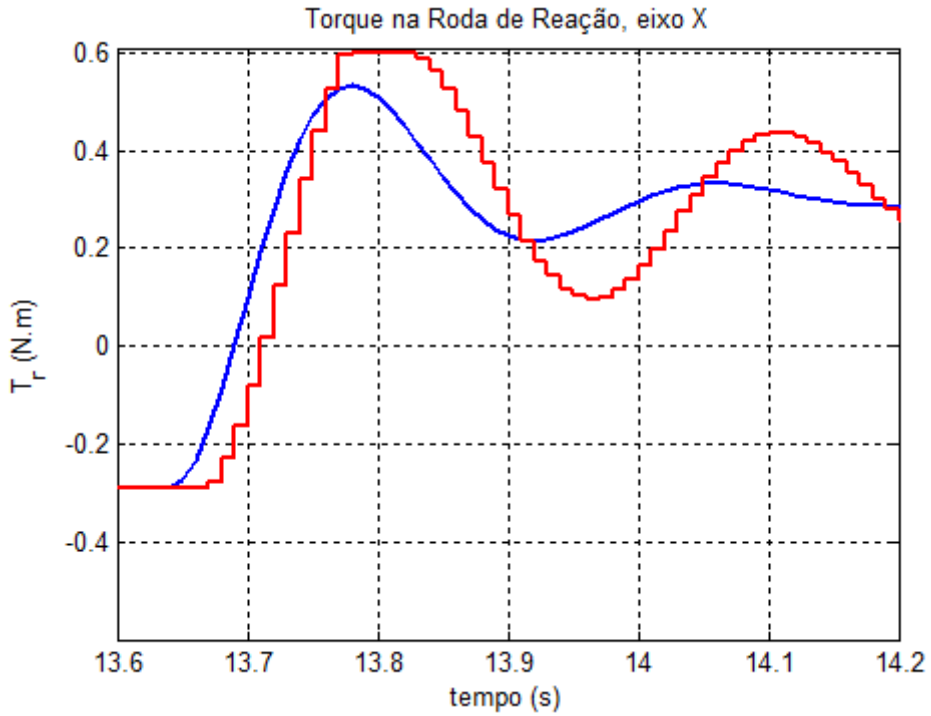


FIGURA 5.38 – Torque na roda de reação, eixo X, detalhe da Figura 5.37.

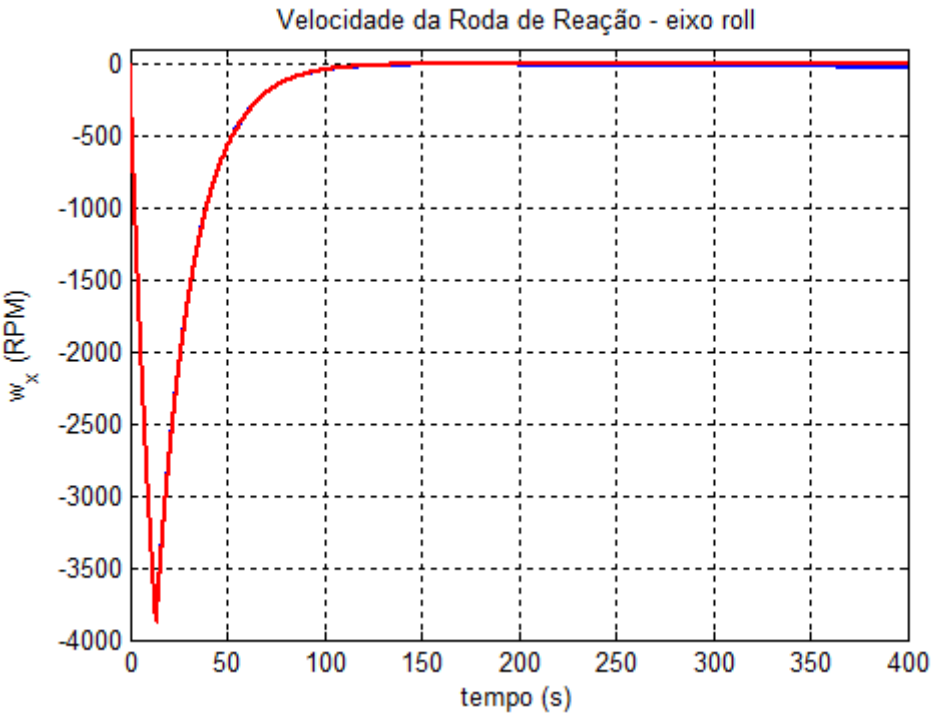


FIGURA 5.39 – Velocidade da roda de reação, eixo X.

CAPÍTULO 6

CASO AERONÁUTICO: MODELAGEM E SIMULAÇÕES

Adotando-se o mesmo fluxo de desenvolvimento adotado no Capítulo 3, neste capítulo procura-se modelar, simular em um ambiente de tempo virtual e migrar para um ambiente co-simulado em tempo real a parte de um sistema FBW relativa ao atuador eletro-hidráulico servo assistido e seu ACE, como apresentado na Seção 2.3.

Os resultados advindos dessa aplicação procuram verificar a abrangência do fluxo de desenvolvimento adotado anteriormente, cuja maior diferença com a aplicação anterior está relacionada com as constantes de tempo envolvidas, neste caso significativamente menores.

Com isso, o maior impacto será na distribuição da simulação. Em um processador tradicional usando sistemas operacionais adaptados para operar com alguns requisitos de tempo real, não será possível conseguir passos de integração menores que 1ms de uma maneira determinística.

A partir desse valor de referência, o modelo deve ser executado no FPGA. Nessa condição, os seus recursos serão exercitados em uma situação mais exigente, com relação aos requisitos decorrentes do passo de integração – tempo menor para execução – e tamanho do algoritmo implementado.

6.1 Definição do Problema

Diferentemente do caso espacial, no caso aeronáutico em estudo, pressupõe-se que existam outras malhas, externas ao atuador, sua EHSV e o ACE. Dessa forma, requisitos aplicáveis neste caso são mais específicos a uma resposta padrão para uma malha de controle de posição e pouco relacionados com o veículo.

Assim, serão adotados como requisitos de desempenho um valor de sobreelevação menor que 5% e tempo de acomodação de 60 ms para um degrau de 10% de excursão do atuador.

6.2 Modelagem do Sistema

Conforme já indicado anteriormente no Capítulo 2 e na Figura 2.4, o sistema considerado a ser modelado neste trabalho é um atuador hidráulico, com seus componentes de amplificação de potência (EHSV e fonte hidráulica) e controle (ACE).

A fonte de potência hidráulica será considerada como uma fonte ideal, com as seguintes características:

- Pressão (P_S): 3000 psi;
- Fluxo máximo (Q_{PUMP}): $1{,}67 \cdot 10^{-3}$ m^3/s.

6.2.1 Servoválvula Eletro-Hidráulica (EHSV)

A servoválvula modelada neste trabalho foi apresentada na Seção 2.3. Ela é uma válvula convencional de duplo estágio, sendo o primeiro estágio bocal-palheta e o segundo estágio uma válvula carretel de centro crítico e 4 vias.

Usando a Figura 2.5 como referência, a Figura 6.1 mostra o mecanismo de acionamento elétrico do primeiro estágio e o correspondente efeito na válvula carretel.

O acionamento elétrico é baseado no seguinte princípio de operação: imãs permanentes mantém os pólos polarizados; a injeção de corrente contínua nas bobinas provoca uma força magnetomotriz diagonalmente oposta nos entreferros, provocando um desalinhamento na armadura. Dessa forma, a força lateral na palheta é proporcional ao torque provocado pela corrente do motor, provocando o movimento da válvula carretel.

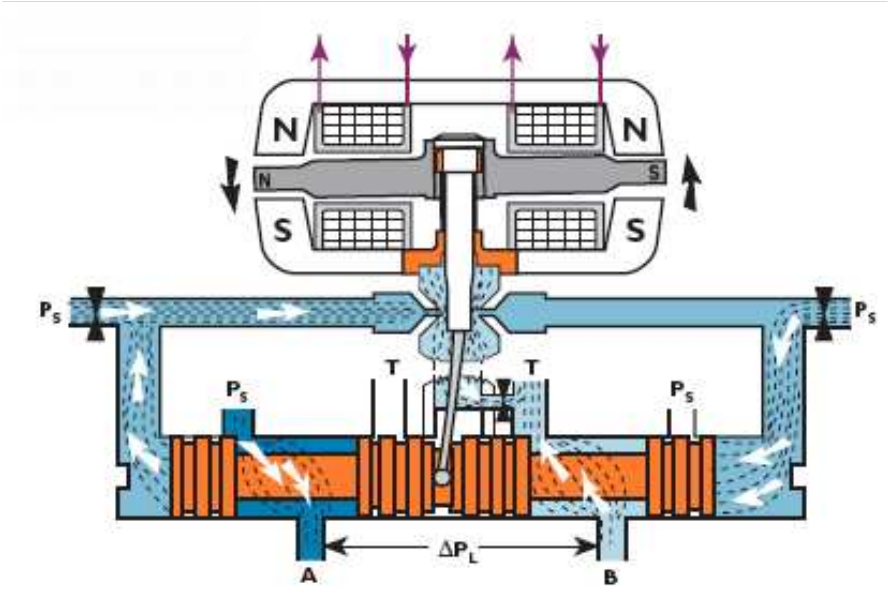


FIGURA 6.1 – EHSV: Resposta da válvula a um degrau de corrente de entrada.
FONTE: Moog (2000).

Por simplicidade, as características elétricas do motor do primeiro estágio podem ser modeladas como um circuito série L-R, negligenciando-se qualquer força contraeletromotriz gerada pela carga. A função de transferência de um circuito L-R é:

$$\frac{I_V(s)}{U_V(s)} = \frac{1}{s \cdot L_C + R_C} = \frac{R_C}{s \cdot \frac{L_C}{R_C} + 1} \tag{6.1}$$

onde L_C é a indutância da bobina do motor, R_C a resistência do enrolamento.

Uma servoválvula é um dispositivo complexo que exibe uma resposta não linear. Para a formulação precisa do modelo matemático da válvula é necessário o conhecimento dos seus diversos parâmetros internos. De fato, parâmetros como tamanhos de orifícios e agulhas, constantes de mola, geometria do carretel, dentre outros são ajustados pelo fabricante para calibrar a resposta da válvula, e normalmente não estão disponíveis ao usuário (POLEY, 2005).

Ao modelar servoválvulas, normalmente aplica-se a análise de pequenas perturbações em torno de um ponto de operação. Como resultado, esses modelos são normalmente

baseados em equações diferenciais de primeira ou segunda ordens, sendo apenas uma aproximação do comportamento real.

No entanto, como a servoválvula não é o elemento dinâmico fundamental dentro das faixas de freqüências em estudo, a dinâmica da servoválvula pode ser aproximada por uma função de transferência de segunda ordem, sem grandes impactos na exatidão do sistema modelado (POLEY, 2005).

O diagrama de Bode da Figura 6.2 representa a resposta dinâmica típica de uma servoválvula. Supondo uma aproximação de segunda ordem, pode-se adotar como modelo um filtro passa baixas de mesma ordem, adotando-se $\zeta = 0.48$ como coeficiente de amortecimento, e $\omega_V = 400.5$ rad/s para a freqüência crítica, conforme a eq. 6.2.

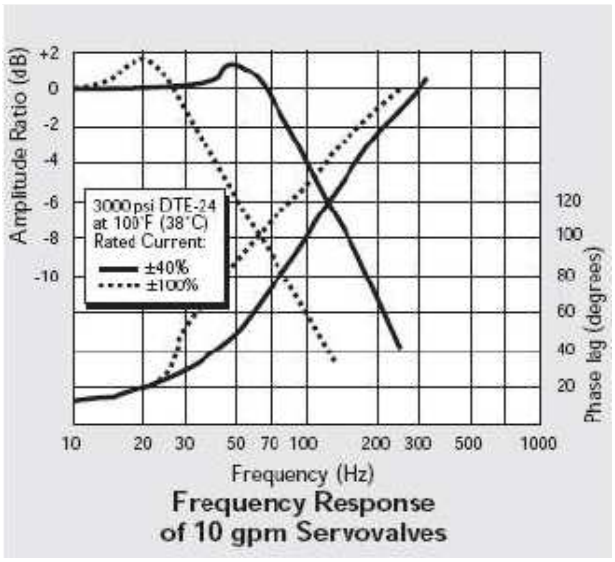


FIGURA 6.2 – Resposta em freqüência de uma servoválvula típica.
FONTE: Moog (2000).

$$\frac{X_V(s)}{I_V(s)} = \frac{\omega_V^2}{s^2 + 2 \cdot \varsigma \cdot \omega_V + \omega_V^2} = \frac{160400}{s^2 + 384.5 \cdot s + 160400} \tag{6.2}$$

Dessa forma, conjugando as eqs 6.1 e 6.2 pode-se obter um modelo simplificado da dinâmica da válvula carretel da EHSV.

Relacionado ao fluxo de controle fornecido pela servoválvula, ele é proporcional ao deslocamento da válvula carretel para uma carga constante. Com carga variável, e considerando o regime de fluxo turbulento, o fluxo do fluido é proporcional ao quadrado da queda de pressões na válvula (MERRIT, 1967 segundo ANDRIOLI, 2005 e POLEY, 2005), conforme a eq. 6.3.

$$Q_L = Q_{NOM} \cdot i_V^* \cdot \sqrt{\frac{P_V}{P_{NOM}}} \tag{6.3}$$

Na equação 6.3 acima, Q_L é o fluxo hidráulico fornecido ao atuador, Q_{NOM} é o valor nominal de fluxo na condição de pressão nominal P_{NOM}, e i^*_V é a corrente do motor normalizada com relação ao seu valor máximo. A queda de pressão na válvula P_V é dada por $P_V = P_S - P_T - P_L$, onde P_S, P_T e P_L são respectivamente as pressões na linha de alimentação, na de retorno e a pressão no atuador.

A Tabela 6.1 apresenta os valores utilizados neste trabalho, referente à servoválvula modelo EHSV G761, fabricada pela Moog.

TABELA 6.1 – Parâmetros da EHSV adotados neste trabalho.

L_C	0,0015 [H]
R_C	100 [Ω]
I_{V_MAX}	0,04 [A]
ω_V	400,5 [rad/s]
ζ	0,48
Q_{NOM}	$0{,}6 \cdot 10^3$ [m³/s]
P_{NOM}	1000 [psi]

6.2.2 Atuador Linear

Com relação ao atuador, pode-se modelá-lo considerando dois aspectos: o movimento dinâmico do pistão aplicando a 2ª lei de Newton, e o cálculo das pressões nas câmaras. Para fins de modelagem deste componente, a Figura 2.7 pode ser usada como referência.

Para as pressões nas câmaras, existe um relacionamento entre o fluxo de controle e a pressão, que provoca um efeito mola nas câmaras e gera uma freqüência de ressonância na interação com a massa do pistão. Esse efeito pode ser modelado usando a equação de continuidade de fluxo, que relaciona o fluxo resultante em um recipiente com o volume e pressão do fluido:

$$\sum Q_{in} - \sum Q_{out} = \frac{dV}{dt} + \frac{V}{\beta}\frac{dP}{dt} \tag{6.4}$$

O termo do lado esquerdo da equação é o fluxo resultante fornecido à câmara pela servoválvula. O primeiro termo do lado direito é o fluxo consumido pela variação de volume provocada pelo movimento do pistão. O segundo termo é normalmente dominado pela compressibilidade do fluido hidráulico, supondo que a câmara é perfeitamente rígida, e possui um valor β para o "bulk modulus" do óleo (MERRIT, 1967) segundo Poley (2005). Óleos minerais usados como fluidos em sistemas hidráulicos possuem um "bulk modulus" na região de $1.4 \cdot 10^9$ N/m.

A equação 6.4 pode ser rearranjada para obter-se a pressão instantânea na câmara A (vide Figura 2.7) como segue:

$$P_A = \frac{\beta}{V_A}\int\left(Q_A - \frac{dV_A}{dt}\right)dt \tag{6.5}$$

De forma equivalente, essa mesma equação 6.5 é válida para a câmara B.

Uma vez que as pressões em ambas as câmaras é conhecida, pode-se calcular a força resultante agindo sobre o pistão (F_P) através da multiplicação com a sua área (A_P), conforme a eq. 6.6:

$$F_P = (P_A - P_B) \cdot A_P \tag{6.6}$$

Por outro lado, aplicando a 2ª lei de Newton para o pistão, o seu movimento pode ser definido como:

$$F_P = M_P \frac{d^2 X_P}{dt^2} + F_f + K_L \cdot X_P \tag{6.7}$$

onde:

- X_P : posição do pistão;
- M_P : massa do pistão;
- K_L : coeficiente de amortecimento de uma carga acoplada ao pistão;
- F_f : coeficiente de atrito, devido ao selo de óleo no pistão e nas extremidades do êmbolo com as câmaras (vide Figura 2.7).

A Tabela 6.2 apresenta os valores utilizados neste trabalho, referente ao atuador da série A085, fabricado pela Moog.

TABELA 6.2 – Parâmetros do atuador adotados neste trabalho.

M_P	9.0 [Kg]
A_P	$4.387*10^{-3}$ [m^2]
Excursão do Pistão	0.1016 [m]

6.2.3 Controlador de Posição

O controlador de posição no contexto dessa aplicação, é normalmente conhecido como ACE ("Actuator Control Electronics"). Em algumas situações esse controlador é projetado para funcionamento em malha aberta (ANDRIOLI, 2005).

No contexto deste trabalho, a configuração adotada será em malha fechada, com a mesma estratégia de controle (PID) adotada no caso espacial.

6.3 Simulação em Tempo Virtual

Utilizando o mesmo ambiente de tempo virtual que o usado no Capítulo 4, o caso aeronáutico é modelado a partir do equacionamento obtido na Seção 6.2. A Figura 6.3 apresenta o diagrama Simulink representando o sistema de controle de posição do atuador hidráulico, seu servo acionamento e controle.

A Figura 6.4 mostra a implementação do modelo da EHSV a partir das equações 6.1 e 6.3. A dinâmica da válvula carretel é mostrada na Figura 6.7, implementando a equação 6.2.

O modelo do atuador está dividido em três blocos: dois que implementam o cálculo da pressão em cada câmara (Figura 6.6) a partir da equação 6.5, e o terceiro que implementa a dinâmica do pistão (Figura 6.5) conforme as equações 6.6 e 6.7.

O bloco *ServoControllerDig* é a mesma implementação do caso espacial, ou seja, o diagrama apresentado na Figura 4.9, Capítulo 4, que também é utilizado neste caso.

Os valores de inicialização dos modelos foram calculados a partir dos dados apresentados nas Tabelas 6.1 e 6.2. A posição – X_P – central do pistão foi adotada como o seu zero. Portanto, valores positivos de X_P aumentam o volume da câmara A e diminuem o da câmara B, e o contrário para valores negativos de posição.

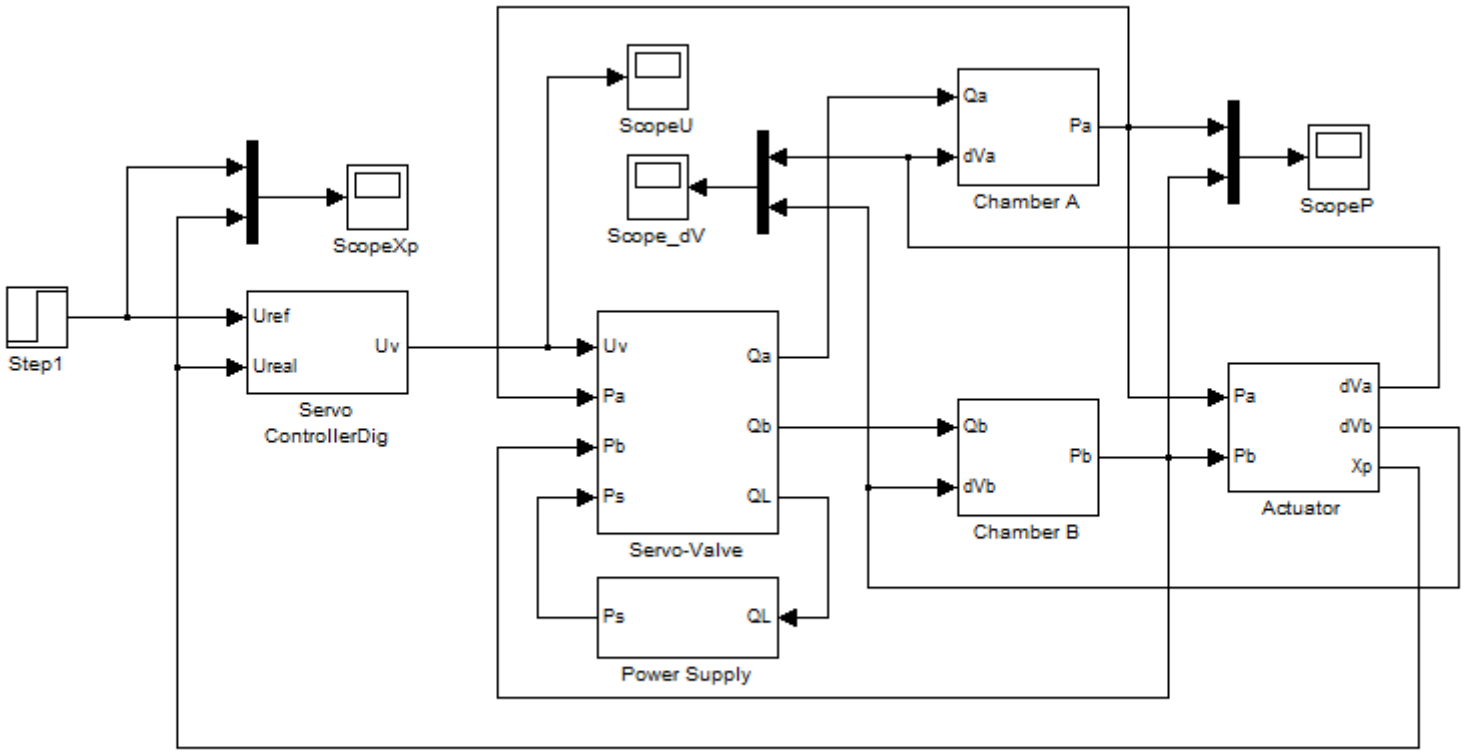


FIGURA 6.3 – Modelo da aplicação aeronáutica: diagrama principal.

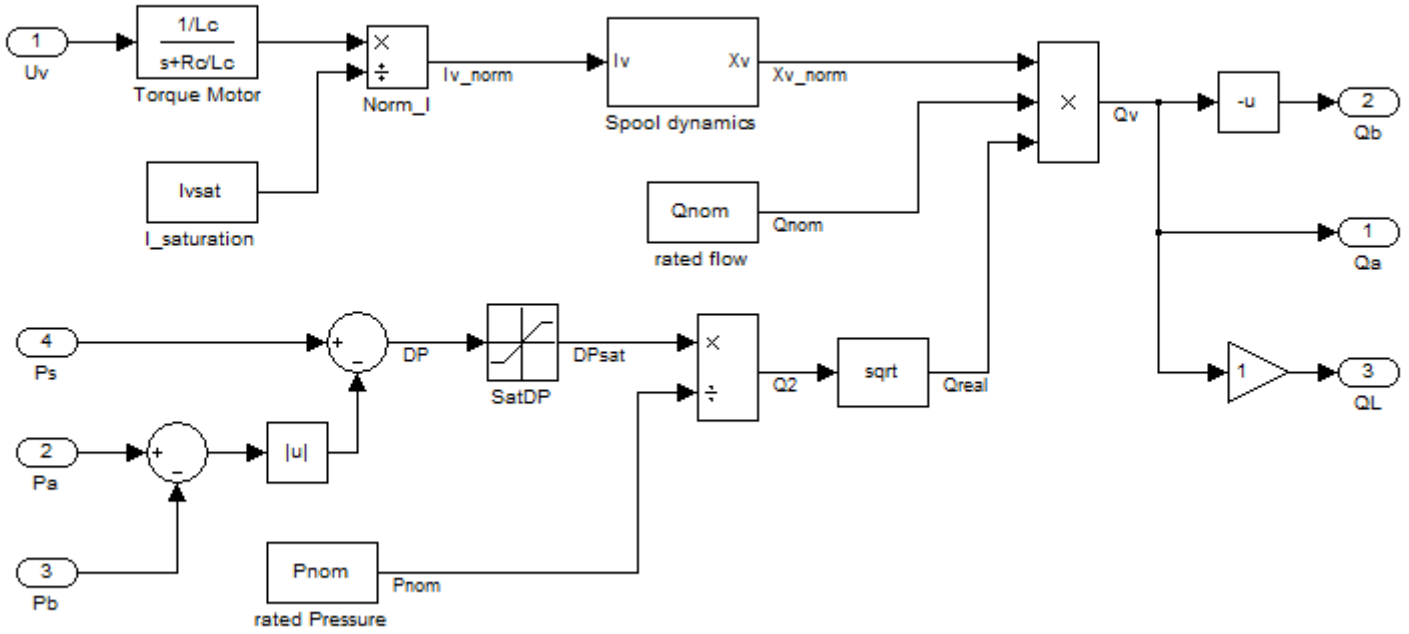


FIGURA 6.4 – Modelo da aplicação aeronáutica: bloco *Servo-Valve*.

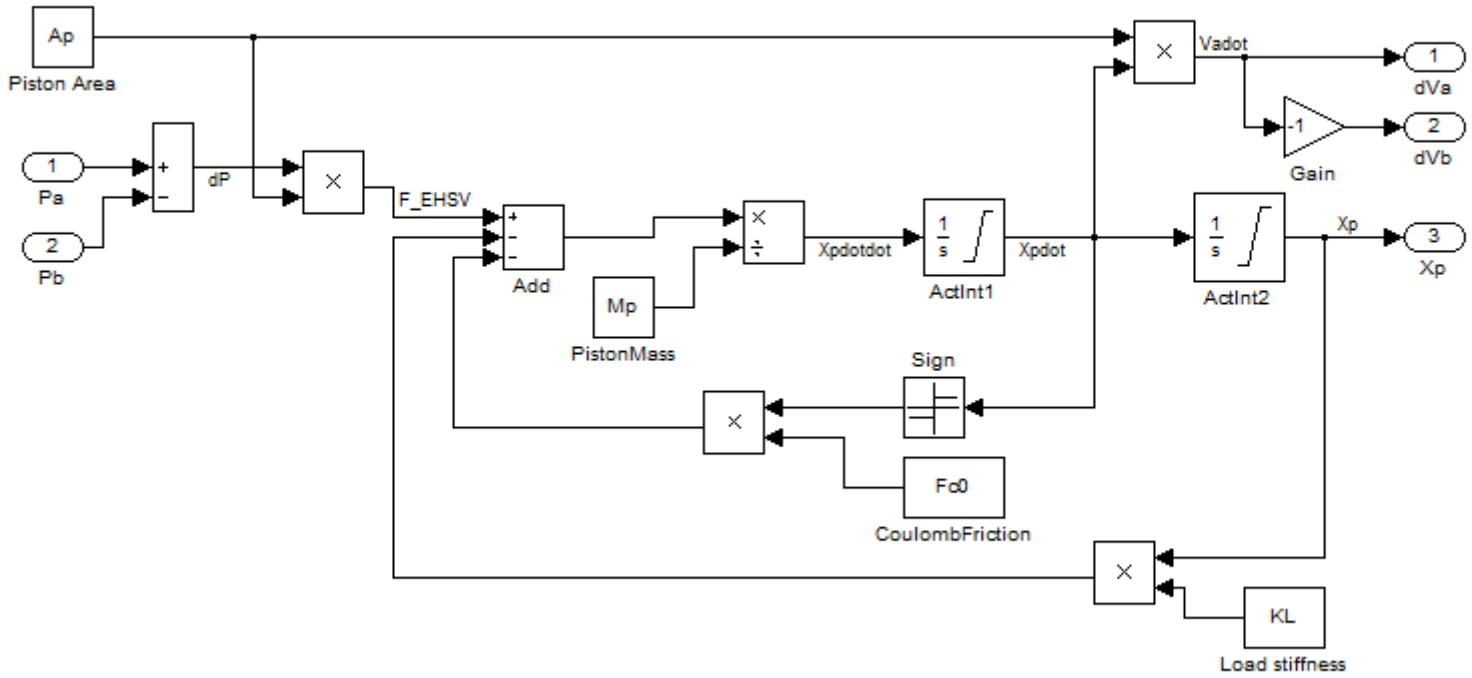


FIGURA 6.5 – Modelo da aplicação aeronáutica: bloco *Actuator*.

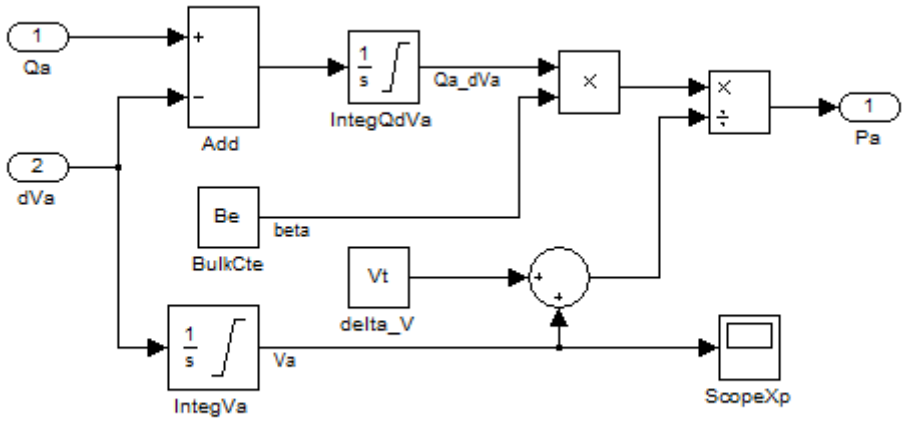


FIGURA 6.6 – Modelo da aplicação aeronáutica: bloco *Chamber A(B)*.

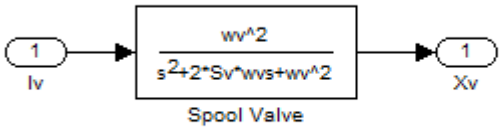


FIGURA 6.7 – Modelo da aplicação aeronáutica: bloco *Spool dynamics*, dentro de *Servo-Valve.*

Tipicamente, a menor constante de tempo de um sistema hidráulico é da ordem de 1 ms. Por isto, o passo de integração utilizado no modelo foi de 0.1 ms.

Os valores dos ganhos do controlador PID foram obtidos por tentativa e erro, seus ganhos estão apresentados na Tabela 6.3 e a resposta do sistema nas Figuras 6.8 a 6.11.

TABELA 6.3 – Caso aeronáutico: ganhos PID no ambiente simulado.

K_p	K_i	K_d
$K_p = 1041,7$	$K_i = 3,75$	$K_d = 0,8$

Nessa resposta, os coeficientes de atrito e amortecimento do atuador, F_f e K_L respectivamente, foram zerados, negligenciando ambos os efeitos.

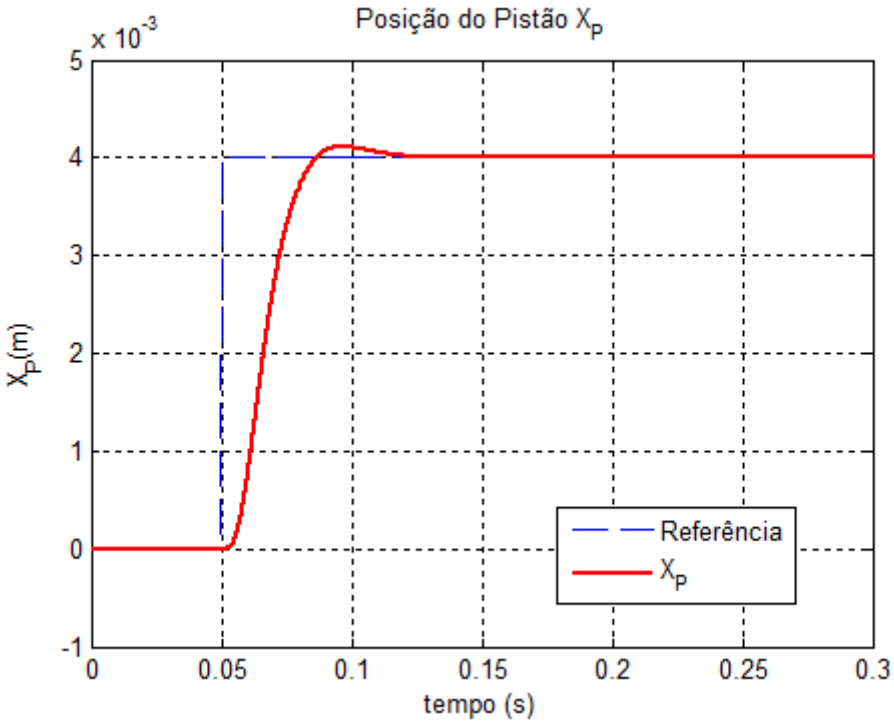


FIGURA 6.8 – Posição do pistão: resposta a degrau.

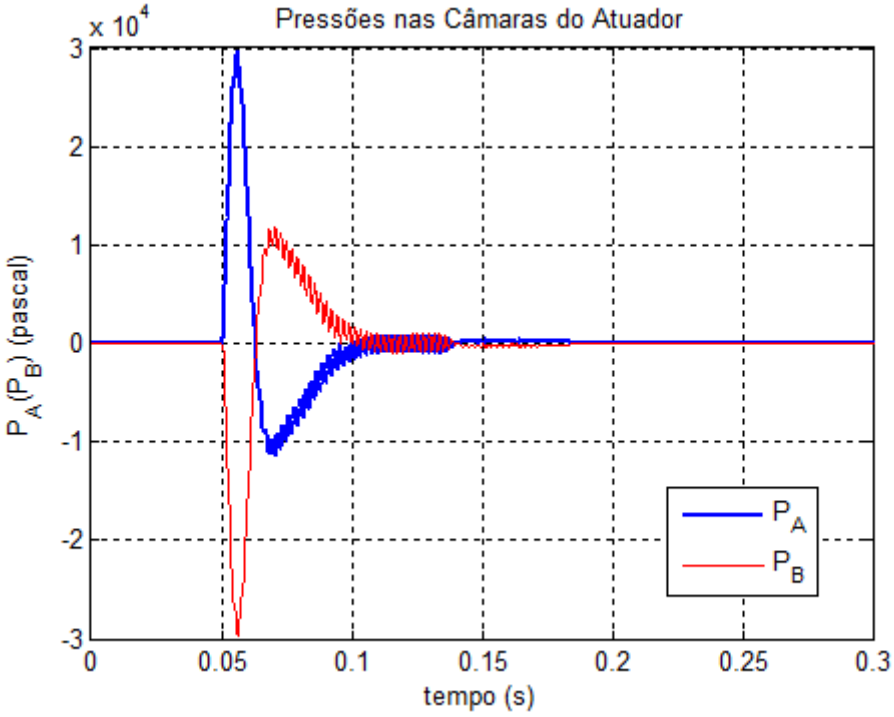


FIGURA 6.9 – Pressões nas câmaras do atuador, durante a resposta a degrau.

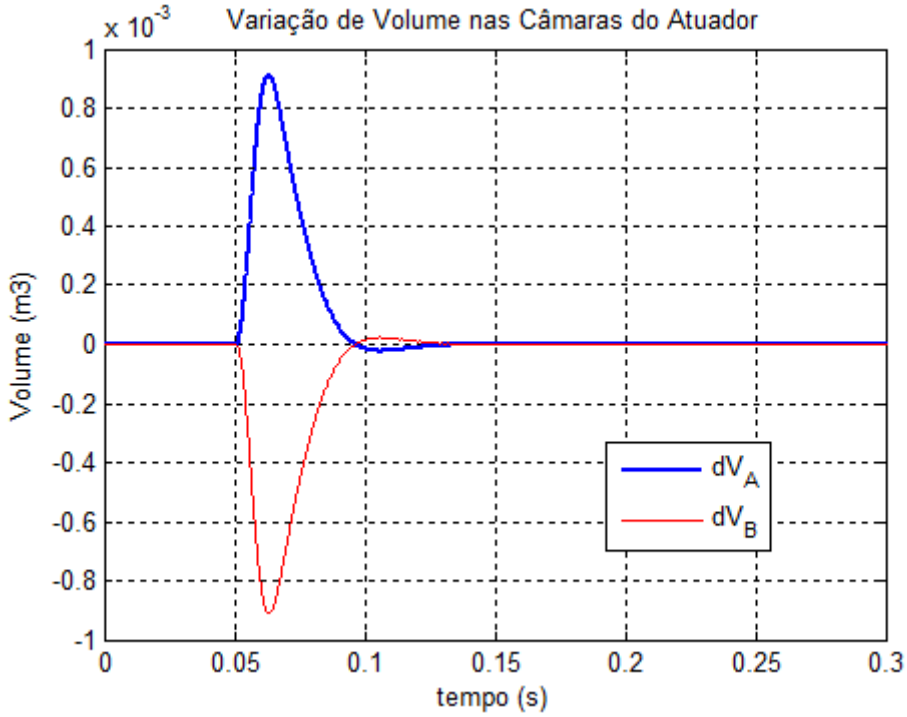


FIGURA 6.10 – Variação de volume nas câmaras do atuador, durante a resposta a degrau.

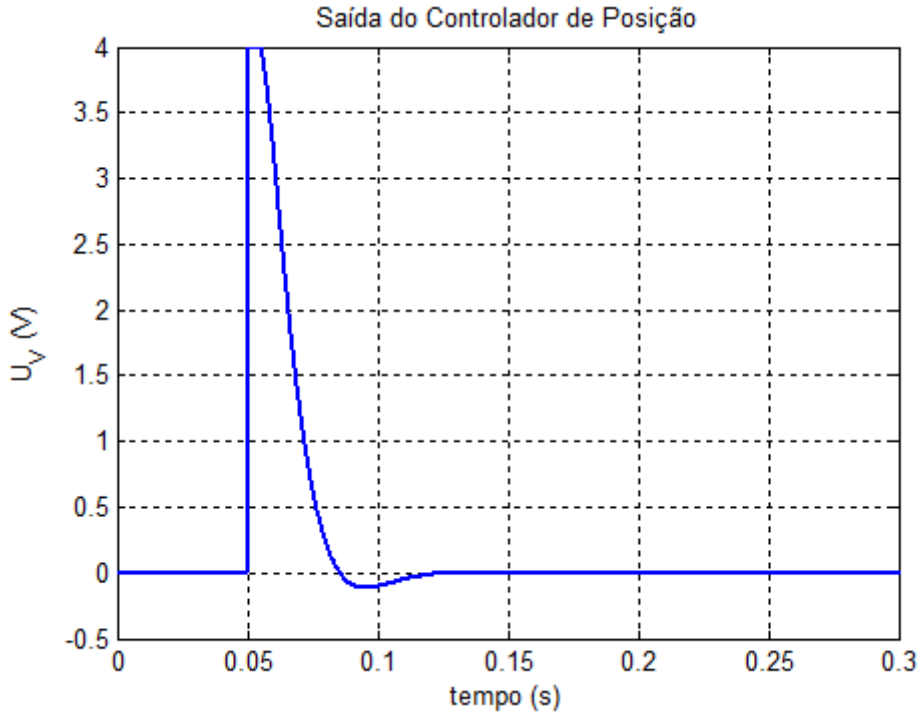


FIGURA 6.11 – Saída do controlador de posição, durante a resposta a degrau.

6.4 Simulação em Tempo Real

Conforme já antecipado, em virtude do passo de integração adotado – 100μs – a estratégia neste caso será migrar todo o modelo da aplicação aeronáutica para o FPGA. O modelo em questão é o apresentado na Figura 6.3.

O processo de migração é o mesmo que o descrito no Capítulo 5 (Seção 5.3), e produz como resultado blocos implementados em LabVIEW FPGA, conforme pode ser observado no seu diagrama principal, na Figura 6.13. Essa figura está desmembrada em duas partes: na primeira o algoritmo do modelo é implementado, com seus resultados sendo enviados a uma fila FIFO interna do FPGA; e, na segunda parte, os dados da fila FIFO interna são lidos e enviados ao processador do cRIO através de DMA a cada 10 passos de execução (1ms no caso). Por questões de otimização de espaço e tempo de execução, apenas 5 grandezas são transferidas:

- saída do controlador;
- fluxo de saída da servo-válvula para a câmara A;
- posição do êmbolo do atuador;
- referência de posição do êmbolo;
- tempo decorrido entre dois passos de execução do modelo.

As duas partes são executadas em paralelo, utilizando a fila FIFO interna do FPGA como mecanismo de transferência de dados entre os laços.

A Tabela 6.4 mostra a comparação entre os blocos gerados em Simulink e LabVIEW FPGA, do ponto de vista funcional. Da mesma forma que no caso espacial, o processo de migração foi realizado considerando o uso de uma biblioteca de operações (soma, subtração, multiplicação, divisão e raiz quadrada) em ponto flutuante, "single-precision", desenvolvida como um acessório para os casos estudados neste trabalho.

TABELA 6.4 – Equivalência funcional entre os blocos Simulink e LabVIEW FPGA para a aplicação aeronáutica.

Simulink	LabVIEW FPGA
ServoControllerDig	EHSV – PID
Servo-Valve *Power Supply*	EHSV – ODE RK4v1
Chamber A *Chamber B* *Actuator*	ActFull – ODE RK4

Relacionado ao controlador do sistema – *ServoControllerDig* e EHSV-PID – foi utilizada a mesma implementação adotada no caso espacial.

Quanto aos modelos do atuador e servoválvula, também de forma similar ao caso espacial, o procedimento utilizado foi migrá-los para LabVIEW FPGA a partir do código C gerado automaticamente pelo Simulink (usando o RTW), baseado em integradores Runge-Kutta de 4ª ordem.

6.4.1 Ambiente de Emulação

O sistema escolhido para estudo no caso aeronáutico possui uma aplicação prática bastante imediata e significativa: substituir, ou pelo menos reduzir a dependência de, um sistema hidráulico com atuadores reais em um laboratório para validar o desenvolvimento das leis de controle para sistemas de comando de vôo de uma aeronave. Assim, nesse caso, busca-se uma emulação de parte de um sistema cuja interface (pelo menos elétrica) possa ser trocada entre os componentes reais e simulados sem que o elemento sob teste perceba diferenças significativas.

Como propósito deste trabalho, apenas a parte computacional dessa emulação será abordada. No entanto, deve-se ter em mente que a incorporação de canais de entrada e

saída analógicas no ambiente proprietário (ver Seção 5.1.1) é bastante simples, decorrente da própria arquitetura do equipamento.

Ademais, resultados preliminares do processo de migração eliminaram o ambiente padrão (PC com cartão FPGA no formato PCI) como alternativa de execução: ao compilar apenas o bloco da servo-válvula (EHSV–ODE RK4v1), a quantidade de recursos utilizada no FPGA ultrapassou 1M gates. Com isso, apenas o ambiente proprietário baseado no cRIO com FPGA de 3M gates foi considerado.

Com relação ao ambiente proprietário, a Figura 6.12 mostra a seqüência de atividades para a execução do modelo. Ela é uma adaptação das seqüências utilizadas no caso espacial (ver Figuras 5.4 e 5.6). Aqui, primeiramente o usuário envia o comando para iniciar a execução do modelo a *Driver VI* [1], que reenvia [2] esse comando para o elemento que realmente o executa, *FPGA VI*. A seguir, a cada 10 passos de execução (1ms), *FPGA VI* envia os dados dos últimos passos a *Driver VI* [3], processo que se repete até o final da execução do modelo. Ao final da execução, *Driver VI* envia os dados de todos os passos da execução do modelo para *Host VI* [4].

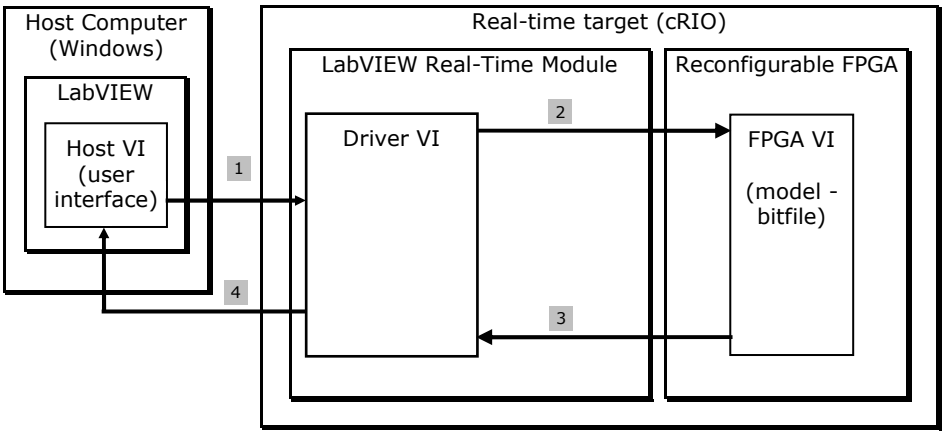


FIGURA 6.12 – Estrutura do ambiente de emulação do caso aeronáutico.

A Figura 6.16 mostra a parte do recebimento dos dados vindos do FPGA implementada em *Driver VI*.

6.4.2 Processo de Otimização da Implementação para FPGA

Durante o processo de migração do modelo implementado em Simulink para FPGA, surgiram alguns obstáculos que dificultaram a implementação em LabVIEW FPGA. Como a superação desses obstáculos só foi possível adotando mecanismos que aproveitam algumas das características mais marcantes do FPGA, torna-se relevante deixar registrado esse processo neste trabalho.

Inicialmente, o desempenho temporal do resultado da migração ficou bastante aquém esperado. Enquanto que o sistema inteiro deveria ser executado em 100μs, apenas um dos blocos (*EHSV–ODE RK4v1*) já consumia quase 300μs. Ao se procurar paralelizar intensamente os algoritmos utilizados (forma automática disponível na ferramenta), os recursos do FPGA foram consumidos, gerando falha na compilação por incapacidade do compilador na alocação dos recursos.

Ao se examinar mais profundamente o algoritmo e sua execução, operações básicas como soma/subtração e multiplicação, usadas extensivamente, estavam sendo alocadas pelo compilador como um recurso único. Ou seja, é como se existisse apenas um operador de soma/subtração e outro a multiplicação para todo o algoritmo. Mandar o compilador tratar todas as instâncias de soma/subtração e multiplicação existentes nos diagramas como independentes (o método automático da ferramenta) ultrapassou os recursos do componente.

Assim, a alternativa foi implementar um mecanismo manual de alocação de recursos, na forma de cópias físicas do mesmo bloco (arquivos independentes na estrutura de blocos utilizada) e posteriormente alocadas no diagrama. Ao se utilizar 4 instâncias de soma/subtração e 3 instâncias de multiplicação, conseguiu-se reduzir o tempo de execução por volta de 3 vezes, utilizando aproximadamente 90% dos recursos do FPGA.

A Figura 6.14 mostra a hierarquia de subrotinas (ou subVIs, no jargão LabVIEW), onde pode-se observar as múltiplas instâncias manualmente alocadas na implementação LabVIEW FPGA. As instâncias estão identificadas por um pequeno quadrado amarelo, com um número em seu interior, no canto superior direito dos blocos.

Um exemplo da estratégia de alocação manual pode ser observada na Figura 6.15, retratando uma parte do algoritmo de integração Runge-Kutta ao utilizar 2 instâncias dos blocos *SglByteA±v01* – soma/subtração – e *SglByteAmul* – multiplicação. Dessa forma consegue-se um paralelismo parcial com relação ao indicado no diagrama. Parcial porque foram utilizados apenas 2 conjuntos de somadores e multiplicadores; para que o paralelismo do código fosse total seriam necessários 6 conjuntos.

Ainda assim, otimizações adicionais foram necessárias, já que os 3 blocos – controlador, modelo da servo-válvula e modelo do atuador – ligados em série consumiam aproximadamente 250μs.

A alternativa adotada foi utilizar a técnica de "pipelining" apresentada no Capítulo 2 (simplificadamente mostrada nas Figuras 2.13 e 2.14 para a implementação LabVIEW). Com isso, os três blocos citados anteriormente passam a ser executados em paralelo, ao invés de sequencialmente. Para tanto: a saída do primeiro bloco a ser executado (neste caso o controlador - *EHSV – PID*) é armazenada, o bloco seguinte usa como entrada a saída do passo anterior do primeiro bloco, e assim sucessivamente. O resultado pode ser observado no diagrama principal da implementação do caso aeronáutico, Figura 6.13.

Como resultado final, o sistema definido para o caso aeronáutico pode ser executado dentro dos 100μs especificado(equivalente a 4000 contagens do oscilador de 40MHz).

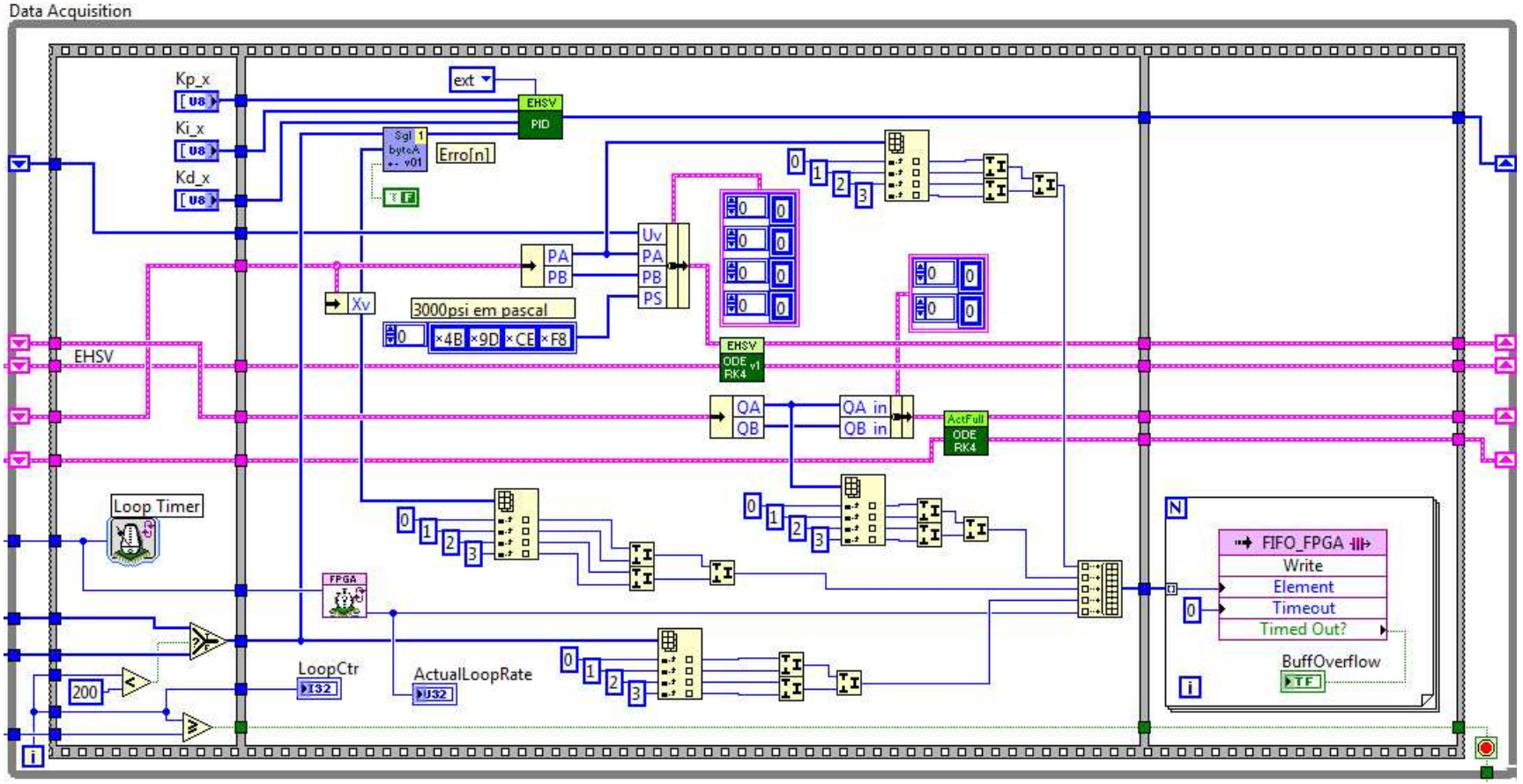


FIGURA 6.13 – Diagrama principal da implementação do caso aeronáutico em LabVIEW FPGA.
(continua)

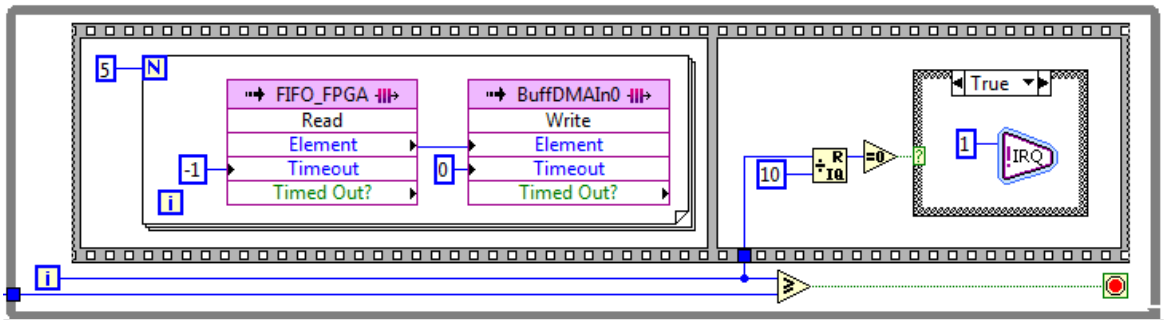


FIGURA 6.13 – Conclusão.

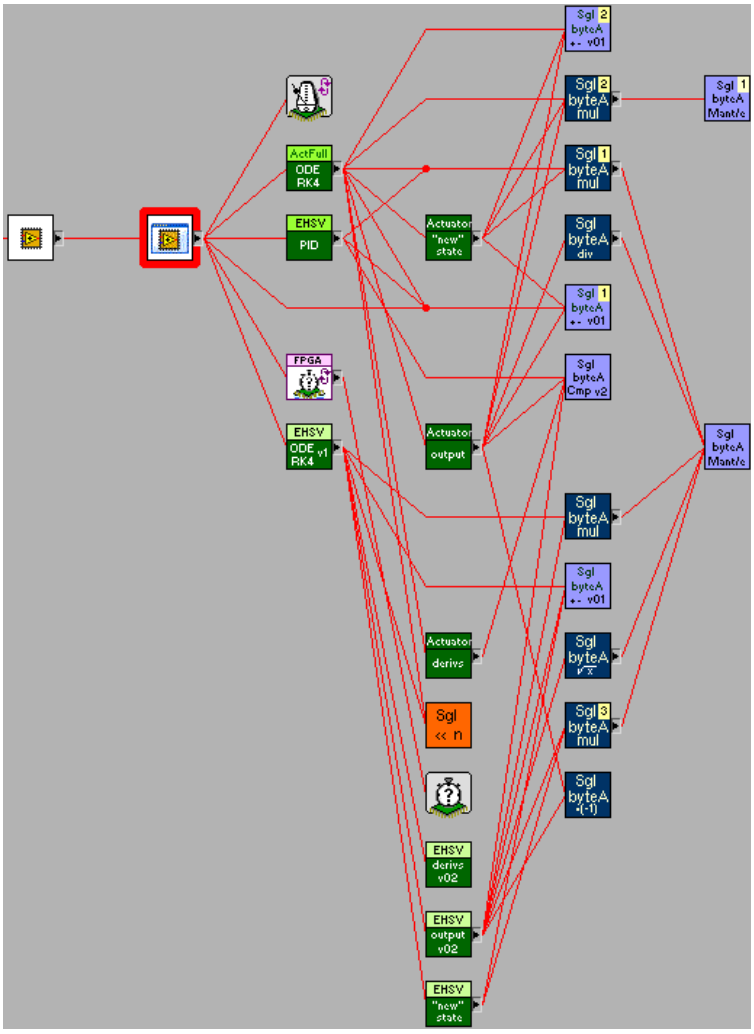


FIGURA 6.14 – Estrutura hierárquica de subVIs na implementação do caso aeronáutico em LabVIEW FPGA (*FPGA VI*).

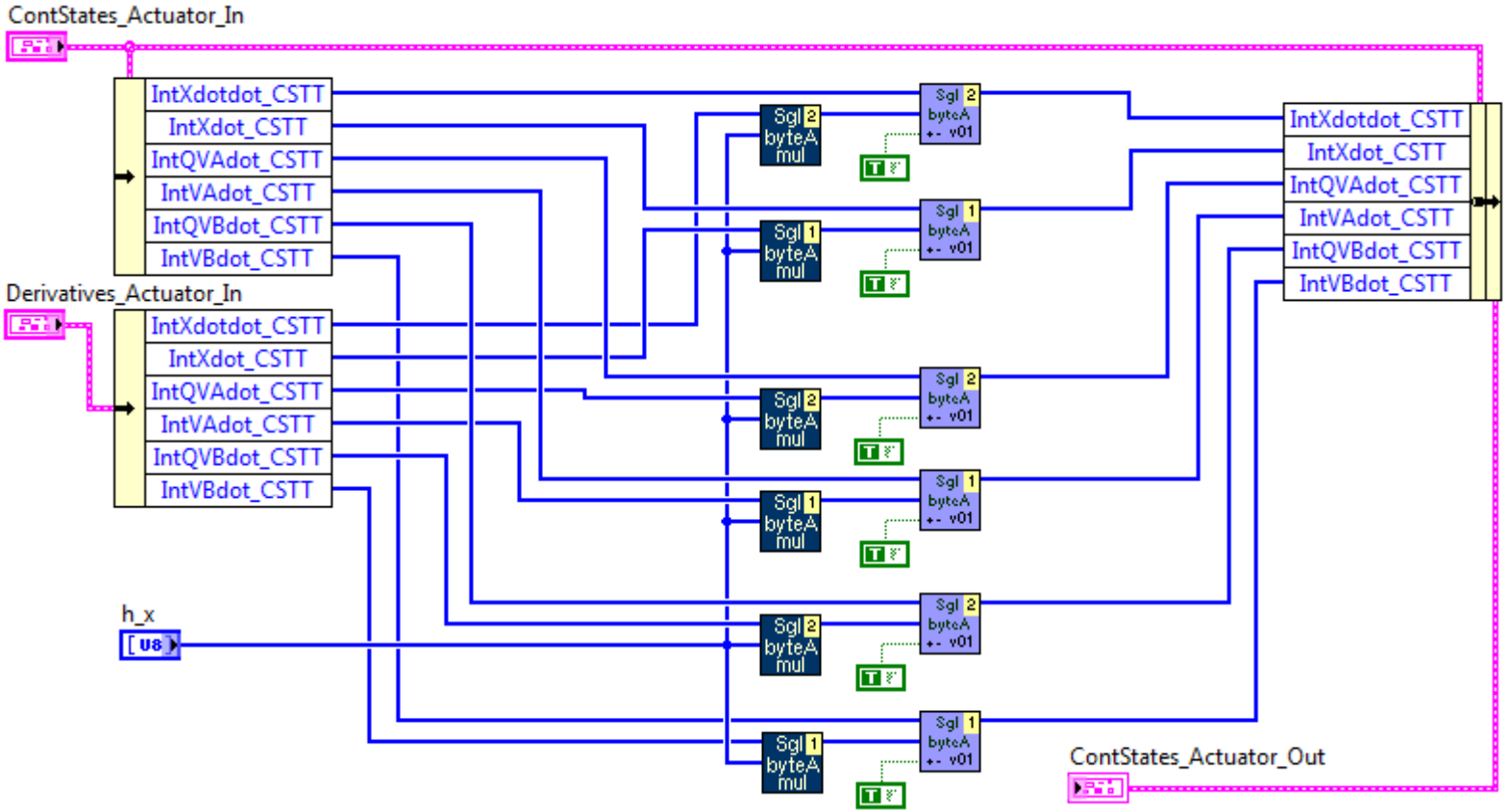


FIGURA 6.15 – Exemplo de alocação manual de instâncias.

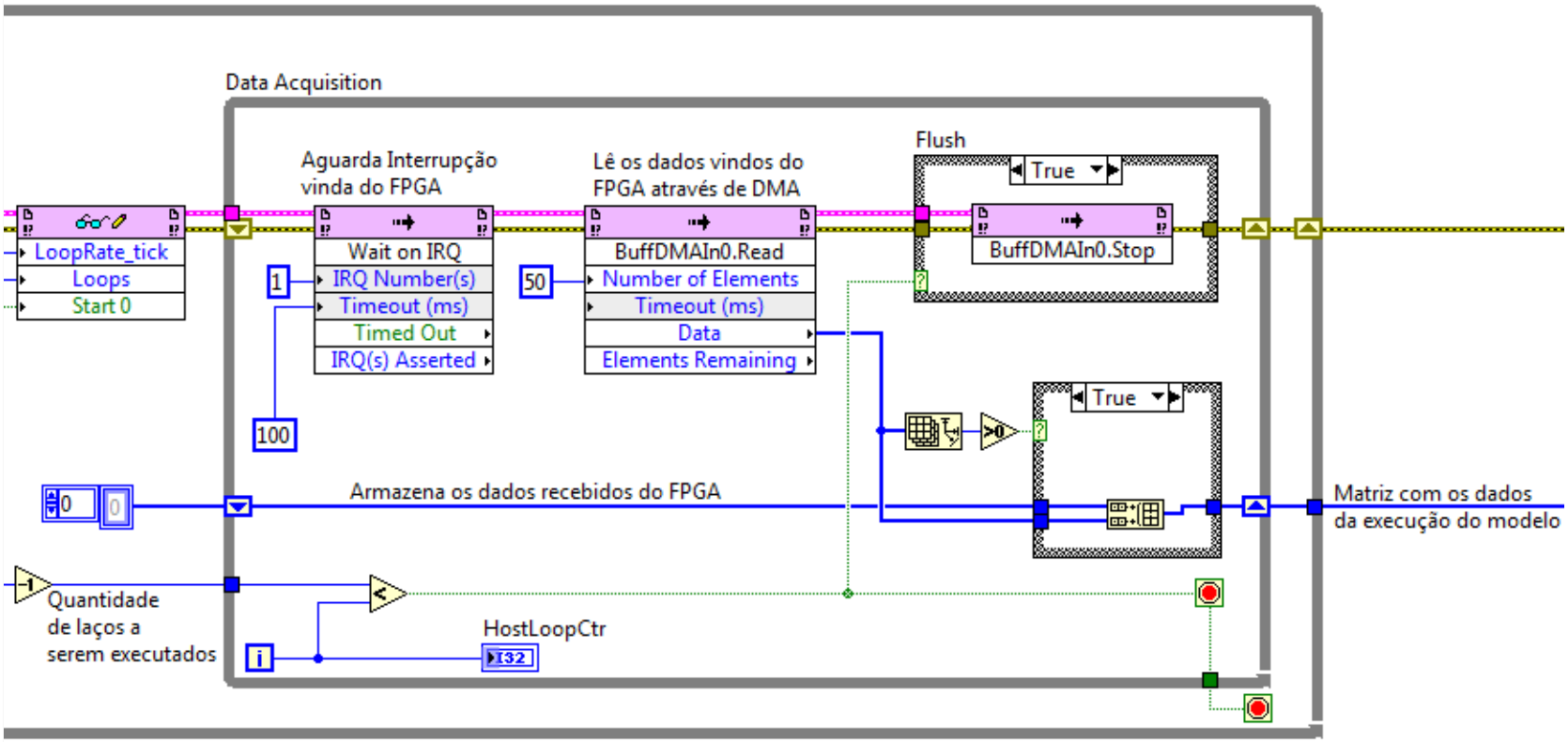


FIGURA 6.16 - *Driver VI*: recebimento dos dados vindos do FPGA.

6.4.3 Resultados do Ambiente de Emulação

As Figuras 6.17 a 6.22 mostram a resposta do sistema a um degrau de comando de posição do êmbolo do atuador, com a mesma amplitude utilizada para analisar a resposta em tempo virtual (Seção 6.3). Em todas as figuras estão apresentadas, para efeitos de comparação, as respostas tanto em tempo virtual como em tempo real.

Nas Figuras 6.17 e 6.18 pode-se observar a resposta da posição do êmbolo do atuador, e nas Figuras 6.19 e 6.20 está apresentado o comportamento da vazão Q_A entre a servoválvula e a câmara A do atuador. Finalmente, nas Figuras 6.21 e 6.22 está apresentado o comportamento da pressão hidráulica P_A na câmara A do atuador.

No que tange à resposta de posição do atuador, bem como o fluxo Q_A, as respostas nos ambientes de tempo virtual e real apresentaram resultados bastante parecidos. Apenas com relação à pressão P_A, observa-se um nível de oscilação maior na resposta em tempo real que em tempo virtual, mas ainda assim dentro de valores aceitáveis.

É importante repetir que, na definição dos parâmetros do sistema modelado neste trabalho, foram adotadas simplificações – considerar vazamento nulo no selo do êmbolo, coeficiente de atrito e constante de amortecimento nulos – que dificultam a obtenção de uma resposta estável e amortecida.

Deve-se mencionar ainda que, devido às limitações de tempo e espaço no FPGA, apenas os valores referentes ao fluxo e pressão do lado A do atuador foram gravados; assume-se que o comportamento do lado B seja similar.

Em função da introdução de um pequeno atraso na implementação em LabVIEW FPGA devido ao "pipelining", foram acrescentados atrasos equivalentes no modelo em Simulink, mas não houve alterações na resposta do sistema em tempo virtual. Os valores aqui apresentados não incluem estes atrasos nas respostas.

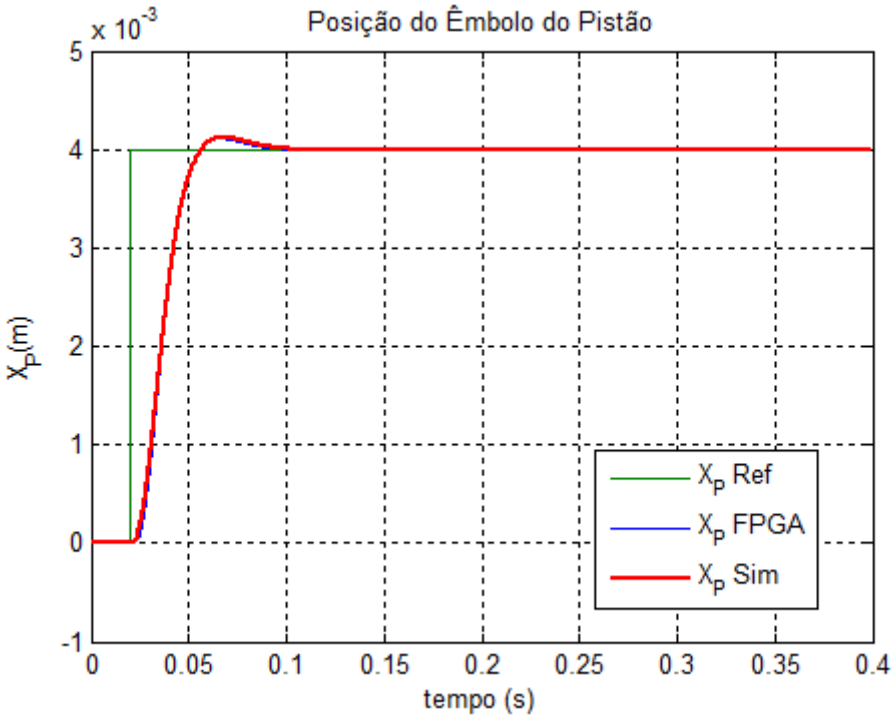


FIGURA 6.17 – Posição X_P do êmbolo do atuador.

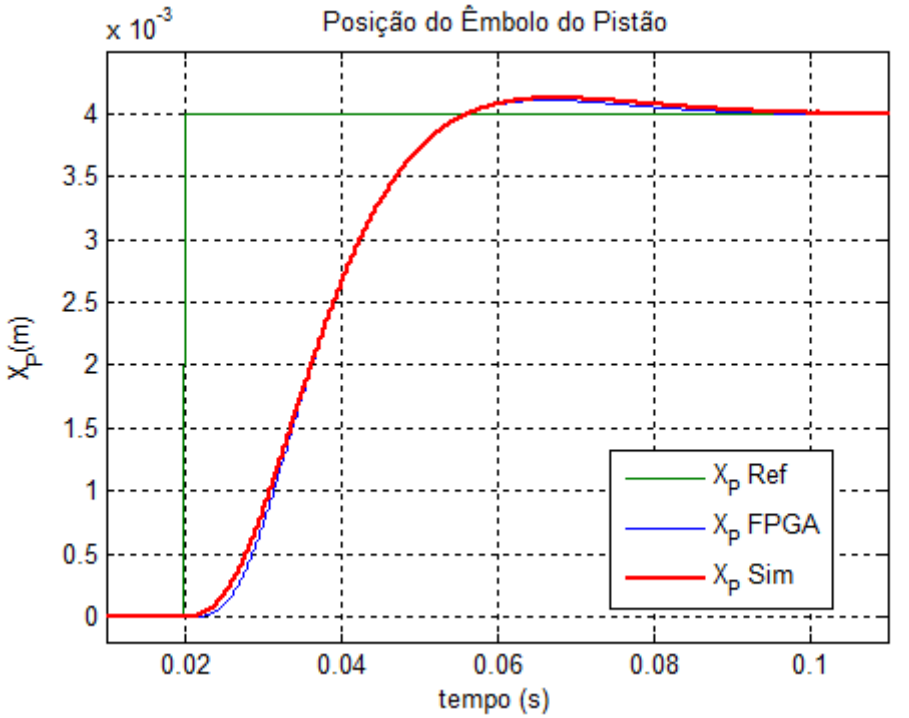


FIGURA 6.18 – Posição X_P do êmbolo do atuador, detalhe da Figura 6.17.

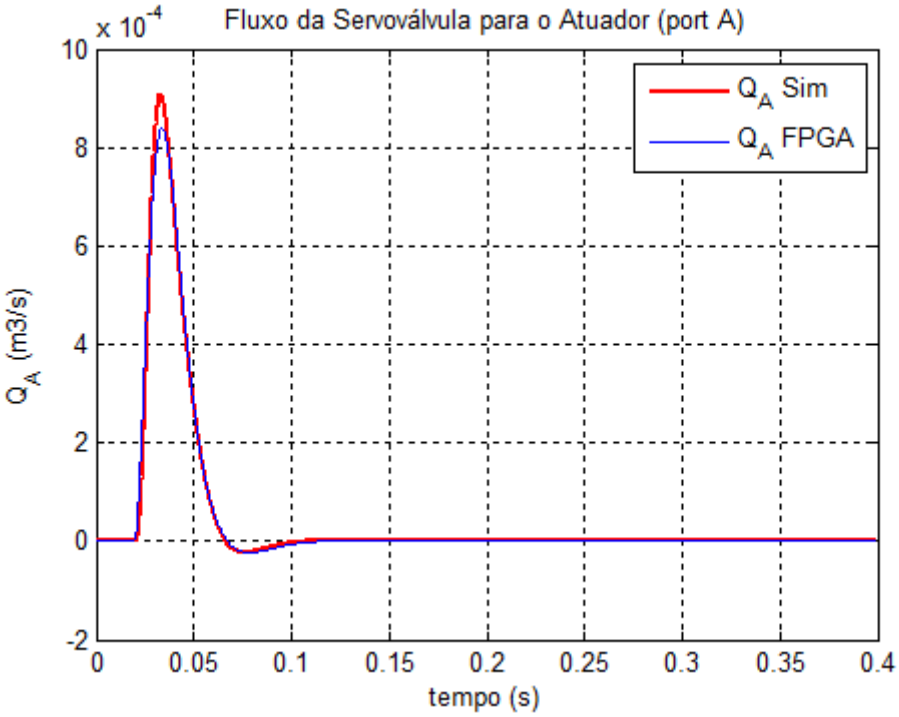


FIGURA 6.19 – Fluxo da servoválvula para o atuador, port A.

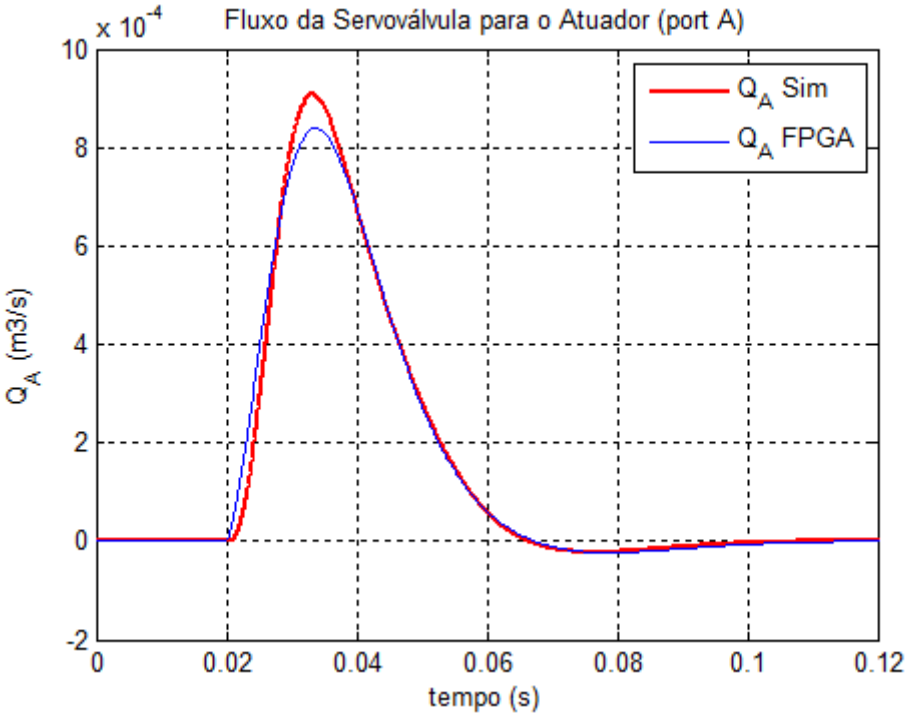


FIGURA 6.20 – Fluxo da servoválvula para o atuador, port A, detalhe da Figura 6.19.

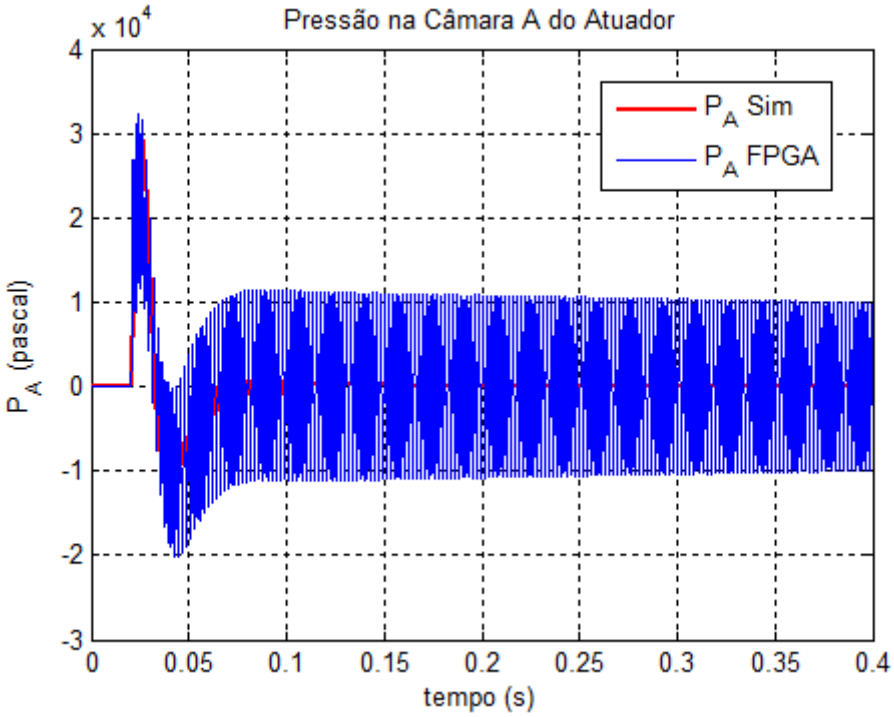


FIGURA 6.21 – Pressão na câmara A do atuador.

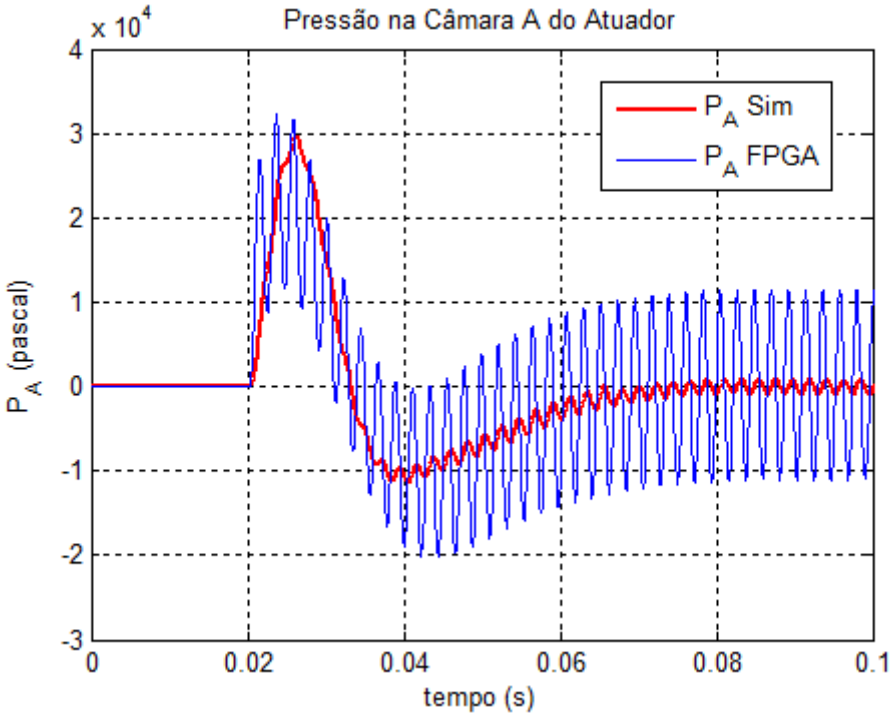


FIGURA 6.22 – Pressão na câmara A do atuador, detalhe da Figura 6.21.

CAPÍTULO 7

CONCLUSÕES

Foram apresentados diferentes fluxos de desenvolvimento, compreendendo desde a formulação do problema, até a execução de uma parte por um FPGA, permitindo avaliar que ainda não existe um processo de geração automática similar à existente para códigos embarcados em processadores tradicionais.

Apesar da quantidade impressionante de recursos das diversas ferramentas envolvidas nos processos de desenvolvimento, a questão do aproveitamento adequado dos recursos e características de um FPGA, de uma forma completamente automática, ainda permanece como um desafio a ser atingido.

O fluxo de desenvolvimento adotado, apesar da parcela de intervenção manual inerente, apresentou resultados satisfatórios comparando os resultados obtidos nos ambientes com características de tempo real com os obtidos em tempo virtual. Em todas as respostas, os resultados foram praticamente idênticos. Foi possível experimentar pelo menos uma das características mais marcantes do FPGA, o paralelismo.

Ao traduzir o modelo no padrão Simulink para o LabVIEW, pôde-se notar que, se o algoritmo está bem definido como no caso do controlador PID, o esforço dispendido é relativamente baixo. No caso, deve-se entender como bem definido aquele algoritmo escrito em um formato próximo ao da sua implementação final.

Já a tarefa de traduzir um algoritmo como o de integração, onde implicitamente está a execução de uma parte substancial do modelo, pode tornar-se bastante complexa, dependendo das configurações utilizadas na geração do código intermediário C, e da própria complexidade do modelo a ser traduzido.

Com relação aos Ambientes de co-simulação, o Padrão mostrou-se mais flexível para o processo de desenvolvimento e validação do sistema. Por executar parte do modelo em Windows, ele possui limitações óbvias quanto à capacidade de atendimento de requisitos de tempo real. De fato, é estimado que se for necessário executar um modelo com passo de amostragem abaixo de 10ms, o resultado não será confiável, em termos do seu determinismo.

O Ambiente Proprietário de co-simulação mostrou-se bastante robusto para execução em tempo real, com recursos adequados de monitoramento "on-line" da resposta do sistema, mas um pouco mais limitado quanto ao processo de desenvolvimento em si. Em suma, combinando os pontos positivos de ambos os ambientes, a preparação e validação dos modelos pode ser realizada no Ambiente Padrão, e a execução final no Ambiente Proprietário.

Logicamente, as considerações feitas levam em conta um modelo que é executado parcialmente em um FPGA comunicando com um processador tradicional, que por sua vez executa o resto do modelo. Se o modelo inteiro foi embarcado no FPGA, essas considerações passam a ser irrelevantes, sendo a capacidade do FPGA o único fator de comparação.

A escolha do algoritmo de controle PID, auxiliado pelo critério ITAE para o ajuste de seus parâmetros, mostrou-se uma alternativa válida com resultados expressivos do ponto de vista de tempo de resposta e sobreelevação. Trabalhos futuros podem explorar melhor métodos como o ITAE, na obtenção de algoritmos adaptativos usando a resposta do sistema como critério de ajuste dos parâmetros, a serem executados concomitantemente ao controlador, e não apenas como critério de projeto.

Quanto ao caso aeronáutico, o desafio de embarcar todo um conjunto atuador, servoválvula e controlador de posição foi atingido com resultados bastante promissores. Este é o tipo da aplicação onde equipamentos com FPGAs mais

poderosos podem ajudar a incorporar mecanismos adicionais de registro de dados internos, algoritmos mais sofisticados, dentre outros.

Ainda sobre o caso aeronáutico, é importante mencionar que, através dele, foi possível identificar e exercitar duas das características mais importantes do FPGA – paralelismo e "pipelining". A capacidade de reordenar uma arquitetura digital para cumprir uma determinada tarefa, ou a própria discussão a respeito de alternativas de arquitetura, estavam tornando-se cada vez mais desnecessárias em função na enorme popularidade no uso de processadores tradicionais. O uso de FPGAs resgata essa discussão sobre alternativas de arquiteturas digitais para implementação de algoritmos. Há um enorme futuro a se desbravar sobre alternativas de arquitetura e capacidade de reconfiguração – a terceira característica marcante dos FPGAs.

Com relação a trabalhos futuros, várias áreas também podem ser abordadas:

- Diversas alternativas de fluxo de desenvolvimento foram apresentadas neste trabalho, mas apenas um deles pôde ser explorado. Este é um terreno fértil para experimentação, seja pelo número de abordagens de desenvolvimento possíveis, seja pelas alternativas disponibilizadas pelos fabricantes de FPGA.

- Explorar a utilização do fluxo de desenvolvimento proposto a outras aplicações, em especial o aeronáutico, seja com o objetivo de obter um modelo executável em laboratório, seja para obter um processo de desenvolvimento qualificável segundo normas aeronáuticas. Na verdade, nesse último objetivo, a aplicabilidade pode ser tanto para o meio aeronáutico, como o espacial, guardadas as diferenças entre as alternativas possíveis dos fabricantes de FPGA.

- Ainda como conseqüência dessa tarefa, uma área adicional de estudo refere-se aos processos de verificação e validação necessários e compatíveis com os

métodos de desenvolvimento disponíveis, para demonstrar observância às normas DO-178B, nível A e DO-254.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDRIOLI Jr., R.L. **Modelagem e análise dos efeitos da rigidez estrutural na estabilidade de um sistema eletro-hidráulico de comando de profundor**. São José dos Campos. 103p. Dissertação (Mestrado em Engenharia Aeronáutica e Mecânica) - Instituto de Tecnologia de Aeronáutica, 2005.

ARANTES Jr., G. **Estudo comparativo de técnicas de controle de atitude em três eixos para satélites artificiais**. São José dos Campos. 201p. (INPE-12970-TDI/1018). Dissertação (Mestrado em Engenharia e Tecnologia Espacial) - Instituto Nacional de Pesquisas Espaciais, 2005.

ASTROM, K. J.; HAGGLUND, T. The future of PID control. **Control Engineering Practice**, v. 9, n. 11, p. 1163-1175, nov. 2001.

BANERJEE, P. An overview of a compiler for mapping MATLAB programs onto FPGAs. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE ASP-DAC, 2003, [S.l]. **Proceedings…** [S.l]: IEEE, 2003.

BANERJEE, P.; et al. Overview of a compiler for synthesizing MATLAB programs onto FPGAs. **IEEE Transactions on Very Large Scale Integration Systems**, v. 12, n. 3, p. 312, mar. 2004.

BHATT, T. M.; McCAIN, D. Matlab as a development environment for FPGA design. In: Anual ACM IEEE Design Automation Conference. **Proceedings…** Anaheim, California: Association of Computing Machinery (ACM), 2005. v., p. 607-610.

CHATZAKOS, P.; PAPADOPOULOS, E. **On model-based control of hydraulic actuators**. Proceedings of RAAD'03, 12th International Workshop on Robotics, maio 2003.

CHU, P. P. **RTL hardware design using VHDL**. Hoboken, New Jersey: John Wiley and Sons, 2006. 696 p.

COMINOS, P.; MUNRO, N. PID controllers: recent tuning methods and design to specification. **IEE Proc. Control Theory Appl.**, v. 149, n. 1, p. 46-53, jan. 2002.

DEMARCHI, F. L. **Modeling and identification of a fly-by-wire control system**. São José dos Campos. 119p. Dissertação (Mestrado em Engenharia Mecânica e Aeronáutica) - Instituto de Tecnologia de Aeronáutica, 2005.

DESCHAMPS, J. P.; BIOUL, G. J. A.; SUTTER, G. D. **Synthesis of arithmetic circuits**. Hoboken, New Jersey: John Wiley and Sons, 2006. 578 p.

FIELDING, C. The design of fly-by-wire flight control systems. **Proceedings…** Warton Aerodrome, Preston, UK: BAE Systems, Aerodynamics, 2000.

GENEST, G.; CHAMBERLAIN, R.; BRUCE, R. Programming a FPGA-based super computer using a C-to-VHDL compiler. **Second NASA/EASA Conference on Adaptive Hardware and Systems**, 2007.

GOBATO, M. F.; **Controles monovariáveis aplicados a sistemas aeroespaciais fracamente ou fortemente acoplados**. São José dos Campos. 388p. (INPE-14494-TDI/1175). Dissertação (Mestrado em Engenharia e Tecnologia Espaciais) - Instituto Nacional de Pesquisas Espaciais, 2007.

ISERMAN, R.; **Digital control Systems**. Berlin: Springer-Verlag, 1989. 334 p.

KRUKOWSI, A.; KALE, I. Simulink/Matlab-to-VHDL route for full-custom/FPGA rapid prototyping of DSP algorithms. In: **Matlab DSP Conference**, 1999.

KUGA, H. K.; RAO, K. R. **Introdução à mecânica orbital**. São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 1995. 66 p. (INPE-5615-PUD/064)

LALL, N.; CIGAN, E. Plug and play design methodologies for FPGA-based signal processing. **FPGA Journal**., mar. 2005. Disponível em: <www.fpgajournal.com/articles_2005/20050308_xilinx.htm>. Acesso em 21 dezembro 2008.

MEIROVITCH, L. **Methods of analytical dynamics**. USA: McGraw-Hill, Inc, 1970. 540 p.

MERRIT, H. E. **Hydraulic control systems**. New York: John Wiley, Inc, 1967. p.

MOOG INDUSTRIAL CONTROLS DIVISION. **Electrohydraulic valves. A technical look**. East Aurora, NY, 2007.

MOREIRA, M. L. B. **Projeto e simulação de um controle discreto para a plataforma multi-missão e sua migração para um sistema operacional de tempo real**. São José dos Campos. 181p. (INPE-14202-TDI/1103). Dissertação (Mestrado em Engenharia e Tecnologia Espacia) - Instituto Nacional de Pesquisas Espaciais, 2006.

NATIONAL INSTRUMENTS. **LabVIEW FPGA help – pipelining to optimize FPGA VIs**. [S.l.], 2008.

NATIONAL INSTRUMENTS. **cRIO Backplane**. Disponível em http://www.ni.com/pdf/products/us/cat_crio_9104.pdf. Acesso em 05 fev 2009.

NATIONAL INSTRUMENTS. **Understanding parallel hardware: multiprocessors, hyperthreading, dual-core, multicore and FPGAs**. www.ni.com, 2006.

OGATA, K. **Engenharia de controle moderno**. Rio de Janeiro - RJ: Prentice/Hall do Brasil, 1982. 929 p.

OPPENHEIM, A. V.; SCHAFER, R. W. **Discrete-time signal processing**. Upper Saddle River, New Jersey: Prentice-Hall, Inc, 1989. 879 p.

POLEY, R. **DSP control of electro-hydraulic servo actuators**. EUA: Texas Instruments, 2005. 26 p. (SPRAA76 - Application Report)

SHANBLATT, M. A.; FOULDS, B. A. Simulink-to-FPGA implementation tool for enhanced design flow. In: IEEE INTERNATIONAL CONFERENCE ON MICROELECTRONICS SYSTEMS EDUCATION, 2005, Anaheim, CA. **Proceedings...** Anaheim: IEEE, 2005.

SOUZA, M. L. O. **Estudo e desenvolvimento de um sistema de controle de atitude ativo em três eixos para satélites artificiais usando atuadores pneumáticos a gás frio e volantes de reação**. São José dos Campos. 368p. (INPE-2000-TDL/042). Dissertação (Mestrado em Ciência Espacial) - Instituto Nacional de Pesquisas Espaciais, 1980.

STEVENS, B. L.; LEWIS, F. L. **Aircraft control and simulation**. Hoboken, New Jersey: John Wiley and Sons, 2003. 664 p.

VANEVENHOVEN, T. Generate FPGA designs from M-code. **DSP Design Line**. , n. , p. , Nov. 2007 Disponível em: <www.dspdesignline.com/203100443>. Acesso em 01 janeiro 2009.

WAIN, R.; BUSH, I.; GUEST, M.; DEEGAN, M.; KOZIN, I.; KITCHEN, C. **An overview of FPGA and FPGA programming; initial experiences at Daresbury**. Daresbury, Warrington: CCLRC Daresbury Laboratory, 2006. 24 p. (DL-TR-2006-010)

WERTZ, J. R. **Spacecraft attitude determination and control**. London, England: D. Reideil Publishing Company, 1978. 861 p.

WIKIPEDIA. **Enciclopédia livre**. http://en.wikipedia.org/wiki, 21 dez. 2008.

YABARRENA, J. M. S. C. **Tecnologias system on chip e CAN em sistemas de controle distribuído**. São Carlos. 124p. (). Dissertação (Mestrado em Engenharia Mecânica) - Universidade de São Paulo, 2006.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo