Carolina Amant Gabiatti

Desenvolvimento da Lógica de um Equipamento de Testes de Capacetes com Display Integrado

Porto Alegre

2014

Carolina Amant Gabiatti

Desenvolvimento da Lógica de um Equipamento de Testes de Capacetes com Display Integrado

Projeto de Diplomação submetido ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, sendo requisito para a conclusão do curso de Graduação em Engenharia Elétrica.

Universidade Federal do Rio Grande do Sul - UFRGS

Escola de Engenharia

Departamento de Engenharia Elétrica

Orientador: Carlos Eduardo Pereira

Porto Alegre

2014

CIP - Catalogação na Publicação

Amant Gabiatti, Carolina
Desenvolvimento da Lógica de um Equipamento de Testes de Capacetes com Display Integrado / Carolina Amant Gabiatti. -- 2014.
92 f.

Orientador: Carlos Eduardo Pereira.

Trabalho de conclusão de curso (Graduação) -- Universidade Federal do Rio Grande do Sul, Escola de Engenharia, Curso de Engenharia Elétrica, Porto Alegre, BR-RS, 2014.

1. FPGA. 2. Equipamento de Testes. 3. Helmet-Mounted Display. 4. VHDL. I. Pereira, Carlos Eduardo, orient. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da UFRGS com os dados fornecidos pelo(a) autor(a).

Carolina Amant Gabiatti

Desenvolvimento da Lógica de um Equipamento de Testes de Capacetes com Display Integrado

Projeto de Diplomação submetido ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, sendo requisito para a conclusão do curso de Graduação em Engenharia Elétrica.

Trabalho aprovado. Porto Alegre, 6 de julho de 2014:

Carlos Eduardo Pereira
Doutor em Engenharia Elétrica pela
Universidade de Stuttgart - Alemanha
(Orientador)

Giovani Mateus Vizzotto
Engenheiro Eletricista pela Universidade
Federal do Rio Grande do Sul - Brasil

Marcelo Götz
Doutor em Informática pela Universidade de
Paderborn - Alemanha

Porto Alegre
2014

Resumo

Este trabalho descreve o projeto e o desenvolvimento de parte da lógica programável de um equipamento de teste de capacetes com *display* integrado (em inglês *Helmet Mounted Display*, ou HMD). Este equipamento deve testar as diversas funcionalidades dos HMDs, como a geração de imagens e a comunicação serial. Além disso, a lógica do equipamento deve ser responsável por controlar o sequenciamento das suas fontes de tensão elétrica e realizar funções de autoteste, como a medida do consumo das fontes de tensão elétrica e a verificação da integridade dos canais de comunicação da placa. A partir dos requisitos dados e do estudo do *hardware* da placa, desenvolvida por um cliente, foi feito o projeto da lógica programável, posteriormente desenvolvida em VHDL. Neste trabalho, é descrita a arquitetura dos blocos de controle dos *clocks*, de aquisição de tensões e correntes da placa, de aquisição de entradas discretas, de sequenciamento das tensões elétricas, de autoteste e de geração de padrões *stroke*. Os blocos desenvolvidos foram testados através de simulações computacionais. A lógica implementada será gravada no FPGA Artix 7, da Xilinx, para o qual apresentou resultados satisfatórios em relação à ocupação de recursos e à temporização.

Palavras-chaves: capacete com *display* integrado. lógica programável. VHDL.

Abstract

This paper describes the design and development of the programmable logic of a test equipment for Helmet Mounted Displays, or HMDs. This equipment shall test many of the features of HMDs, such as image generation and serial communication. Besides, the logic of the equipment shall be responsible for controlling the sequencing of its voltage sources and performing self-test functions, such as measurement of the power consumption of electrical voltage sources and verification of the integrity of the board communication channels. Based on the requirements and the study of the hardware, developed by a client, the programmable logic was designed and further developed in VHDL. In this paper, it is described the architecture of the clocks control, voltages and currents acquisition, the discrete inputs acquisition, the voltages sequencing, the self-test and the stroke patterns generation blocks. The developed blocks were tested through computer simulations. The logic implemented will be programmed in the Xilinx FPGA Artix 7, for which it presents satisfactory results concerning resources occupation and timing.

Key-words: helmet mounted display. programmable logic. VHDL.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ADC	*Analog to Digital Converter*
ADDR	*Address*
AMBA	*Advanced Microcontroller Bus Architecture*
AXI	*Advanced eXtensible Interface*
BIT	*Built-In Test*
BRAM	*Block RAM*
CLK	*Clock*
CMD	*Command*
CMT	*Clock Management Tile*
CRC	*Cyclic Redundancy Check*
CRT	*Cathode Ray Tube*
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection*
DAC	*Digital to Analog Converter*
DASH	*Display And Sight Helmet*
DCO	*Digitally Controlled Oscillator*
DDR	*Dual Data Rate*
DEMUX	Demultiplexador
EN	*Enable*
ETH	*Ethernet*
FCS	*Frame Check Sequence*
FIFO	*First-In First-Out*
FLIR	*Forward Looking Infrared*
FPGA	*Field-Programmable Gate Array*

GEN	*Generator/Generated*
GMII	*Gigabit Media Independent Interface*
HMD	*Helmet Mounted Display*
ID	*IDentifier*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
ILS	*Instrument Landing System*
INT	Interrupção
IOB	*Input/Output Block*
JHMCS	*Joint Helmet Mounted Cueing System*
LAN	*Local Area Network*
LFSR	*Linear-Feedback Shift Register*
LUT	*Look-Up Table*
MAC	*Media Access Control*
MAN	*Metropolitan Area Network*
MDC	*Management Data Clock*
MDIO	*Management Data Input/Output*
MII	*Media Independent Interface*
MISO	*Master-In Slave-Out*
MOSI	*Master-Out Slave-In*
MUX	Multiplexador
OSI	*Open Systems Interconnection*
PC	*Personnal Computer*
PCB	*Printed Circuit Board*
PLL	*Phase-Locked Loop*
PRBS	*Pseudo-Random Binary Sequence*
RAM	*Random-Access Memory*

RE	*Read Enable*
REG	Registrador(es)
RS	*Reconciliation Sublayer*
SDB	*Shut-Down Bit*
SEL	Seletor
SFD	*Start Frame Delimiter*
SPI	*Serial Peripheral Interface*
STA	*STAtion management entity*
TCAS	*Traffic alert and Collision Avoidance System*
TCL	*Tool Command Language*
UART	*Universal Asynchronous Receiver-Transmitter*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
VSYNC	*Vertical SYNChronization*
WEN	*Write ENable*
XDC	*XFA Device Control*
XFA	*XML Forms Architecture*
XML	*eXtensible Markup Language*
XOR	*eXclusive OR*

Sumário

1 Introdução

1.1 Motivações e Contextualização

Uma das principais preocupações do setor da aviação, tanto civil quanto militar, é a segurança. Ao longo dos últimos anos, diversos sistemas foram incluídos nas aeronaves para auxiliar a tripulação em suas operações. O ILS (*Instrument Landing System*, ou Sistema de Pouso por Instrumentos), por exemplo, pode operar juntamente com o sistema de piloto automático para realizar a aproximação e o pouso da aeronave de forma automática (WYATT; TOOLEY, 2013). Já o TCAS (*Traffic alert and Collision Avoidance System*, ou Sistema Anticolisão e de Alerta de Tráfico) monitora a distância e a altitude de aeronaves vizinhas e alerta a tripulação sobre riscos de colisão, sugerindo desvios verticalmente complementares para as duas aeronaves (SPITZER, 2006).

No contexto destes sistemas aviônicos de auxílio à tripulação e monitoramento das variáveis de ambiente da aeronave, foi desenvolvido o equipamento chamado de *Helmet Mounted Display*, ou Capacete com *Display* Integrado. Estes dispositivos apresentam em seus visores as informações primárias da aeronave, como a altitude, a velocidade e o vetor da trajetória de voo, superpostas à visão natural do piloto. Estas informações são projetadas no campo de visão de longa distância, de maneira que o piloto não precise mudar seu foco para vizualizá-las. Além disso, em aeronaves equipadas com sensores FLIR (sensores infravermelho com visão frontal), é possível substituir a imagem que o piloto estaria vendo pela imagem gerada pelos sensores. Desta maneira, o piloto estaria habilitado a operar em condições ambientais adversas, como na presença de neblina ou durante a noite, e operações de reconhecimento e patrulha se tornariam significativamente mais eficientes (COLLINSON, 2011).

Pela eficiência e pela praticidade apresentadas pelos HMDs, eles estão sendo utilizados como *displays* primários pelos pilotos, assumindo papel essencial na segurança da aeronave. Consequentemente, os HMDs são sistemas com nível de confiabilidade extremamante crítico, pois a ocorrência de falhas pode colocar em risco a operação da aeronave e até mesmo a vida da tripulação e dos passageiros. Atualmente, os sistemas para a vizualização das informações primárias das aeronaves devem assegurar que a probabilidade de apresentar informações errôneas ou incertas seja de, no máximo, 10^{-9} por hora de voo (COLLINSON, 2011). Assim, é primordial que estes equipamentos sejam completamente testados antes de serem utilizados em voos, e que os testes englobem todas as funcionalidades do sistema.

1.2 Objetivos do Projeto

O objetivo deste trabalho é projetar e desenvolver, em uma linguagem de descrição de *hardware* (neste caso, VHDL), a lógica programável de um equipamento de testes de HMDs. Para isso, inicialmente, será estudado o circuito deste equipamento de testes, que foi desenvolvido por um cliente da empresa onde o projeto será realizado, e serão analisadas as especificações de funcionamento e restrições da lógica, fornecidas pelo mesmo cliente.

O equipamento deve testar as funcionalidades do HMD, como a geração de imagens no *display* e a comunicação serial. A lógica também deve gerenciar os modos de operação da placa e o sequenciamento das tensões elétricas de alimentação. O equipamento também deve ser capaz de realizar funções de autoteste, checando o consumo das suas fontes de alimentação e testando seus canais de comunicação.

Primeiramente, são apresentados os referenciais teóricos deste trabalho. Nesse capítulo, são estudados alguns conceitos utilizados ao longo do trabalho e são apresentados os principais protocolos de comunicação utilizados no projeto, indicando-se referências para que o leitor possa buscar mais informações sobre os assuntos.

Na segunda parte do trabalho, inicialmente, são apresentados o sistema do equipamento de testes de forma geral e os HMDs a serem testados. Em seguida, são analisados os requisitos técnicos do projeto, a arquitetura do *hardware* da placa, com suas características e limitações, e o FPGA que será utilizado. A partir destas informações, é possível saber as funções que devem ser executadas pela lógica e os recursos disponíveis para isto, dados importantes para a compreensão da elaboração e do desenvolvimento do projeto.

Então, na terceira parte, é apresentada a arquitetura da lógica programável desenvolvida, através de diagramas de blocos e descrições detalhadas. Também é apresentado um exemplo de simulação de um bloco feito em VHDL e o *script* que gera o arquivo de gravação. Na última parte do relatório, são apresentados e analisados os resultados obtidos nas simulações e na síntese. Nesta parte, ainda são discutidas as próximas etapas do projeto e possibilidades de melhoria. Por fim, as conclusões sobre o projeto são apresentadas.

2 Referenciais Teóricos

2.1 Componentes e Conceitos Utilizados

2.1.1 FPGA

FPGA, do inglês *Field Programmable Gate Array*, ou Matriz de Portas Programáveis em Campo, é um tipo de circuito integrado que dispõe internamente de componentes lógicos cujas conexões são configuráveis, permitindo assim que sejam implementadas diversas funções lógicas (ver "*Field-Programmable Gate Array Technology*", Trimberger (1994)). Para implementar tais funções no FPGA, deve-se descrever o *hardware* em uma linguagem apropriada, como VHDL ou Verilog. O código escrito deve ser compilado, sintetizado, devem ser realizadas as operações de *place* (colocação) e *route* (roteamento), e então deve ser gerado o arquivo de gravação para o FPGA.

FPGAs apresentam diversas vantagens em relação à implementação de lógica com diversos circuitos integrados em uma PCB (*Printed Circuit Board*, ou placa de circuito impresso). O uso de FPGAs elimina grande parte do tempo que seria dispensado com o roteamento da placa e diminui consideravelmente a probabilidade de erros de conexão. Além disso, o fato de os FPGAs serem reprogramáveis facilita a correção de eventuais erros de projeto ou mesmo a redefinição do funcionamento da lógica, o que poderia ser inviável se fossem utilizadas outras soluções (WOODS et al., 2008).

2.1.2 DCO

Um Oscilador Digitalmente Controlado (em inglês *Digitally Controlled Oscillator*, ou DCO) é um dispositivo utilizado para a geração de um sinal pulsado com frequência menor do que a do *clock* do sistema, o que muitas vezes é necessário para a sincronização da lógica em um FPGA. DCOs podem ser implementados em VHDL, porém deve-se conhecer a aplicação de destino para escolher o tipo mais adequado.

Um dos tipos de DCOs comumente utilizado é o DCO Contador/Divisor por N, que consiste em um contador de N bits, incrementado a cada borda de subida do *clock*, cujo bit mais significativo é o sinal pulsado desejado. No entanto, este tipo de DCO tem limitações de uso, pois só se pode obter na saída sinais com período múltiplo de 2^N do período da frequência do *clock* do sistema (VALLABHANENIA et al., 2010).

Outro tipo de DCO é o Acumulador, que é mais complexo e gera uma lógica maior do que o DCO Contador/Divisor por N, porém permite a geração de sinais de diversas frequências, eliminando a limitação do DCO tipo Contador/Divisor por N (VALLABHANENIA et al., 2010). Este tipo de DCO consiste em um sinal de N bits, ao qual é somado o valor constante K a cada ciclo de *clock*, sendo que o bit mais significativo deste sinal é a saída do DCO. Assim, a frequência da saída do DCO, f_{DCO}, é dada pela Equação 2.1.

$$f_{DCO} = \frac{K}{2^N} f_{CLK} \tag{2.1}$$

Para se obter uma determinada frequência de saída, é possível utilizar diversas combinações de valores de N e K. Na Tabela 1, é apresentada a relação entre os valores de N e K e a frequência de saída do DCO, visando-se obter 20 KHz, sendo a frequência do *clock* igual a 150 MHz. Normalmente, valores muito baixos de N e K geram sinais com grande desvio em relação à frequência desejada. Neste caso, valores de N menores do que 12 produzem frequência de saída com desvio maior do que 100%. Com N = 12 e K = 1, obtém-se desvio de 83,1%, valor inaceitável para diversas aplicações. Aumentando-se em um bit o tamanho do contador (N = 13), obtém-se desvio aproximadamente dez vezes menor.

Nota-se, portanto, que com o aumento dos valores de N e K, diminui-se o desvio entre a frequência de saída e a frequência desejada. Com N = 36 e K = 9162597, por exemplo, obteria-se um desvio menor do que 0,001 Hz. Porém, valores altos de N e K geram uma lógica muito grande, pois o tamanho do contador é aumentado, comprometendo a performance. Por isso, normalmente são escolhidos os valores de N e K mais baixos que geram uma frequência dentro dos limites aceitáveis para a aplicação de destino. Neste caso, para aplicações onde fosse requerido desvio menor do que 5%, poderia-se escolher K = 16 e N = 9. Para desvio menor do que 1%, poderia-se escolher N = 18 e K = 35.

2.1.3 ODDR

Um ODDR é um componente presente nos FPGAs da Série 7 da Xilinx. Este componente consiste em um registrador de saída dedicado com dupla taxa de dados, ou seja, um registrador que envia para a saída o registro de uma de suas entradas (D1) na borda de subida do *clock*, e o registro de outra de suas entradas (D2) na borda de

Tabela 1 – Relação entre Valores de N e K e a Frequência de Saída de um DCO Acumulador.

N	K	Frequência de Saída (Hz)	Desvio (%)
12	1	36621	83,1
13	1	18311	8,44
14	2	18311	8,44
15	4	18311	8,44
16	9	20599	-2,99
17	17	19455	2,72
18	35	20027	-0,13

Fonte: A Autora.

descida do *clock* (XILINX, 2012). Por ser um registrador de saída, este componente é muito utilizado quando o FPGA gera um sinal de *clock* que deve ser enviado ao circuito externo (placa).

Um ODDR possui quatro entradas de controle: uma entrada de *clock*; uma entrada de *reset*, que coloca a saída em nível lógico baixo quando ativa; uma entrada de *set*, que coloca a saída em nível lógico alto quando ativa; e uma entrada de habilitação do *clock*. O ODDR possui duas entradas de dados de um bit cada, D1 e D2, e uma saída de dados. Existe uma *generic* para definir se D1 e D2 serão amostrados na borda de subida do *clock* ou se D1 será amostrado na borda de subida e D2 na borda de descida.

2.1.4 Metaestabilidade

Registradores (ou flip-flops) em FPGAs possuem requisitos de tempo para funcionarem corretamente. Considerando-se um registrador que amostra sua entrada de dado nas bordas de subida do *clock*, sabe-se que o dado da entrada deve permanecer estável por um tempo mínimo antes de cada borda de subida de *clock*, chamado tempo de *setup* (instalação), e por um tempo mínimo após cada borda de subida de *clock*, chamado de tempo de *hold* (sustentação). A saída do registrador será alterada após um tempo mínimo, chamado *clock-to-output delay* (atraso da saída em relação ao *clock*).

Se o tempo de *setup* ou o tempo de *hold* forem violados, o registrador pode entrar em estado de metaestabilidade. Neste estado, a saída do registrador permanece em um valor entre os níveis lógicos baixo e alto durante algum tempo (maior que o *clock-to-output delay*), podendo assumir, após esse período, tanto o valor da entrada antiga quanto o valor

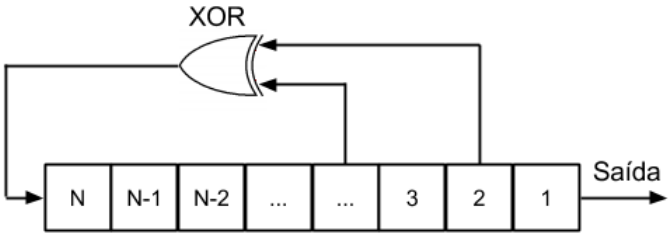
da entrada nova.

A transferência de sinais entre domínios de *clock* dessincronizados ou não relacionados pode causar metaestabilidade. Para evitar este problema, são utilizadas sequências de registradores (normalmente de dois registradores) no domínio de *clock* de destino, fornecendo um tempo adicional para a estabilização de um sinal potencialmente metaestável antes que ele seja utilizado em outros blocos. Estas e outras informações sobre metaestabilidade podem ser encontradas na referência *Understanding Metastability in FPGAs* (ALTERA CORPORATION, 2009).

2.1.5 PRBS

Geradores de Sequências Binárias Pseudo-Aleatórias[1] podem ser implementados utilizando-se Registradores de Deslocamento com Realimentação Linear (ou LFSR, *Linear Feedback Shift-Register*). A implementação de um LFSR pode ser feita utilizando-se N registradores e uma porta XOR (Ou-Exclusivo), como mostra a Figura 1, gerando-se assim uma sequência com período de $2^N - 1$ bits (BADAOUI; FRIGNAC; FEHAM, 2010).

Figura 1 – Gerador de PRBS utilizando LFSR.



Fonte: A Autora.

Geradores de PRBS são bastante utilizados em testes de retorno de canais de comunicação (*loopbacks*). Para isso, substitui-se a saída de um canal por uma sequência gerada por um PRBS, e conecta-se este sinal, preferencialmente no seu ponto de destino, a um canal de entrada. Neste canal de entrada, implementa-se um detector de PRBS, que contém um gerador de PRBS semelhante ao do emissor. Como a sequência é gerada de forma determinística, é possível que o detector compare o sinal recebido a uma sequência conhecida, e quando as duas partes forem semelhantes, sinalize que os sinais estão sincronizados. Após a sincronização, é possível que erros em bits recebidos sejam indicados.

[1] Uma sequência pseudo-aleatória é gerada de forma determinística, mas aparenta ser aleatória.

2.2 Padrões e Protocolos Utilizados

2.2.1 Ethernet

O padrão *Ethernet* define um sistema de conexão para Redes de Área Local (LAN, *Local Area Networks*) tanto em relação à camada Física (PHY, ou *Physical Layer*) quanto em relação à subcamada Controle de Acesso do Meio (MAC, *Media Access Control*), que correspondem aproximadamente à camada Física e à camada Enlace do modelo OSI (*Open Systems Interconnection*, ver Blank (2006)). A *Ethernet* foi padronizada pelo comitê de padrões LAN/MAN, e as informações apresentadas a seguir foram extraídas da norma (IEEE-SA, 2012a).

O IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos) define dois modos de operação para a *Ethernet*: *half duplex*[2] e *full duplex*[3]. Em modo *half duplex*, dois ou mais Equipamentos Terminais de Dados (DTE, ou *Data Terminal Equipment*) compartilham o mesmo meio físico. Para fazê-lo, utiliza-se o padrão CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*, ou Acesso Múltiplo Sensível à Portadora com Detecção de Colisão), definido na norma, para controlar o acesso e detectar colisões de dados. Em modo *full duplex*, apenas dois DTEs são conectados em uma ligação ponto a ponto, não sendo necessário o uso do CSMA/CD.

A interconexão entre a subcamada MAC e a camada física (PHY) pode ser feita pela subcamada de reconciliação (RS, ou *Reconciliation Sublayer*) e pela interface independente do meio (MII, ou *Media Independent Interface*), se a taxa de transmissão de dados for 10 ou 100 Mbps. São transmitidos e recebidos quatro bits de dados simultaneamente, ou seja, a largura dos barramentos de dados é igual a um *nibble*. Assim, a frequência do *clock* é de 2,5 MHz para *Ethernet* 10 Mbps, e 25 MHz para *Ethernet* 100 Mpbs (IEEE-SA, 2012b).

Para a Gigabit *Ethernet*, substitui-se a MII pela GMII (interface gigabit independente do meio, ou *Gigabit Media Independent Interface*). A largura dos barramentos é de oito bits (um *byte*), e a frequência do *clock* utilizado é de 125 MHz para *Ethernet* 1 Gbps (IEEE-SA, 2012c).

Paralelamente à subcamada de reconciliação, existe a Entidade de Gerenciamento da Estação (STA, ou *STAtion management entity*), que através dos sinais MDC (*Management Data Clock*, ou *Clock* dos Dados de Gerenciamento) e MDIO (*Management Data*

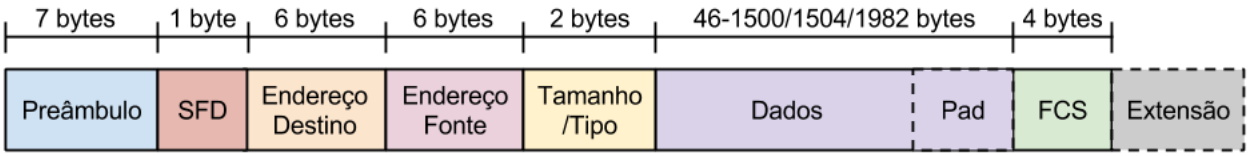
[2] *Half Duplex*: Os terminais podem tanto transmitir quanto receber dados, porém não executar as duas operações simultaneamente.

[3] *Full Duplex*: Os terminais podem transmitir e receber dados simultaneamente.

Input/Output, ou Entrada/Saída de Dados de Gerenciamento), troca dados de controle e *status* com a camada física.

Em termos de cabeamento, a *Ethernet* pode operar sobre cabo coaxial, par trançado ou fibra ótica, por exemplo, sabendo que, para velocidades a partir de 100 Mbit/s, o cabo coaxial não é suportado. Independentemente do suporte físico, o formato dos *frames Ethernet* é padronizado, como mostra a Figura 2. Os octetos são transmitidos da esquerda para a direita, começando pelo preâmbulo, sendo que a transmissão de um octeto inicia pelo seu bit menos significativo e termina pelo seu bit mais significativo (exceto pelo campo FCS, onde a transmissão se inicia pelo bit mais significativo).

Figura 2 – Formato de um *Frame Ethernet*.



Fonte: A Autora.

Os primeiros sete bytes são o preâmbulo, que consiste na sequência binária "10101010". A função do preâmbulo é sincronizar os relógios do transmissor e do receptor. O byte seguinte é chamado de SFD (*Start Frame Delimiter*, ou Demilitador do Início do Quadro), que consiste na sequência binária "10101011", e serve alertar o receptor sobre o início do quadro MAC.

Tanto o campo "Endereço Destino" quanto o campo "Endereço Fonte" tem seis bytes de largura. O primeiro bit transmitido do "Endereço de Destino" indica se o endereço que será enviado é individual (nível lógico baixo) ou de um grupo (nível lógico alto). O segundo bit transmitido indica se o endereço que será enviado é administrado globalmente (nível lógico baixo) ou localmente (nível lógico alto). Assim, o terminal transmissor seleciona o tipo de transmissão a ser executada. Se o primeiro bit transmitido for igual a zero, a transmissão será ponto a ponto (*unicast*), ou seja, destinada a apenas uma estação da rede. Se todos os bits do endereço de destino forem iguais a um, a transmissão será uma difusão generalizada para todos os terminais da LAN (*broadcast*). Se o primeiro bit transmitido for igual a um (exceto no caso de *broadcast*), a transmissão será uma difusão restrita, filtrada pelas placas de rede das estações receptoras (*multicast*).

O campo "Tamanho/Tipo" pode ter dois significados. Se contiver um valor menor ou igual a 1500 em decimal (ou 05DE em hexadecimal), indica o tamanho do campo "Dados". Se o valor for maior ou igual a 1536 em decimal (ou 0600 em hexadecimal), o campo indica a natureza do protocolo da subcamada superior à subcamada MAC.

A largura mínima do campo "Dados" é de 46 octetos, enquanto a largura máxima depende do tipo de quadro que está sendo enviado, podendo ser 1500, 1504 ou 1982 octetos. É importante também que o tamanho mínimo do quadro da dados seja respeitado para garantir o correto funcionamento do algoritmo de detecção de colisão, que só é possível se o primeito bit enviado chegar ao seu destino antes que o último bit a ser enviado saia da fonte. Por isso, se o tamanho dos dados for inferior a 46 octetos, são inseridos octetos chamados de preenchimento (*pad*) para atingir o tamanho mínimo.

O campo FCS, ou Sequência de Verificação do Quadro (do inglês *Frame Check Sequence*) contém 32 bits de CRC (*Cyclic Redundancy Check*, ou Verificação de Redundância Cíclica) do quadro, calculado a partir do campo "Endereço Destino" até o fim do campo "Dados" (ou "Pad", se existir). O procedimento para o cálculo do CRC e o polinômio utilizado estão definidos na descrição do padrão.

Quando opera-se com velocidade de 1000 Mbit/s em modo *half duplex*, mesmo que se respeite o tamanho mínimo do campo de dados, o tempo de duraçao da transmissão não é suficiente para que se garanta o bom funcionamento do algoritmo de detecção de colisão. Para evitar este problema, pode-se adicionar uma extensão, caso o meio físico utilizado permita que se transmitam símbolos facilmente distinguidos dos símbolos de dados.

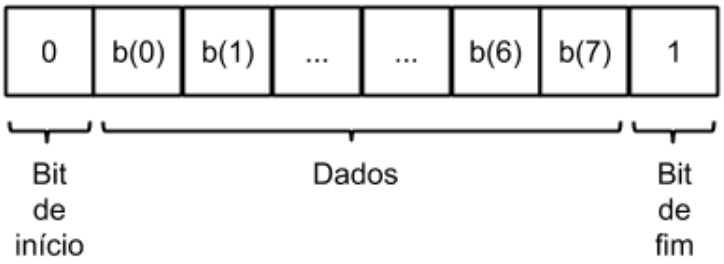
2.2.2 RS-232 e UART

RS-232 é um padrão assíncrono de comunicação serial, normalmente utilizado para a troca de dados entre um PC e um equipamento, que devem possuir uma interface UART (Receptor-Transmissor Assíncrono Universal). Em uma comunicação bidirecional, cada UART deve ter, no mínimo, um receptor, um transmissor e uma referência (*ground*). Neste caso, não há um sinal de sincronismo (como um *clock*, por exemplo), portanto as duas interfaces UART devem estar configuradas para operar na mesma velocidade (TONCICH, 1993).

Enquanto nenhum dado estiver sendo transmitido, o sinal do transmissor deve permanecer em nível lógico alto. Para sinalizar uma transmissão, o transmissor deve enviar nível lógico baixo durante o período de um bit (bit de início, ou *start bit*). Em seguida, o

transmissor envia o dado, iniciando sempre pelo bit menos significativo. No padrão 8-N-1, mostrado na Figura 3, o dado tem largura de oito bits. Como indica o N, em 8-N-1, bits de paridade[4] não são utilizados. Então, após o dado, o transmissor deve enviar nível lógico alto durante o período de pelo menos um bit (bit de fim, ou *stop bit*) antes de poder recomeçar uma nova transmissão (AXELSON, 1998).

Figura 3 – Transferência de Dados no Padrão RS-232.



Fonte: A Autora.

O protocolo RS-232 apresenta limitações em relação ao comprimento do cabo que conecta as duas interfaces UART em função da velocidade de transmissão. Essas limitações devem-se ao fato de que uma transmissão de dados neste padrão é vulnerável a interferências elétricas entre condutores localizados dentro de um mesmo cabo e a atenuações do sinal ao longo do cabo, problemas que se agravam com o aumento da taxa de transmissão e do comprimento do cabo. Originalmente, foi especificada a taxa máxima de transmissão de dados de 20 kbit/s e o comprimento máximo de 15 metros para os cabos (LOXTON; POPE, 1986). Atualmente, a velocidade de transmissão pode chegar a valores bem mais elevados, como 120 kbit/s, e o comprimento do cabo é limitado pela sua capacitância (DALLAS SEMICONDUCTOR, 1998).

2.2.3 CRT e *Stroke*

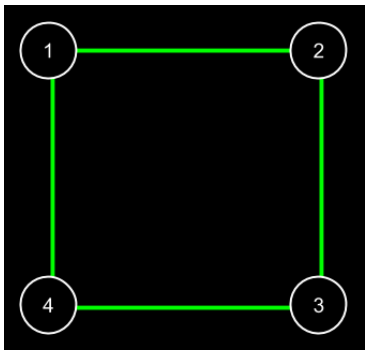
Em um *display* do tipo CRT (*Cathode Ray Tube*, ou Tubo de Raios Catódicos) a imagem apresentada é gerada pela colisão de um feixe de elétrons a alta velocidade com a parede do tubo, revestida de fósforo. Esse fluxo de elétrons é gerado pelo catodo presente no tubo, devido à diferença de potencial elétrico presente dentro do tubo e ao aquecimento do catodo.

[4] Bits de paridade, em transmissões de dados, são utilizados para detecção de erros.

O brilho da imagem pode ser controlado variando-se o potencial elétrico aplicado na grade de controle, elemento responsável por controlar o fluxo de elétrons no tubo, e, consequentemente, a corrente. Existem ainda duas placas de deflexão, uma correspondente ao eixo X e outra ao eixo Y, cujas tensões elétricas aplicadas controlam a posição do foco do feixe de elétrons (WEBSTER, 2010).

A palavra *stroke* pode ser traduzida como traço. Quando se utiliza endereçamento do tipo *stroke*, faz-se um endereçamento ponto a ponto, utilizando-se o traço do feixe de elétrons para formar a imagem desejada. Para fazer um quadrado, como mostrado na Figura 4, por exemplo, é necessário apenas posicionar o cursor no ponto 1, controlando-se as placas de deflexão X e Y, ativar o fluxo de elétrons, deslocar os eixos X e Y para os pontos 2, 3, 4 e, novamente, para o ponto 1.

Figura 4 – Exemplo de Escrita de Vídeo em Formato *Stroke*.



Fonte: A Autora.

2.2.4 Protocolo *Wishbone*

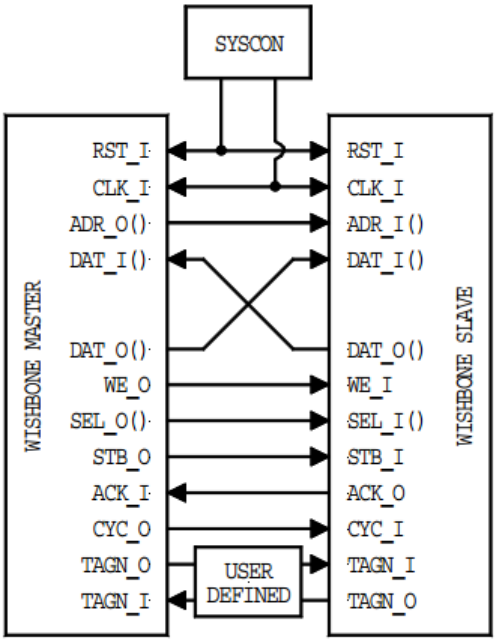
Wishbone é um padrão de transferência de dados utilizado na comunicação entre unidades lógicas. É um protocolo mestre-escravo, ou seja, as operações podem ser inicializadas e finalizadas exclusivamente por uma das partes (o mestre), enquanto que as outras partes (os escravos) apenas respondem às solicitações do mestre. Para que a comunicação seja possível, deve haver um mestre e, no mínimo, um escravo. No protocolo *Wishbone*, a comunicação é efetuada através de leituras e escritas nos registradores dos escravos (COPPOLA et al., 2008).

O protocolo *Wishbone* é muito flexível, apresentando diversas vantagens que o fazem ser bastante utilizado. O tamanho do barramento de dados é configurável, podendo ser 8, 16, 32 ou 64 bits. A decodificação do endereço do escravo é parcial, eliminando

lógica desnecessária e aumentando a velocidade das operações. Também é possível operar com mais de um mestre, o que é essencial em algumas aplicações, como a depuração, por exemplo (OPENCORES; SILICORE, 2002).

Na Figura 5, é apresentada a conexão entre um mestre e um escravo *Wishbone* (*Wishbone Master* e *Wishbone Slave*, respectivamente). A terminação "_I" indica uma entrada de um bloco, enquanto a terminação "_O" indica uma saída de um bloco. Abaixo encontram-se as definições de todas as portas da interface que serão utilizadas, baseadas na especificação do protocolo (OPENCORES; SILICORE, 2002).

Figura 5 – Conexão Padrão do Protocolo *Wishbone*.



Fonte:OpenCores e Silicore (2002, p. 15).

O sinal de *reset* (RST_I) é utilizado para reinicializar a interface *Wishbone*, fazendo todos os sinais receberem seu valor inicial quando o *reset* estiver ativo. Já o *clock* (CLK_I) coordena as operações dos blocos *Wishbone*, fazendo com que sempre estejam sincronizadas entre os blocos.

A interface de endereço (ADR_O no mestre e ADR_I no escravo) serve para o mestre indicar o registrador que será usado em uma operação. Já os sinais de dado são utilizados para que o mestre receba o valor lido após uma operação de leitura de um escravo (DAT_I do mestre e DAT_O do escravo) ou para enviar os dados que o

mestre irá escrever em um registrador do escravo (DAT_O do mestre e DAT_I do escravo).

A habilitação de escrita (WE_O, do inglês *write enable*) indica se o ciclo atual é de leitura (nível lógico baixo) ou de escrita (nível lógico alto). O sinal de seleção (SEL_O do mestre e SEL_I do escravo) indica a localização dos dados válidos em um ciclo de escrita e a localização esperada dos dados válidos em um ciclo de leitura. Seu tamanho depende do tamanho do barramento de dados e da granularidade. No projeto deste trabalho, o barramento de dados é de 32 bits, com granularidade de *byte*, portanto, o sinal de seleção é 4 bits.

O sinal de estrobo (STB_O do mestre e STB_I do escravo) é utilizado para indicar que o escravo foi selecionado, e que os dados presentes na sua interface são válidos. Já o sinal de confirmação (ACK_I do mestre e ACK_O do escravo) é o meio pelo qual o escravo indica o fim de um ciclo normal. O sinal de ciclo (CYC_O do mestre e CYC_I do escravo), quando ativo, indica que um ciclo está ocorrendo. Os sinais TAGN_I e TAGN_O são opcionais e não serão utilizados.

Na Figura 6, pode-se observar um ciclo de leitura *Wishbone*, e na Figura 7 apresenta-se um ciclo de escrita *Wishbone*. A principal diferença entre os dois é a direção da transferência de dados e o sinal de habilitação de escrita, WE_O.

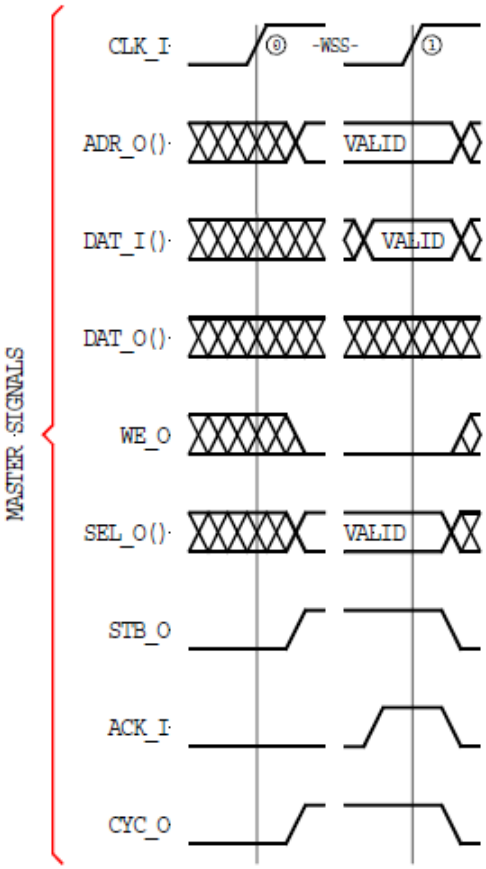
2.2.5 Protocolo SPI

O protocolo SPI (Interface Periférica Serial) permite que se realize transferências de dados entre um mestre e um ou mais escravos de forma bastante simples. Este protocolo síncrono é bastante utilizado em memórias e em conversores, tanto analógico-digital (ADC) quanto digital-analógico (DAC) (CATSOULIS, 2005).

Na Figura 8 observa-se a conexão padrão do protocolo SPI, formada basicamente por quatro sinais. O sinal MOSI é o dado enviado do mestre para o escravo, enquanto o sinal MISO é o dado enviado do escravo para o mestre. SCLK é o sinal de *clock* que coordena os ciclos SPI. O sinal SS é sinal que o mestre utiliza para selecionar um escravo (este sinal é ativo baixo). No escravo, este sinal indica o ciclo de dados (MOTOROLA, 2004).

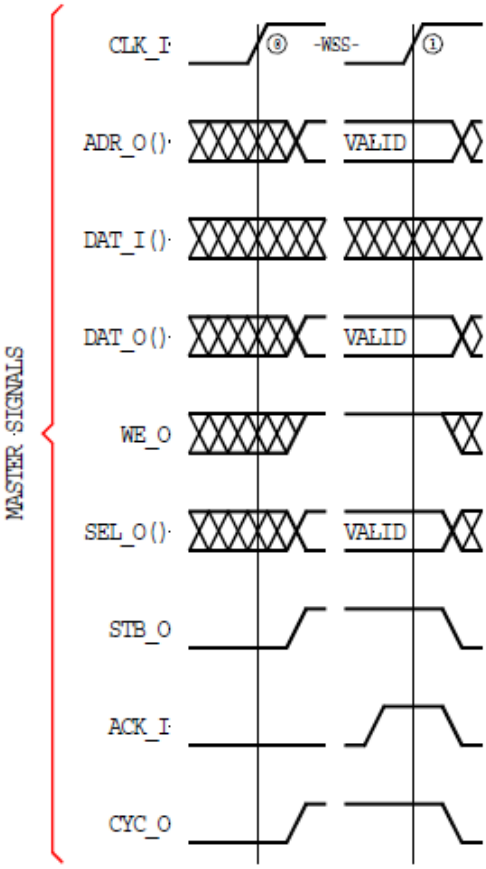
Tanto o mestre quanto os escravos SPI podem ser implementados utilizando-se registradores de deslocamento (conhecidos como *shift registers*), como ilustrado na Figura 8. Tanto o mestre quanto o escravo devem ter o mesmo tamanho de dado (N), armazenado em um registrador. O bit mais significativo do registrador do mestre é conectado ao bit

Figura 6 – Ciclo de Leitura *Wishbone*.



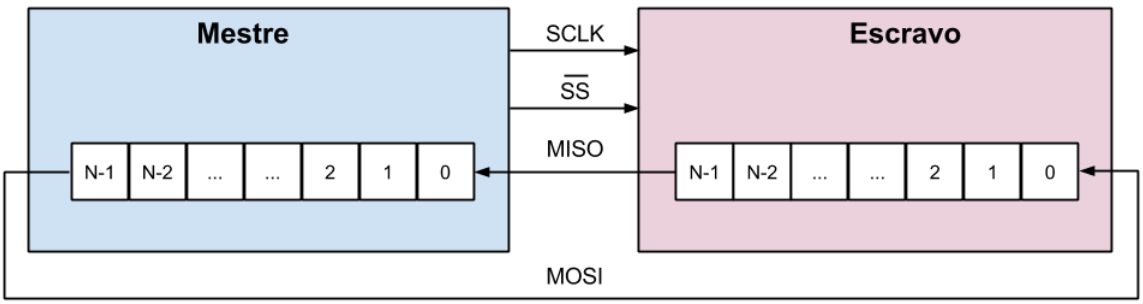
Fonte: OpenCores e Silicore (2002, p. 48).

Figura 7 – Ciclo de Escrita *Wishbone*.



Fonte: OpenCores e Silicore (2002, p. 50).

Figura 8 – Conexão Padrão do Protocolo SPI.



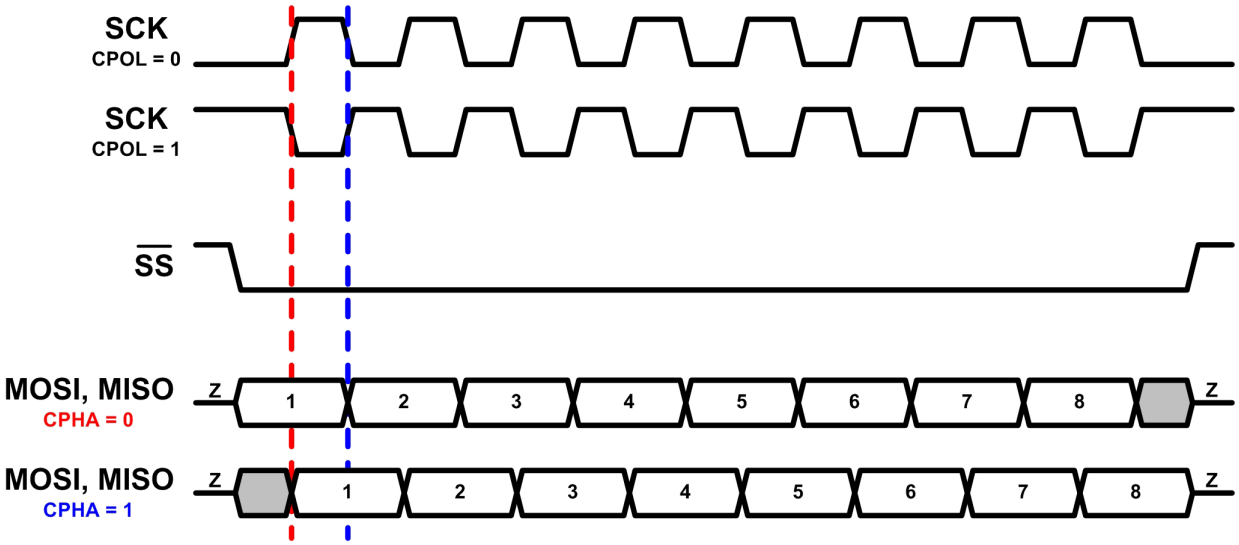
Fonte: A Autora.

menos significativo do registrador do escravo pela porta MOSI, e o bit mais significativo do escravo é conectado ao bit menos significativo do mestre pela porta MISO. Quando o escravo for habilitado, ambos mestre e escravo registram os seus bits menos significativos e deslocam os registradores à esquerda a cada ciclo de *clock*. Após N ciclos de *clock*, o

escravo é desabilitado, colocando-se nível lógico alto no sinal SS.

A polaridade e a fase do *clock* de uma SPI podem variar de um barramento para outro. O valor base do *clock* é 1 se a polaridade for positiva (CPOL = 1) e 0 se a polaridade for negativa (CPOL = 0). Considerando-se que a borda 1 de *clock* é a primeira transição do valor base do *clock* para seu inverso após a descida do sinal "SS", que a borda 2 é a transição seguinte do *clock* (e assim sucessivamente), pode-se dizer que numa SPI com fase zero (CPHA = 0) os dados são amostrados nas bordas ímpares de SCLK e propagados nas bordas pares de SCLK. Já em SPIs com fase um (CPHA = 1), os dados são propagados nas bordas ímpares e amostrados nas bordas pares (MOTOROLA, 2004). Os modos de funcionamento definidos pelos sinais CPOL e CPHA são ilustrados na Figura 9.

Figura 9 – Diagrama Temporal da SPI.



Fonte: Kugelstadt (2009).

Em um barramento SPI, não é necessário implementar o fluxo de dados nos dois sentidos. Pode-se optar por um barramento que realize apenas operações MOSI ou apenas operações MISO, que são os casos utilizados neste projeto.

3 Estudo do Sistema

3.1 Visão Geral do Sistema e HMDs Testados

A lógica do equipamento desenvolvido deve executar três macrofunções: testar a geração de imagens (*stroke*, modo noturno e vídeo), testar os canais de comunicação (protocolos proprietários) e realizar funções de autoteste (verificação do consumo das fontes de tensão elétrica e da integridade das trilhas da placa). Todas estas funções devem ser executadas levando em consideração o tipo de capacete que está sendo testado, que pode ser um DASH (*Display And Sight Helmet*, ou Capacete de Visualização e Observação) ou um JHMCS (*Joint Helmet Mounted Cueing System*, ou Sistema Conjunto de Localização Montado em Capacete).

Ambos HMDs que podem ser testados são destinados a aplicações em aeronaves de asas fixas. Abaixo, são apresentadas descrições gerais destes dois equipamentos, baseadas na referência "Helmet-Mounted Displays: Sensation, Perception and Cognition Issues" (RASH, 2009).

3.1.1 DASH

Os HMDs tipo DASH 3, versão mais popular entre os DASH, consistem em um sistema integrado, onde os componentes ópticos e a bobina de detecção de posição são colocados em um capacete com formato padrão. O DASH 3 possui um visor esférico onde as imagens são projetadas. As imagens não são coloridas, e toda a simbologia é caligráfica. As imagens são geradas por um CRT endereçado por gerador de *stroke* programável.

O capacete é customizado para adaptar-se ao tamanho e ao formato da cabeça dos pilotos, utilizando-se um revestimento de espuma ou termoplástico. O DASH 3 permite que o piloto utilize óculos e máscara de oxigênio (Figura 10). A versão mais pesada do capacete pesa 1,65 kg, e seu centro de gravidade é bem ajustado, fatores muito importantes, visto que o piloto pode ser submetido a forças de até nove vezes a da gravidade.

Figura 10 – DASH.

Fonte: Rash (2009).

3.1.2 JHMCS

O JHMCS é um sistema multifunções baseado em sistemas menores com propósitos distintos, ao contrário do DASH 3, HMD no qual o JHMCS foi baseado, que consiste em um sistema integrado. O JHMCS aumenta a percepção do piloto sobre situações adversas e fornece controle sobre sistemas de direcionamento e sensores.

O JHMCS é monocular (apenas no olho direito), com um CRT de aproximadamente 1,3 cm (Figura 11). Assim como o DASH, o JHMCS utiliza um sensor de detecção de posição eletromagnético. Durante o dia, os símbolos são gerados atráves do uso de um gerador de *stroke*, porém o JHMCS pode ser configurado durante o voo para operar em modo de visão noturna. Além disso, visando o aumento da confiabilidade, os módulos do JHMCS podem ser substituídos durante o voo.

Atualmente, já existe uma versão do JHMCS com *display* digital. O equipamento de testes desenvolvido deve suportar as duas versões do JHMCS.

3.2 Arquitetura do Sistema e Principais Requisitos

Na Figura 12, é apresentado o diagrama de blocos da lógica do FPGA que será desenvolvida neste projeto, definido juntamente com o cliente.

O controle do funcionamento da lógica deve ser feito por um processador de testes, que será implementado utilizando-se o *core* da Xilinx LogiCORE^{TM}MicroBlaze^{TM}Micro

Figura 11 – JHMCS.

Fonte: Rash (2009).

Controller System (XILINX, 2013b). Este processador deve poder se comunicar com todos os blocos internos do FPGA, sendo o mestre do barramento. O processador possui interface AMBA 3.0 (*Advanced Microcontroller Bus Architecture*, ou Arquitetura Avançada de Barramentos de Microcontroladores), também conhecida como AXI (*Advanced eXtensible Interface*, ou Interface Extensível Avançada) (MICHELI; BENINI, 2006). Como a lógica interna do FPGA possui interface *Wishbone*, se faz necessário o uso de pontes (WB_BRIDGE) para a conversão de padrões. O processador possui ainda interface do tipo FSL (*Fast Simplex*[1] *Link*, ou Conexão *Simplex* Rápida).

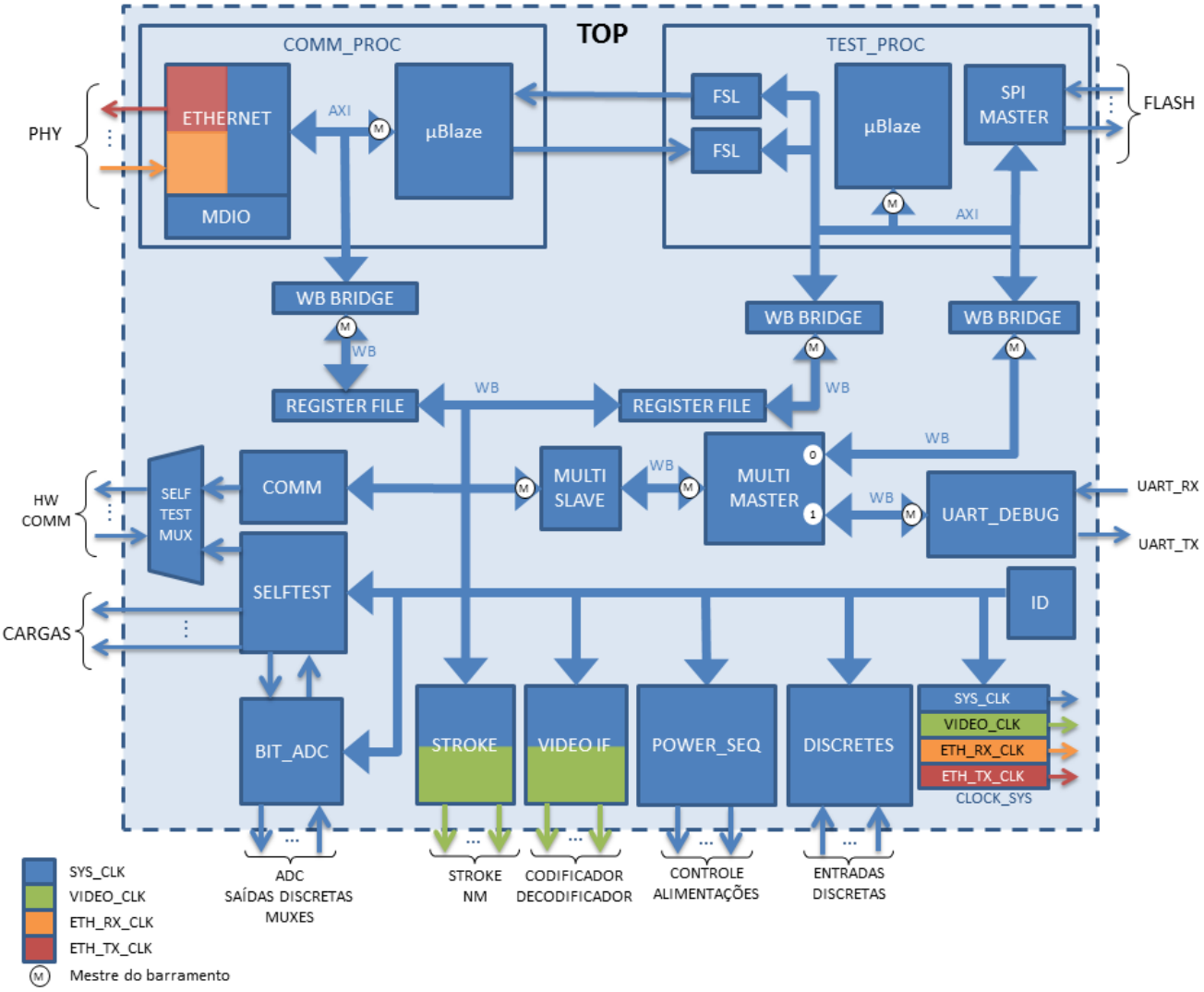
O equipamento de testes deve ser capaz de se comunicar com um PC utilizando o protocolo *Ethernet*. Será utilizado um processador MicroBlaze dedicado e um *core* da Xilinx que incorpora as funções da MII e da MDIO, descritas no protocolo IEEE 802.3 (XILINX, 2013a). Estes blocos também possuem interface AXI, portanto aqui também é necessário o uso do bloco WB_BRIDGE. Também deverão ser usados os blocos REGISTER_FILE, que devem conter informações relevantes de controle e status acessíveis pelo barramento *Wishbone*.

O equipamento também deve possuir uma UART de depuração (*debug*), implementada em VHDL (bloco UART_DEBUG), através da qual deve ser possível acessar todos os blocos da lógica. Como o processador de testes também é um mestre do barramento, o bloco MULTIMASTER deve ser utilizado para conectar os dois mestres ao barramento. A interconexão dos blocos escravos é feita pelo bloco MULTISLAVE.

Existe um bloco de entradas discretas (DISCRETES), que é utilizado para registrar

[1] *Simplex*: Um terminal pode apenas receber ou transmitir dados.

Figura 12 – Diagrama de Blocos do FPGA do Sistema.



Fonte: A Autora.

os sinais vindos do *hardware* que indicam modo de operação e presença de conectores. O processador de testes, TEST_PROC, é o responsável pelo gerenciamento dos modos de operação. A partir dos dados lidos no bloco DISCRETES, este processador controla os testes e o acionamento da lógica.

A lógica do equipamento deve gerenciar o sequenciamento das fontes de tensão elétrica da placa, exceto as que são necessárias para o funcionamento do próprio FPGA (o sequenciamento destas fontes é feito diretamente pelo *hardware* da placa). Esta função será executada pelo bloco POWER_SEQ.

O equipamento de teste deve ser capaz de ler valores analógicos da sua própria

placa (*Built-In Test*). Durante a inicialização ou mediante solicitação externa, devem ser lidas as tensões e correntes internas, as tensões da interface STROKE e as tensões da interface de saídas discretas. O bloco BIT_ADC é o responsável por executar as leituras, que são solicitadas pelo processador de testes.

Deve existir também um modo de autoteste no equipamento. Quando este modo estiver habilitado, deve ser possível testar a integridade de alguns dos canais de comunicação e monitorar a tensão e a corrente das fontes de tensão elétrica do equipamento de testes, através do bloco SELFTEST.

O equipamento de teste deve ser capaz de se comunicar com o HMD e suas diversas placas, através do bloco COMM. A lógica deste equipamento também deve controlar os sinais da interface *stroke*, gerando padrões internamente ou interpretando dados recebidos do processador de testes através de registradores, funções executadas pelo bloco STROKE. O controle da interface de vídeo é feito pelo bloco VIDEO_IF.

Neste trabalho, será descrita a primeira etapa do desenvolvimento da lógica, que consiste nos blocos CLOCK_SYS, BIT_ADC, DISCRETES, MULTISLAVE, MULTIMASTER, POWER_SEQ, SELFTEST, STROKE e UART_DEBUG. A divisão interna de alguns blocos já foi especificada nos requisitos (como os blocos SELFTEST e STROKE), mas a arquitetura dos outros blocos foi definida após o estudo do circuito e dos requisitos.

A lógica implementada deve ser gravada no FPGA Artix-7 XC7A200T-FBG676, da Xilinx, e não poderá ser utilizado mais de 75% dos recursos do FPGA, que possui 215360 células lógicas, 269200 flip-flops e 13140Kb de memória divididos em 365 *block* RAMs. O FPGA possui dez CMTs (*Clock Management Tile*), responsáveis pelo gerenciamento dos diferentes domínios de *clock*.

3.3 Estudo do Circuito

3.3.1 CLOCK_SYS

O bloco CLOCK_SYS deve ser responsável pela distribuição dos *clocks*, pela geração dos *resets* e de algumas bases de tempo. O FPGA deste projeto terá quatro domínios de *clock* diferentes (SYS_CLK, VIDEO_CLK, ETH_RX_CLK e ETH_TX_CLK), e cada um deles deve ter uma linha de *clock* exclusiva no FPGA. A seguir, é explicado o uso de cada um destes sinais.

3.3.1.1 SYS_CLK

Este é o *clock* geral do sistema, que deve ser utilizado para sincronizar quase todos os blocos, exceto a cadeia de processamento de vídeo e o MAC *Ethernet*. O oscilador presente na placa é de 100 MHz, com *jitter*[2] menor do que 1 ps, porém a frequência do sinal utilizado internamente deve ser igual a 150 MHz. Para realizar este aumento de frequência, deve ser utilizado um PLL (*Phase-Locked Loop*, ou Malha de Captura de Fase) interno ao FPGA (ver *Phase-Locked Loops: Theory and Applications* (STENSBY, 1997)).

Conforme documentação da Xilinx (XILINX, 2012), o código VHDL correspondente a um PLL pode ser gerado utilizando-se a ferramenta CORE Generator^{TM}, da Xilinx. Deve-se informar à ferramenta alguns parâmetros, como frequência e *jitter* do *clock* de entrada e frequência do *clock* de saída, e deve-se selecionar as portas que serão utilizadas do bloco original. A partir destas informações, a ferramenta gera o código VHDL do bloco e o *testbench* correspondente. O PLL pode levar até 100 μs para atingir o alinhamento de fase.

3.3.1.2 VIDEO_CLK

Este é o *clock* utilizado para o processamento de imagens da placa (interface de vídeo, *stroke* e modo noturno). O FPGA recebe um *clock* de 27 MHz proveniente de um oscilador presente na placa, que é o mesmo sinal utilizado pelo codificador de vídeo. O FPGA também recebe um *clock* de 54 Mhz, proveniente do decodificador de vídeo presente na placa. O processador de testes deve selecionar, entre estes dois sinais de *clock*, qual sinal será utilizado para o processamento de imagens. A seleção da fonte do sinal VIDEO_CLK será feita através de registradores *Wishbone*.

3.3.1.3 ETH_RX_CLK e ETH_TX_CLK

Estes são os *clocks* do receptor (ETH_RX_CLK) e do transmissor (ETH_TX_CLK) *Ethernet*, recebidos pela interface MII. A frequência destes sinais pode ser igual a 2,5 MHz ou 25 MHz, dependendo do modo de operação da *Ethernet* (10 ou 100 Mbps). A lógica deve suportar mudanças dinâmicas na frequência destes sinais, ou seja, durante a operação.

[2] *Jitter*: falta de exatidão das bordas de *clock* causada pelo circuito gerador do sinal de *clock* (BERNSTEIN, 1998).

3.3.2 BIT_ADC

O bloco BIT_ADC tem duas funções principais: adquirir dados de tensão e corrente da placa, através de quatro multiplexadores e de dois conversores analógico-digital, e configurar seis saídas discretas.

A aquisição de dados através dos conversores poderá ser feita de dois modos, automático ou manual, sendo que a seleção do modo deve ser feita pelo processador de testes, utilizando o barramento *Wishbone*. As saídas discretas também devem ser configuradas pelo processador utilizando o barramento *Wishbone*.

No modo automático, o processador irá requisitar um teste escrevendo em um registrador *Wishbone*. A lógica deverá operar de maneira a realizar a conversão dos valores analógicos e registrá-los, para que posteriormente o processador possa verificá-los. Um registrador de status deve informar ao processador que o teste solicitado já foi realizado. Já no modo manual, o processador solicitará a leitura de um único valor por vez, através do bloco SELFTEST. Neste modo, apenas dois multiplexadores e um ADC serão utilizados, sendo controlados pelo bloco BIT_ADC. Após o término da operação, o bloco BIT_ADC deve enviar o dado convertido e uma indicação de que a conversão já foi feita ao bloco SELFTEST.

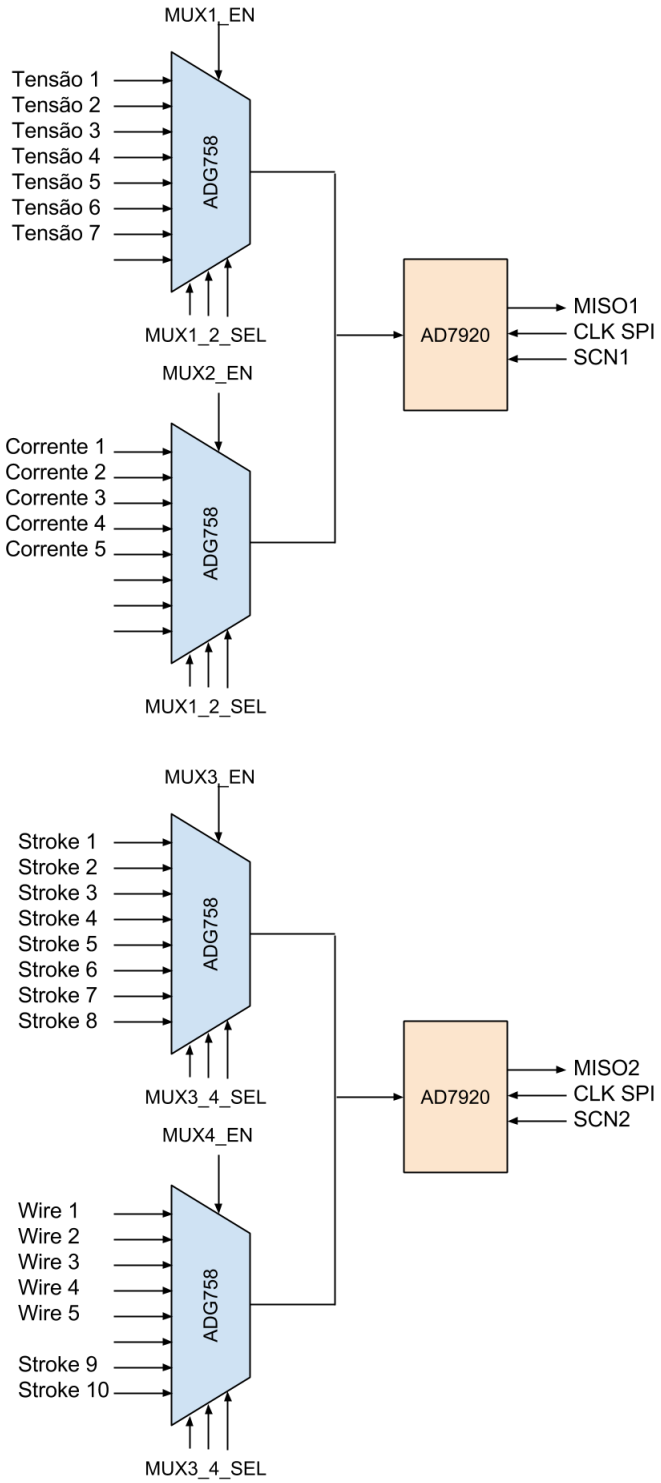
Para que se projete adequadamente este bloco de lógica, é necessário estudar os componentes com os quais o FPGA interage, atentando-se para limites temporais. A configuração dos ADCs e dos multiplexadores na placa é mostrada na Figura 13. A seguir são apresentadas as principais características dos ADCs e dos multiplexadores presentes na placa.

3.3.2.1 AD7920

O conversor analógico-digital (ADC) utilizado é o AD7920, da Analog Devices. Este componente é um escravo SPI que envia o valor de sua entrada analógica convertido a um mestre SPI, recebendo apenas os sinais de controle do mestre (*clock* e seleção). A máxima frequência de *clock* que este componente suporta é 5 MHz. Este conversor é de doze bits, porém o tamanho do dado enviado é de dezesseis bits, sendo os quatro bits mais significativos iguais a zero (ANALOG DEVICES, 2005).

Como é possível observar na Figura 14, a polaridade do *clock* do AD7920 é positiva, ou seja, CPOL é igual a um. A fase do *clock* está deslocada, portanto, CPHA também é igual a um. Assim, após a descida do sinal de seleção (CS, equivalente ao sinal SS, descrito

Figura 13 – Diagrama do Circuito Controlado pelo Bloco BIT_ADC.

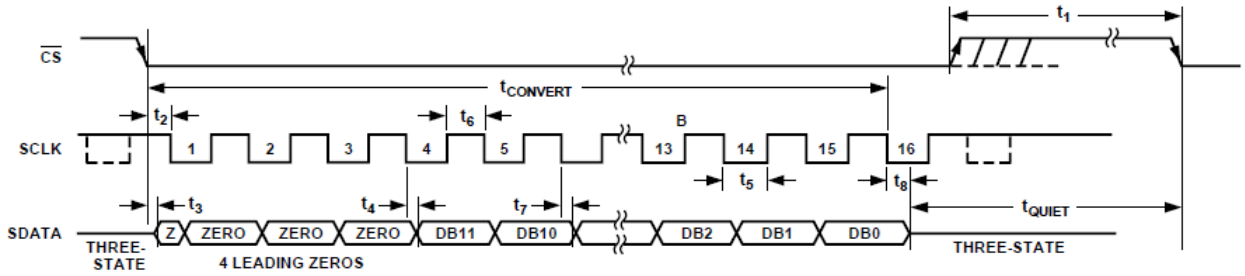


Fonte: A Autora

na subseção 2.2.5), o conversor irá enviar o primeiro bit (zero) após a primeira borda de descida do sinal SCLK, desde que respeitado o tempo mínimo entre a atribuição do valor zero ao sinal CS e a borda de descida do SCLK (t_2), que é de 10 ns. Logo, o mestre deverá

amostrar os dados nas bordas de subida do sinal SCLK.

Figura 14 – Diagrama Temporal da SPI do Conversor AD7920.



Fonte: *Datasheet* do AD7920, Analog Devices (2005).

A largura mínima dos pulsos alto e baixo de *clock* (t_6 e t_5, respectivamente) é de 40% do período de *clock*. A largura mínima do pulso CS, t_1, é de 10 ns. Já o tempo mínimo de silêncio entre o final do envio do último bit de uma conversão e o início de outra (t_{QUIET}) é de 50 ns.

3.3.2.2 ADG758

O ADG758 é um multiplexador com oito canais de entrada. Sua interface consiste na alimentação (tensão positiva, negativa e de referência), oito entradas (S1 a S8), uma saída (D), três seletores (A2, A1 e A0) e um sinal de habilitação (EN). Na Tabela 2 é apresentada a relação entre os valores dos seletores e do *enable* com a saída.

Conforme *datasheet* do fabricante (ANALOG DEVICES, 2013c), durante a habilitação do componente, o tempo máximo para que a saída receba 90% da tensão da entrada selecionada após o sinal de *enable* atingir 50% do seu valor máximo é de 30 ns. Já durante a desabilitação, o tempo máximo para a saída atingir um valor menor do que 90% da última entrada selecionada após o sinal de *enable* atingir 50% do seu valor máximo é de 15 ns. Em uma mudança de seleção, o tempo máximo para que a saída receba 90% da tensão da entrada selecionada após os seletores (A2, A1 e A0) atingirem 50% do seu valor máximo é de 30 ns.

3.3.3 DISCRETES

O bloco DISCRETES deve guardar as informações das entradas discretas em registradores *Wishbone*. Estas entradas discretas permanecem constantes por um tempo

Tabela 2 – Relação entre as Entradas e as Saídas do Multiplexador ADG758.

A2	A1	A0	EN	Saída (D)
X	X	X	0	Nenhum
0	0	0	1	S1
0	0	1	1	S2
0	1	0	1	S3
0	1	1	1	S4
1	0	0	1	S5
1	0	1	1	S6
1	1	0	1	S7
1	1	1	1	S8

Fonte: *Datasheet* do componente ADG758, Analog Devices (2013c).

relativamente longo em relação à frequência de funcionamento do processador (a informação do tipo de capacete, por exemplo, permanece constante durante todo o período do teste). Para impedir que informações incorretas sejam lidas pelo processador de testes por causa de possíveis ruídos na placa ou durante um momento de transição, só devem ser considerados os valores das entradas que permanecerem estáveis por mais de 200 μs.

3.3.4 POWER_SEQ

A lógica do equipamento de testes é responsável por controlar o sequenciamento das fontes de alimentação, tanto durante a inicialização (*power up*), quanto durante o desligamento (*power down*). O processador de testes deve requisitar o início da sequência de *power up* após receber um comando do PC de teste. O processador de testes deve informar ao bloco POWER_SEQ qual o tipo de HMD que está sendo testado (informação adquirida através do bloco DISCRETES), pois existe uma sequência para cada HMD.

O controle das fontes de tensão elétrica é feito a partir dos pinos de habilitação dos reguladores e conversores DC/DC que geram os níveis de tensão requeridos. Além disso, existem trilhas de alimentação que recebem valores de tensão diferentes, dependendo do tipo de capacete conectado. A seleção do valor de tensão destas trilhas é feita a partir de circuitos de *switch*, baseados em MOSFETs, cujas tensões de *gate* são saídas do FPGA.

Para os dois tipos de capacetes, existem três grupos de fontes de tensão que devem ser habilitados sequencialmente, com um intervalo de 10 ms entre os grupos. No entanto, para os capacetes do tipo JHMCS, após a habilitação do grupo 1, o bloco POWER_SEQ deve indicar ao processador, através de registradores, que o teste de comunicação (*handshake*) já pode ser realizado, e deve aguardar que o processador sinalize que a comunição foi realizada com sucesso. Se houver falha na comunicação, o processador deve solicitar o início da sequência de *power down*.

Durante a sequência de inicialização, o processador de testes deve monitorar as fontes de tensão elétrica que estão sendo habilitadas. Se for verificada alguma falha nestas fontes, o processador deve solicitar o início da sequência de desligamento. As sequências de inicialização e desligamento são fixas, não podendo ser alteradas, sendo a sequência de desligamento inversa à de inicialização.

3.3.5 SELFTEST

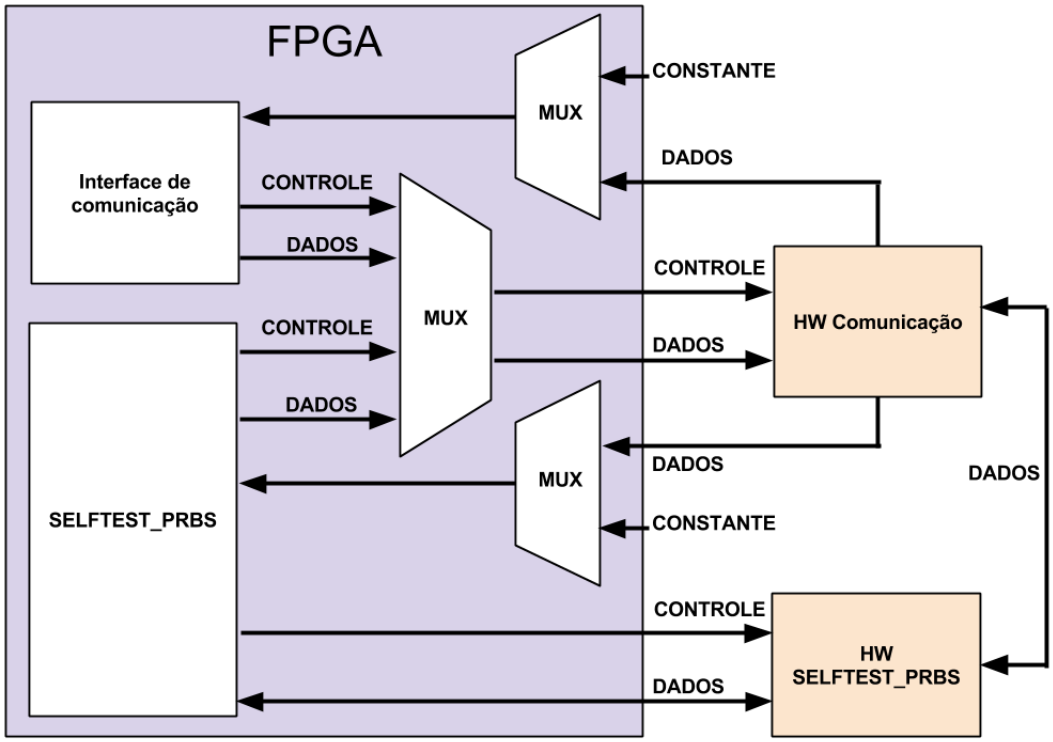
O modo de autoteste no equipamento é habilitado pelo uso do conector de autoteste, que é indicado em uma das entradas do FPGA registradas pelo bloco de entradas discretas. Com este conector deve ser possível testar a integridade dos canais de comunicação, pois ele interliga as saídas dos canais de comunicação às entradas do bloco SELFTEST, e as entradas dos canais de comunicação às saídas do bloco SELFTEST. Esta função será executada pelo bloco SELFTEST_PRBS. O bloco SELFTEST também deve permitir que se ative a carga de uma fonte de tensão e que se leia, através do bloco BIT_ADC, os valores de tensão e corrente elétrica associados a esta fonte, função que deve ser executada pelo bloco SELFTEST_PWR.

3.3.5.1 SELFTEST_PRBS

O bloco SELFTEST_PRBS é responsável por testar a integridade dos canais de comunicação do equipamento de teste com o HMD. Quando o equipamento estiver em modo de autoteste, a interface com o *hardware* responsável pela comunicação será feita pelo bloco SELFTEST, e não pelos blocos de comunicação, seleção feita por multiplexadores no topo do *design*. Além disso, um *hardware* semelhante ao da comunicação foi replicado, invertendo-se os *buffers* de entrada com os de saída. Desta maneira, todos os sinais de entrada do *hardware* de comunicação correspondem a um sinal de saída do *hardware* SELFTEST_PRBS, e vice-versa. A configuração das conexões entre estes blocos é mostrada na Figura 15. A conexão dos sinais dos blocos de *hardware* entre si é feita pelo conector de

autoteste.

Figura 15 – Configuração do Teste dos Canais de Comunicação.



Fonte: A Autora.

Em cada uma das saídas de sinal destes circuitos, o bloco SELFTEST_PRBS deve inserir geradores de PRBS diferentes, que possam ser habilitados separadamente pela escrita em registradores, permitindo assim a detecção de eventuais curtos-circuitos entre os canais de comunicação e de circuitos em aberto. Em cada sinal de entrada destes circuitos deverá ser inserido um detector, que deverá conter um detector de PRBS, um detector de *all-zeros* (todos em zero) e um detector de *all-ones* (todos em um). Os detectores de PRBS deverão indicar, em registradores, se estão sincronizados e se algum erro na sequência foi detectado, após atingir o sincronismo. Os detectores de *all-zeros* e *all-ones* servem para indicar possíveis curtos-circuitos dos canais de comunicação com alguma fonte de alimentação ou com a referência.

Também utilizando registradores, o bloco SELFTEST_PRBS poderá controlar os transceptores presentes na placa (dois LTC2854 e dois LTC2855), os dois transmissores AM26LV31 e os dois receptores AM26LV32. A seguir, serão apresentadas algumas características destes componentes.

3.3.5.1.1 LTC2854 e LTC2855

Estes transceptores possuem um *buffer* de recepção, com entrada diferencial e saída única (*single ended*), e um *buffer* de transmissão, com entrada única e saída diferencial, e recebem três sinais de controle cada: habilitação da recepção (RE), habilitação da transmissão (DE) e habilitação do resistor de terminação entre os terminais diferenciais (TE). O transceptor LTC2854 é *half duplex*, logo, ele pode operar como receptor ou como transmissor, mas não como os dois ao mesmo tempo. Já o transceptor LTC2855 é *full duplex*, portanto pode operar como receptor e transmissor ao mesmo tempo, conforme *datasheet* do fabricante (LINEAR TECHNOLOGY CORPORATION, 2007).

O sinal RE é ativo baixo, ou seja, se estiver em nível lógico alto, as saídas do *buffer* receptor serão colocadas em alta impedância. Já o sinal DE é ativo alto, ou seja, se estiver em nível lógico baixo, as saídas do *buffer* transmissor são colocadas em alta impedância. O sinal TE também é ativo alto, ou seja, a resistência de terminação inserida se TE estiver em nível lógico alto. Se RE estiver em nível lógico alto e DE e TE estiverem em nível lógico baixo, o componente entra no seu estado de desligamento, baixando seu consumo de potência elétrica.

Os transceptores LTC2854 e LTC2855 suportam uma taxa de fluxo de dados de, no máximo, 20 Mbps. Tipicamente, um atraso de 10 ns é observado entre a entrada e a saída do transmissor, e um atraso de 50 ns é observado entre a entrada e a saída do receptor. O tempo máximo de habilitação e desabilitação do transmissor é de 70 ns, reduzindo-se a 50 ns para o receptor. No entanto, partindo do estado de *shutdown*, o tempo de habilitação aumenta para 8 μs, tanto para o receptor quando para o transmissor. Já o tempo máximo de habilitação e desabilitação da resistência de terminação é de 100 μs.

3.3.5.1.2 AM26LV31 e AM26LV32

Os componentes AM26LV31 e AM26LV32 possuem dois sinais de habilitação, porém somente um deles é utilizado, tendo o outro sido conectado à referência. Assim, estes componentes serão controlados apenas pelo sinal de habilitação ativo baixo (G), de maneira que se este sinal estiver em nível lógico alto, as saidas dos *buffers* serão colocadas em alta impedância. O componente AM26LV31 possui quatro *buffers* de transmissão, com entrada única e saída diferencial, enquanto que o AM26LV32 possui quatro *buffers* de recepção, com entrada diferencial e saida única (TEXAS INSTRUMENTS INCORPORATED, 2005a) (TEXAS INSTRUMENTS INCORPORATED, 2005b).

Tanto o componente AM26LV31 quando o AM26LV32 suportam uma taxa de fluxo de dados máxima de 32 Mbps. O transmissor AM26LV31 insere, tipicamente, um atraso de 8 ns entre a entrada e a saída, sendo que o tempo máximo de habilitação e desabilitação do componente é de 20 ns. Já o receptor AM26LV32 insere um atraso típico de 16 ns entre a entrada e a saída, e seu tempo máximo de habilitação e desabilitação é de 40 ns.

3.3.5.1.3 SN65LVDS179D

Existem ainda na placa dois transceptores SN65LVDS179D, porém estes não apresentam sinais de controle. Estes componentes suportam uma taxa de fluxo de dados de até 400 Mbps.Os transmissores tipicamente inserem 1,7 ns de atraso entre a saída e a entrada, enquanto o receptor insere tipicamente 3,7 ns entre a saída e a entrada, conforme *datasheet* do fabricante (TEXAS INSTRUMENTS INCORPORATED, 2009).

3.3.5.2 SELFTEST_PWR

Quando a entrada de indicação do modo de autoteste estiver habilitada, o processador de testes deve poder ler a corrente ou a tensão elétrica de uma determinada fonte de tensão com a carga desta fonte habilitada. Para isso, o processador de testes deve escrever em um registrador *Wishbone* de seleção. Ao ocorrer esta escrita, se nenhuma outra fonte estiver sendo testada, deve ser enviado um pulso de habilitação com 1,5 ms de duração à carga resistiva selecionada. O sinal proveniente do FPGA é conectado ao *gate* de um MOSFET, fazendo a carga em questão ser conectada à sua respectiva fonte.

Após 1 ms do início do pulso, os sinais analógicos da corrente e da tensão já estão estabilizados, então deve ser enviada uma requisição de conversão AD ao bloco BIT_ADC, previamente configurado pelo processador de testes para operar em modo manual. O valor adquirido, enviado ao bloco SELFTEST, deve ser armazenado em um registrador, onde poderá ser lido pelo processador de testes, que irá verificá-lo e compará-lo a valores predefinidos para gerar o resultado do teste (com ou sem falha). Após o fim do envio do pulso, o bloco SELFTEST deve indicar em um registrador de status que está pronto para habilitar outra carga.

3.3.6 STROKE

O bloco STROKE é responsável por gerar padrões de testes no formato *stroke* e no formato de visão noturna, além de controlar a interface com o *hardware*, que consiste

em dois conversores AD9744, um conversor AD5621 e um multiplexador. A geração dos padrões de teste no formato *stroke* é feita pelo bloco STROKE_GEN, enquanto que a geração dos padrões de teste no formato de visão noturna é feita pelo bloco NM_IF. O controle dos dois conversores AD9744, correspondentes aos eixos X e Y, é feito pelo bloco AD9744_IF, enquanto que o controle do conversor AD5621, correspondente ao eixo Z (intensidade), é feito pelo bloco AD5621_IF. O controle do multiplexador, responsável por definir o formato que será enviado ao eixo Z (*stroke*, visão noturna ou desligado), é feito pelo processador através de registradores.

3.3.6.1 STROKE_GEN

O bloco STROKE_GEN é responsável por gerar padrões de testes para a interface *stroke*. Dois padrões de teste devem ser gerados internamente e um padrão de teste deve ser definido pelo processador de testes e escrito em uma memória RAM. Para isso, o processador deve realizar operações de escrita de dados em registradores *Wishbone*, que serão convertidas em operações de escrita na RAM. Através de registradores, também deve ser possível selecionar um dos três padrões de teste, além de comandar o início ou o fim da transmissão do padrão selecionado.

3.3.6.2 NM_IF

O bloco NM_IF é responsável por gerar padrões de testes para a interface de visão noturna. Três padrões de teste devem ser gerados pelo processador de testes e escritos em RAMs, de maneira semelhante ao bloco STROKE_GEN. A seleção de um dos três padrões de teste e os comandos para iniciar ou finalizar a transmissão do padrão selecionado também devem ser feitos a partir de escritas em registradores.

3.3.6.3 AD9744_IF

Este bloco é responsável por controlar os dois conversores digital analógico AD9744, descritos a seguir, utilizados para gerar os valores analógicos a serem enviados às bobinas X e Y. O AD9744_IF deve permitir que o processador de testes selecione a fonte dos dados enviados para os conversores entre os registradores (modo manual) e o bloco STROKE_GEN (modo automático). Esta seleção é feita através de um registrador. Assim, devem existir também registradores para os dados do eixo X, do eixo Y e para o bit SLEEP. Mesmo se o bloco estiver no modo automático, o processador deve poder habilitar o modo *shutdown* através do registrador SLEEP.

O processador também deve ser capaz de habilitar e desabilitar o ganho e o *offset* dos eixos X e Y. Este controle é feito através de escritas em registradores, sendo que inicialmente tanto o ganho quanto o *offset* devem estar desabilitados.

3.3.6.3.1 AD9744

O AD9744 é um conversor paralelo com quatorze bits de entrada e saída diferencial (ANALOG DEVICES, 2013b). Neste circuito, são utilizados apenas os doze bits mais significativos da entrada, estando os dois bits menos significativos do conversor conectados à referência. Estes DACs estão selecionados para operar em modo complemento de dois, pois sua entrada MODE está conectada à fonte de alimentação digital (DVDD).

O AD9744 apresenta um modo de *shutdown*, onde se desabilita a corrente da saída e se reduz o consumo da fonte de alimentação. A habilitação deste modo é controlada pelo bit SLEEP, configurado pelo FPGA, sendo que o mesmo comando é enviado para os dois DACs. O tempo para o componente entrar no modo de *shutdown* é menor do que 50 ns, e para voltar ao modo normal de operação é de aproximadamente 5 μs.

Os bits da entrada são amostrados na borda de subida do *clock*, sendo que a largura mínima do pulso positivo de *clock* é de 1,5 ns. O tempo de *setup* é de 2 ns, enquanto o tempo de *hold* é de 1,5 ns. Como este bloco faz parte do domínio de *clock* do vídeo, será utilizada a frequência de 27 MHz para os conversores.

3.3.6.4 AD5621_IF

Este bloco é o mestre SPI responsável por controlar o conversor digital analógico AD5621, que gera o valor analógico utilizado para controlar a intensidade da interface *stroke* (eixo Z). De forma semelhante ao bloco AD9744_IF, o bloco AD5621_IF deve permitir que o processador de testes selecione a fonte dos dados entre os registradores (modo manual) e o bloco STROKE_GEN (modo automático), conforme o registrador de seleção do modo. Deve existir também um registrador que contém os doze bits de dado e os dois bits de seleção do modo de operação do conversor. Mesmo se o bloco estiver no modo automático, o processador deve poder habilitar os modos *power down* através deste registrador.

3.3.6.4.1 AD5621

O AD5621 é um conversor serial com largura de dezesseis bits, sendo os dois bits mais significativos o controle do modo de operação, os próximos doze bits o dado a ser convertido, e os dois bits menos significativos não utilizados, conforme *datasheet* do fabricante (ANALOG DEVICES, 2013a). Seus modos de operação são apresentados na Tabela 3. O tempo que o conversor leva para sair de qualquer um dos três modos de *powerdown* é de aproximadamente 13 μs.

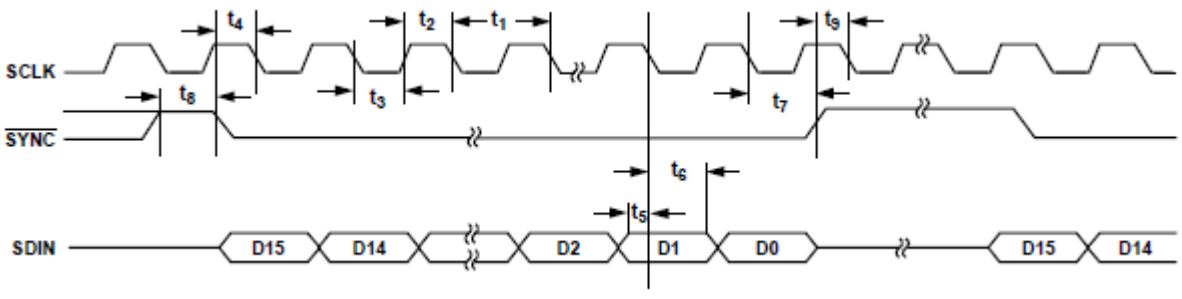
Tabela 3 – Modos de Operação do Conversor AD5621.

Bit 15	Bit 14	Modos de Operação
0	0	Operação Normal
0	1	Modo *Power down*: 1 kΩ da saída para o GND
1	0	Modo *Power down*: 100 kΩ da saída para o GND
1	1	Modo *Power down*: *Three-state*

Fonte: *Datasheet* do componente AD5621, Analog Devices (2013a).

O período mínimo do sinal do *clock* SCLK (t_1) é de 33 ns, ou seja, sua frequência máxima é de 30 MHz. Como este bloco está no domínio de *clock* do vídeo, será utilizado um *clock* de 27 MHz para a SPI. A largura mínima dos pulsos alto e baixo de *clock* (t_2 e t_3, respectivamente) é de 5 ns. O tempo entre a ativação do escravo (SYNC em nível lógico baixo) e a próxima borda de descida de *clock* deve ser de, no mínimo, 10 ns (lembrando que os dados são amostrados pelo escravo na borda de descida). Os tempos mínimos de *setup* (t_5) e *hold* (t_6) são de 5 e 4,5 ns, respectivamente. A largura mínima do pulso de nível alto de SYNC, t_8, é de 20 ns.

Figura 16 – Diagrama Temporal da SPI do Conversor AD5621.



Fonte: *Datasheet* do AD5621, Analog Devices (2013a).

3.3.7 UART_DEBUG

O bloco UART_DEBUG será utilizado durante a depuração para emular os testes que seriam feitos pelo processador de testes e para configurar parâmetros da lógica do FPGA. Para isto, a UART_DEBUG deve ter acesso a todos os blocos internos implementados pela lógica do FPGA, permitindo que se realize escritas e leituras em seus registradores *Wishbone*. Logo, o bloco UART_DEBUG deve ser mestre *Wishbone*, assim como o processador de testes.

A interface UART deve ser escrava, enquanto o PC será o mestre. O bloco UART_DEBUG deverá suportar a velocidade de transferência de 115200 bits/s, utilizando o padrão 8-N-1 (oito bits de dado, sem bit de paridade, um *stop-bit*).

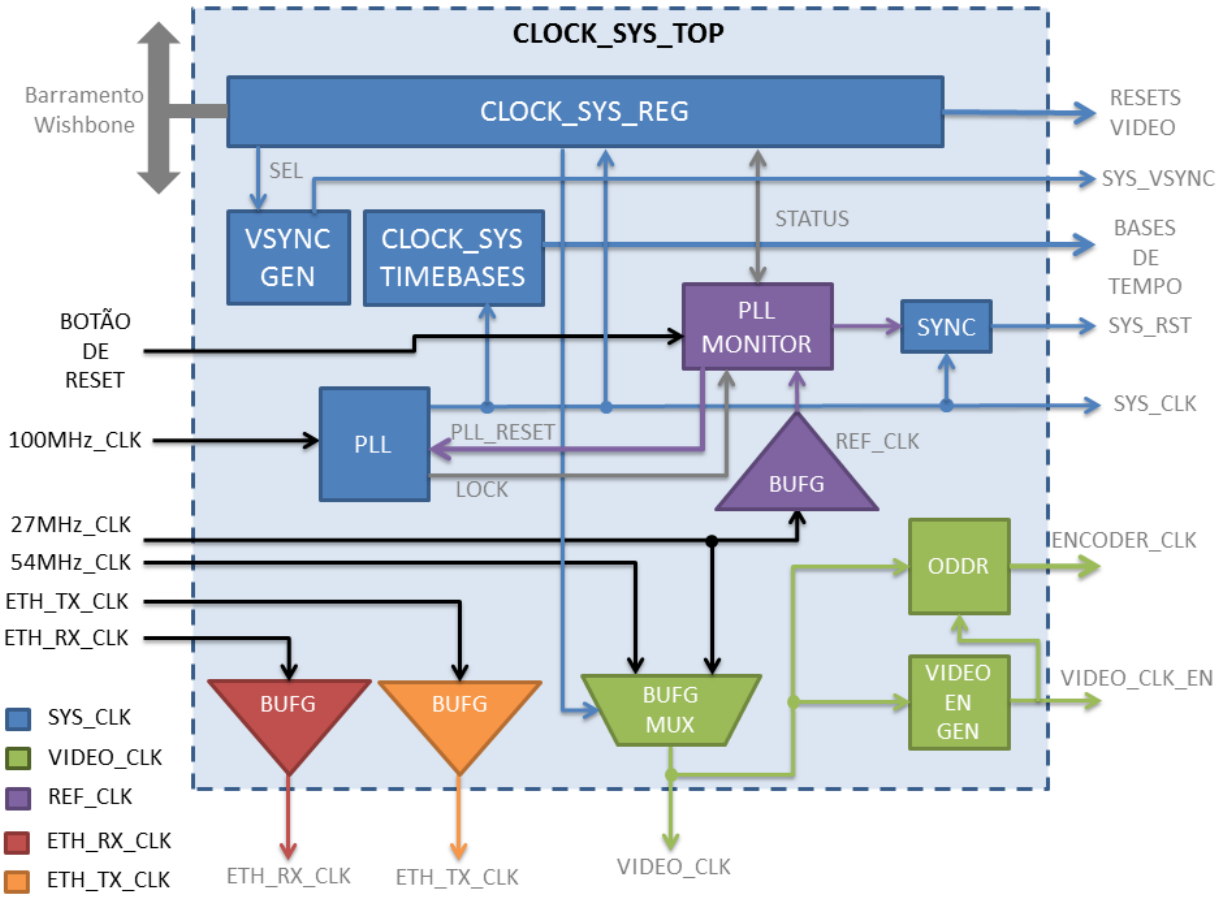
4 Desenvolvimento

4.1 Lógica Programável

4.1.1 CLOCK_SYS

O diagrama de blocos de CLOCK_SYS_TOP é apresentado na Figura 17. O bloco CLOCK_SYS_TOP é estrutural, servindo para interconectar seus blocos internos.

Figura 17 – Diagrama de Blocos de CLOCK_SYS.



Fonte: A Autora.

4.1.1.1 BUFG

Os blocos BUFG são componentes próprios do FPGA. Um BUFG é definido como um *buffer* de *clock* global com alto *fanout* limite[1] e com baixo *skew*[2]. BUFGs são indicados para *clocks*, *resets* e *clock enables*, pois suas saídas são conectadas nos recursos de roteamento global (XILINX, 2012). Neste caso, foram utilizados BUFGs para os sinais REF_CLK, ETH_RX_CLK e ETH_TX_CLK.

4.1.1.2 BUFGMUX

O bloco BUFGMUX é um *buffer* multiplexador de *clock* global, e também é um componente próprio do FPGA. Ele contém duas entradas de *clock*, I0 e I1, uma entrada de seleção, S, e uma saída de *clock*, O. Se o seletor S estiver em nível lógico baixo, a saida O recebe a entrada I0, e se o seletor estiver em nível lógico alto, a saida recebe a entrada I1 (XILINX, 2012). Nesta aplicação, I0 é a entrada de *clock* de 27 MHz, I1 é a entrada de *clock* de 54 MHz, S é uma saida do bloco CLOCK_SYS_REG e O é o sinal VIDEO_CLK.

4.1.1.3 VIDEO_EN_GEN

Este bloco é responsável por gerar o sinal de *clock enable* para o VIDEO_CLK, e é constituído basicamente por um *flip-flop*, cuja entrada de *clock* é o sinal VIDEO_CLK, cuja entrada de *reset* é o sinal VIDEO_RST, e cuja entrada de dado é variada. Quando reinicializado, a saída do bloco é sempre zero. Quando o seletor do VIDEO_CLK é igual a zero (frequência de VIDEO_CLK igual a 27 MHz), a entrada de dado do *flip-flop* é sempre um. Quando o seletor do VIDEO_CLK é igual a um, a entrada de dado do *flip-flop* é realimentada com o inverso da sua saída (*flip-flop toggle*), fazendo o sinal VIDEO_CLK_EN ter frequência de 27 MHz. Este sinal é necessário para que o processamento dos dados dos blocos do domínio VIDEO_CLK seja feito sempre na mesma taxa, independentemente da frequência de VIDEO_CLK.

4.1.1.4 ODDR

O ODDR (ver subseção 2.1.3) foi instanciado de maneira a gerar em sua saída um *clock* de 27 MHz, independente da frequência da entrada VIDEO_CLK. Para fazer isto, conectou-se o sinal VIDEO_CLK_EN na entrada D1, e o inverso deste sinal, com atraso de um ciclo de VIDEO_CLK, na entrada D2. Assim, se a frequência de VIDEO_CLK

[1] *Fanout* Limite: número máximo de portas de carga que podem ser conectadas a uma saida.

[2] *Clock Skew*: diferença de tempo de chegada de uma mesma borda de *clock* em partes diferentes do circuito, causada pelo circuito de distribuição do sinal de *clock* (BERNSTEIN, 1998).

for 27 MHz, o sinal VIDEO_CLK_EN estará sempre em nível lógico alto, ou seja, D1 = 1 e D2 = 0, de maneira que o sinal de saída do ODDR seja semelhante ao da entrada. Se a frequência de VIDEO_CLK for 54 MHz, o sinal VIDEO_CLK_EN oscilará a 27 MHz. Quando VIDEO_CLK_EN for igual a 1, D1 será igual a 1 e D2 também será 1, pois no ciclo anterior VIDEO_CLK_EN era igual a 0. Quando VIDEO_CLK_EN for igual a 0, D1 e D2 são iguais a 0. Assim, neste caso, tem-se na saída a mesma frequência de VIDEO_CLK_EN.

4.1.1.5 CLOCK_SYS_TIMEBASES

Este bloco é constituído por dois contadores, um de oito bits e outro de quatorze bits. O menor contador é utilizado para gerar a base de tempo de 1 μs, e o maior utilizado para gerar a base de tempo de 100 μs. Ambos os contadores são incrementados a cada ciclo de SYS_CLK, e zerados quando atingem o número de ciclos de *clock* correspondente ao período do sinal que se quer gerar. As saídas correspondentes às bases de tempo permanecem em nível lógico baixo, recebendo nível lógico alto apenas durante o ciclo de *clock* no qual o contador atinge seu valor limite.

4.1.1.6 VSYNC_GEN

Este bloco é responsável por gerar internamente o sinal VSYNC, que será utilizado para sincronizar a geração de vídeo e de padrões *stroke*. Este sinal pode ter frequência igual a 50 ou 60 Hz, cuja seleção é feita através de um registrador. A largura do pulso de VSYNC difere para as frequências de 50 e 60 Hz. Para gerar tais pulsos, criou-se um contador que, ao atingir o valor zero, é carregado com uma constante correspondente à frequência selecionada. Também é carregado, neste momento, o valor da largura de pulso correspondente à frequência selecionada. Então, o contador passa a ser decrementado a cada ciclo de SYS_CLK, enviando o valor lógico zero para sua saída SYS_VSYNC. Quando o contador atinge o valor da largura do pulso, a saída SYS_VSYNC passa a ter valor lógico um. Quando o contador atinge zero, a saída é novamente desativada e os valores iniciais são recarregados.

4.1.1.7 PLL

O bloco PLL, gerado com o auxílio do Core Generator^{TM}, possui quatro portas: uma entrada de *clock*, uma entrada de *reset*, uma saída de *clock* e uma saída de *lock*. A saída de *lock* serve para que o PLL indique que atingiu alinhamento de fase e a frequência desejada, dentro de limites pré-definidos. Se a entrada de *clock* parar de oscilar na frequência correta

ou se o seu alinhamento de fase for violado, a saída *lock* será desativada.

Executando-se a simulação do bloco PLL gerada pelo Core Generator™, observou-se que a saída de *clock* leva aproximadamente 500 ns para começar a oscilar, e que a saída *lock* leva aproximadamente 600 ns para ser ativada. Porém, enquanto a saída de *clock* não estiver oscilando na frequência correta, não se pode tirar o restante do circuito do estado de *reset*. Por isso, foi criado o bloco PLL_MONITOR, descrito a seguir.

4.1.1.8 PLL_MONITOR

O bloco PLL_MONITOR opera no domínio REF_CLK, proveniente da entrada de *clock* de 27 MHz, para que não seja dependente da saída de *clock* do PLL. Este bloco é formado por uma máquina de estados, que depende do sinal de *lock* do PLL e do *reset* gerado por um botão na placa. Neste bloco são gerados os sinais de *reset* para o PLL e para o domínio SYS_CLK (sincronizado pelo bloco SYNC).

O sinal de *reset* passa por um bloco de estabilização (*debounce*) para garantir que possíveis ruídos sejam filtrados. Este bloco altera o valor de sua saída apenas se a entrada permanecer estável por um número definido de pulsos de uma base de tempo. Neste caso, definiu-se que após três pulsos de 100 μs, ou seja, após, no mínimo, 200 μs, o sinal de entrada é considerado estável. A base de tempo é gerada da mesma forma descrita no bloco CLOCK_SYS_TIMEBASES, alterando apenas o tamanho do contador e o *clock* mestre, que neste caso é REF_CLK.

A máquina de estados é inicializada no estado NOT_LOCKED, no qual as duas saídas de *reset* permanecem ativas. Se a entrada de *reset* do botão não estiver ativa, a máquina permanece nesse estado por quatro ciclos de REF_CLK, indo para o estado WAIT, onde apenas REF_SYS_RST permanece ativo. Neste estado, a máquina espera o tempo de inicialização do PLL durante aproximadamente 1,2 ms. Se, dentro deste tempo, a entrada de *lock* for ativada, a máquina vai para o estado LOCKED, onde nenhuma saída de *reset* permanece ativa. No entanto, se após 1,2 ms o PLL não ativar a *flag lock*, a máquina volta para o estado NOT_LOCKED, onde o PLL é reiniciado novamente. Se a máquina estiver no estado LOCKED e o PLL perder o sincronismo, a máquina vai para o estado NOT_LOCKED. A partir de qualquer estado, se o *reset* da placa for acionado, a máquina volta para o estado NOT_LOCKED.

4.1.1.9 SYNC

Este bloco consiste em dois registradores no domínio SYS_CLK encadeados. As entradas de *set* dos dois registradores são conectadas à saída REF_SYS_RST do bloco PLL_MONITOR. Quando estas entradas não estiverem ativas, o primeiro registrador amostra o valor lógico zero, e o segundo registrador amostra a saída do primeiro. Assim, a saída deste bloco, SYS_RST, não provocará violações de temporização no restante dos blocos.

4.1.1.10 CLOCK_SYS_REG

Este é o bloco de registradores *Wishbone*. Através dele, é possivel selecionar a fonte do sinal VIDEO_CLK (27 ou 54 MHz), controlar o *reset* dos blocos do domínio VIDEO_CLK, do codificador e do decodificador de vídeo da placa, monitorar o funcionamento do PLL, a partir do bloco PLL_MONITOR, e selecionar a frequência do sinal VSYNC.

4.1.2 BIT_ADC

Na Figura 18 é apresentado o diagrama de blocos de BIT_ADC_TOP, bloco estrutural que opera no domínio SYS_CLK. Pode-se observar nesta Figura que os blocos BIT_CTRL e SPI_IF estão identificados com seu nome e um número entre parênteses, o que significa que se trata de diferentes instâncias da mesma entidade.

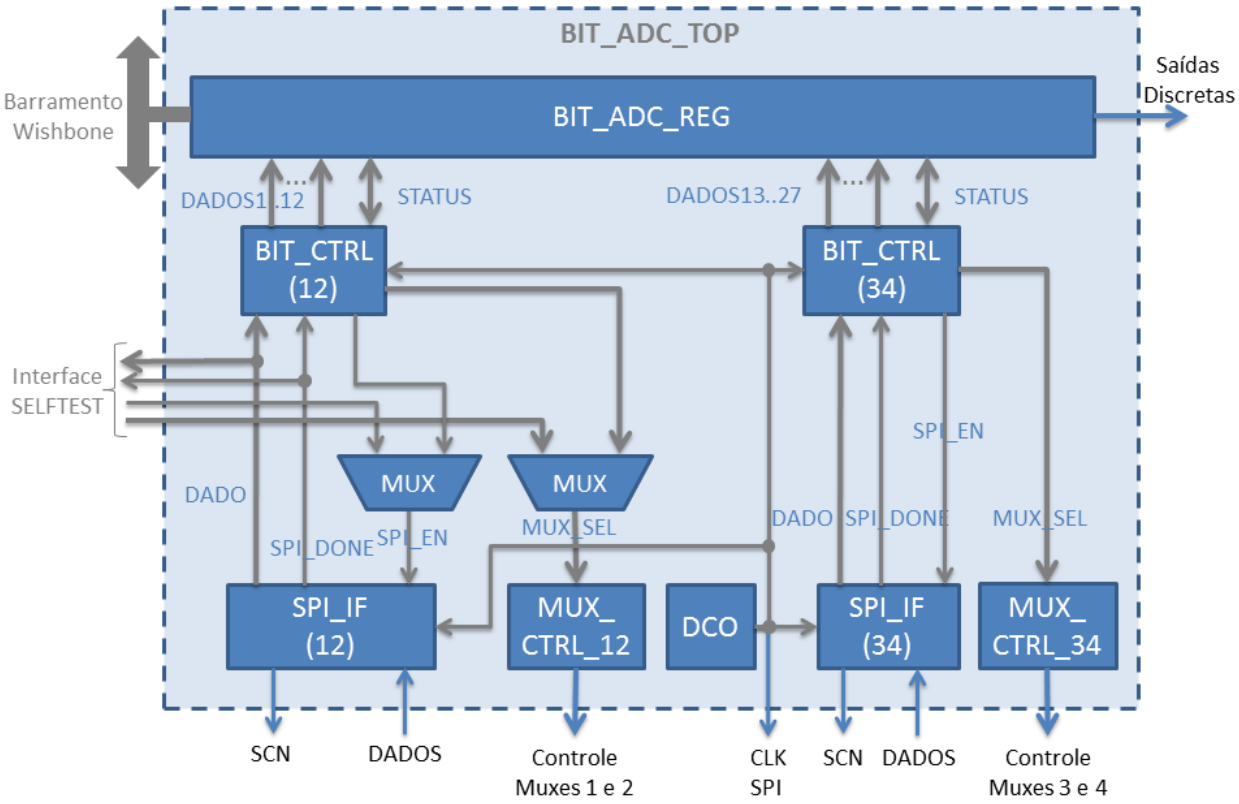
4.1.2.1 DCO

O DCO é utilizado para gerar o *clock* para a interface SPI, com frequência de 4 MHz. Para isso, utilizou-se um DCO do tipo Acumulador, com N igual a 10 e K igual a 27 (ver subseção 2.1.2).

4.1.2.2 BIT_CTRL

Este bloco é constituído pela máquina de estados que controla o teste automático. Em seu estado inicial do modo automático (IDLE), a máquina permanece monitorando a seleção do modo de operação, vinda do bloco BIT_ADC_REG, que tem prioridade sobre uma requisição de teste. Se a máquina estiver neste estado e o modo de operação for alterado para manual, a máquina passa para o estado inicial do modo manual (INIT), onde não realiza nenhuma operação, apenas monitora o modo de operação e retorna para

Figura 18 – Diagrama de Blocos de BIT_ADC.



Fonte: A Autora.

o estado anterior se houver alteração do modo. O modo de operação deste bloco só pode ser alterado se a máquina estiver nos estados INIT ou IDLE.

No estado IDLE, a máquina inicializa um contador de quatro bits com o valor 1 e aguarda uma requisição de testes. Se um teste for requisitado, a máquina executa os seguintes passos:

1. envia um comando ao bloco SPI_IF correspondente para que uma conversão seja realizada e envia o valor do contador ao bloco MUX_CTRL correspondente;
2. aguarda a indicação de que a recepção SPI foi finalizada;
3. registra o dado recebido do bloco SPI_IF correspondente;
4. aguarda um ciclo de *clock* do SPI;
5. compara o valor do contador a uma *generic* que define o valor máximo. Se o valor do contador for menor, incrementa o contador e volta para o passo 1. Se o valor do

contador for igual ao limite, aguarda um ciclo de *clock* SPI, indica que o teste foi finalizado e volta para o estado IDLE.

A *generic* que define o valor máximo do contador é igual a 12 na instância de BIT_CTRL correspondente aos multiplexadores 1 e 2 e igual a 15 na instância correspondente aos multiplexadores 3 e 4.

4.1.2.3 MUX

Os dois multiplexadores são utilizados para selecionar a fonte da requisição de recepção de dados do bloco SPI_IF correspondente aos multiplexadores 1 e 2 da placa e para selecionar a fonte dos bits de seleção enviados ao bloco MUX_CTRL_12. Em modo de operação automático, estes dados são provenientes do bloco BIT_CTRL (12), e, em modo manual, do bloco SELFTEST, externo ao bloco BIT_ADC_TOP.

4.1.2.4 MUX_CTRL_12 e MUX_CTRL_34

Estes dois blocos são utilizados para converter os sinais de seleção dos multiplexadores, de quatro bits cada, nos sinais que fazem interface com os multiplexadores fisicamente (seletores e *enables*). Para isso, o bloco MUX_CTRL_12 converte sua entrada, que deve conter valores de 1 a 12, em um sinal de *enable* para o multiplexador 1, um sinal de *enable* para o multiplexador 2 e um seletor de três bits, comum para os multiplexadores 1 e 2. Já o bloco MUX_CTRL_34 converte sua entrada, que deve conter valores de 1 a 15, em um sinal de *enable* para o multiplexador 3, um sinal de *enable* para o multiplexador 4 e um seletor de três bits, comum para os multiplexadores 3 e 4.

Ambos blocos são baseados em uma instrução *case* dependendo do valor da entrada de seleção, que é lido a cada borda de subida de SYS_CLK. Nos dois blocos, se for recebido um valor inválido na entrada (fora da faixa especificada) ou se SYS_RST for ativado, os multiplexadores são desabilitados.

4.1.2.5 SPI_IF

Este bloco é composto por uma máquina de estados que controla o ciclo de recepção de dados pela interface SPI. Ao receber um comando, se estiver em seu estado inicial, a máquina aguarda uma borda de subida do *clock* SPI, e então habilita o sinal de seleção do escravo (SS). Então, nas dezesseis bordas de subida do *clock* SPI seguintes, a máquina desloca seu registrador de dados (ver subseção 2.2.5). Após o último ciclo de recebimento

de dados, a máquina desabilita o sinal SS e envia uma indicação de que a operação foi finalizada (SPI_DONE). Após o término da operação, a máquina aguarda um ciclo de *clock* do SPI e então retorna para seu estado inicial, garantindo assim que o tempo de silêncio do conversor AD7920 seja respeitado.

4.1.2.6 BIT_ADC_REG

Este é o bloco de registradores *Wishbone* de BIT_ADC_TOP. Por meio deste bloco, é possível configurar as seis saídas discretas, escrevendo seus valores diretamente em um registrador. Através dos registradores também é possível controlar os testes, selecionando o modo de operação do bloco BIT_ADC_TOP (automático ou manual) e requisitando testes automáticos (requisição feita separadamente para os multiplexadores 1 e 2 e para os multiplexadores 3 e 4). As requisições de testes são feitas através de ações automáticas: ao se escrever o valor lógico 1 nos bits correspondentes às requisições de testes, um pulso é enviado ao bloco BIT_CTRL correspondente.

Utilizando um registrador, os blocos internos fornecem informações de status (se os testes foram finalizados, se cada um dos testes automáticos já foi realizado, entre outras). Existe também um registrador para cada entrada de dado dos multiplexadores, totalizando 27 registradores de 12 bits cada. Por isso, é importante que se tenha a informação de que os testes automáticos já foram executados pelo menos uma vez, para que se garanta que os dados lidos são válidos.

4.1.3 DISCRETES

O bloco DISCRETES é constituído por um bloco de registradores *Wishbone* e um bloco de estabilização (*debounce*) para cada uma das entradas, idêntico ao que foi utilizado no bloco PLL_MONITOR. Neste caso, também definiu-se que após três pulsos de 100 μs, ou seja, após, no mínimo, 200 μs, o sinal de entrada é considerado estável. As saídas dos estabilizadores são conectadas aos registradores, que não permitem a operação de escrita, apenas de leitura.

4.1.4 ID

Este bloco não foi solicitado pelo cliente nos requisitos, porém foi adicionado para garantir o controle das versões do código VHDL gravadas no FPGA do equipamento de teste. O bloco ID, propriedade intelectual da empresa onde o projeto está sendo realizado, é

basicamente um bloco escravo *Wishbone*, onde são registradas as informações do fabricante, o número da versão do código, a data e a hora da compilação. A atualização dos dados registrados é feita por *script*, conforme será explicado na seção 4.3.

4.1.5 MULTISLAVE e MULTIMASTER

Os blocos MULTISLAVE e MULTIMASTER são necessários para conectar os diversos escravos *Wishbone* aos dois mestres *Wishbone* (processador de testes e UART_DEBUG). Foram utilizados *cores* previamente desenvolvidos, de propriedade intelectual da empresa onde o projeto está sendo desenvolvido. Para o bloco MULTISLAVE, deve-se definir um mapa com o endereço base de cada escravo, que é utilizado pelo mestre para realizar operações nos registradores de um determinado bloco. Para o bloco MULTIMASTER, deve-se definir um mestre prioritário (neste caso, o processador de testes), que tem preferência no caso de os dois mestres tentarem iniciar uma operação simultaneamente.

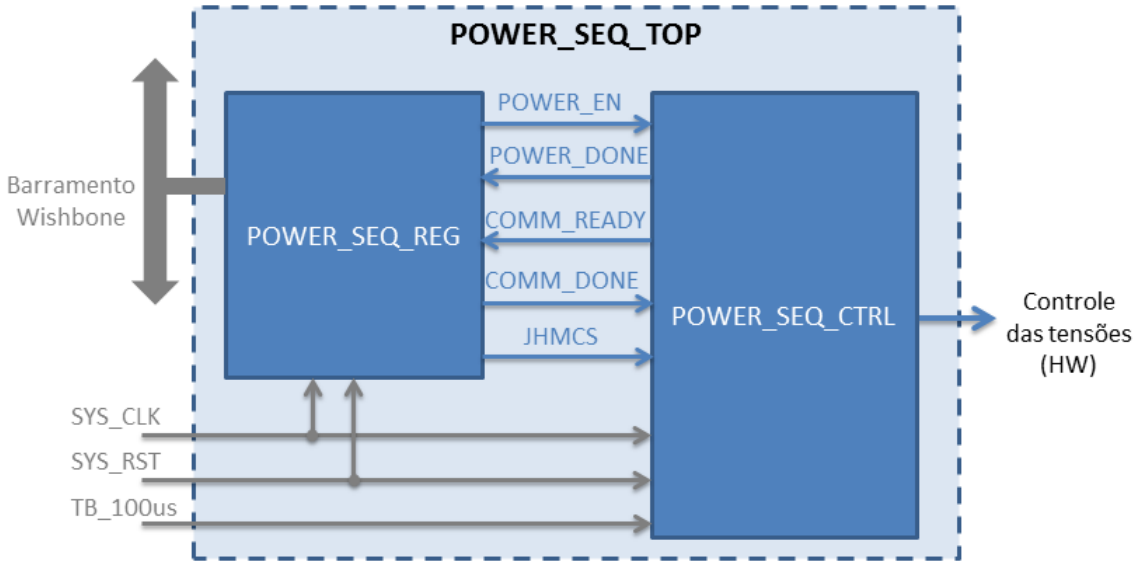
4.1.6 POWER_SEQ

O bloco POWER_SEQ foi dividido em três blocos: o bloco de registradores *Wishbone* (POWER_SEQ_REG), o bloco de controle (POWER_SEQ_CTRL) e o bloco estrutural (POWER_SEQ_TOP), como mostra a Figura 19. O bloco POWER_SEQ_TOP é estrutural, servindo apenas para conectar os outros dois blocos entre si e ao topo da lógica do projeto.

4.1.6.1 POWER_SEQ_REG

O bloco POWER_SEQ_REG possui apenas um registrador composto por cinco bits: POWER_EN, POWER_DONE, COMM_READY, COMM_DONE e JHMCS. O bit POWER_EN é escrito pelo processador de testes e serve para iniciar as sequências de *power up* ou *power down*. Através do bit POWER_DONE, o bloco POWER_SEQ_CTRL indica que a sequência de *power up* foi finalizada. Ativando o bit COMM_READY, o bloco POWER_SEQ_CTRL indica ao processador de testes que o *handshake* da comunicação pode ser executado. Ativando o bit COMM_DONE, o processador de testes indica que o *handshake* foi realizado com sucesso. Através do bit JHMCS, o processador de testes indica qual tipo de capacete está sendo testado.

Figura 19 – Diagrama de Blocos de POWER_SEQ.



Fonte: A Autora.

4.1.6.2 POWER_SEQ_CTRL

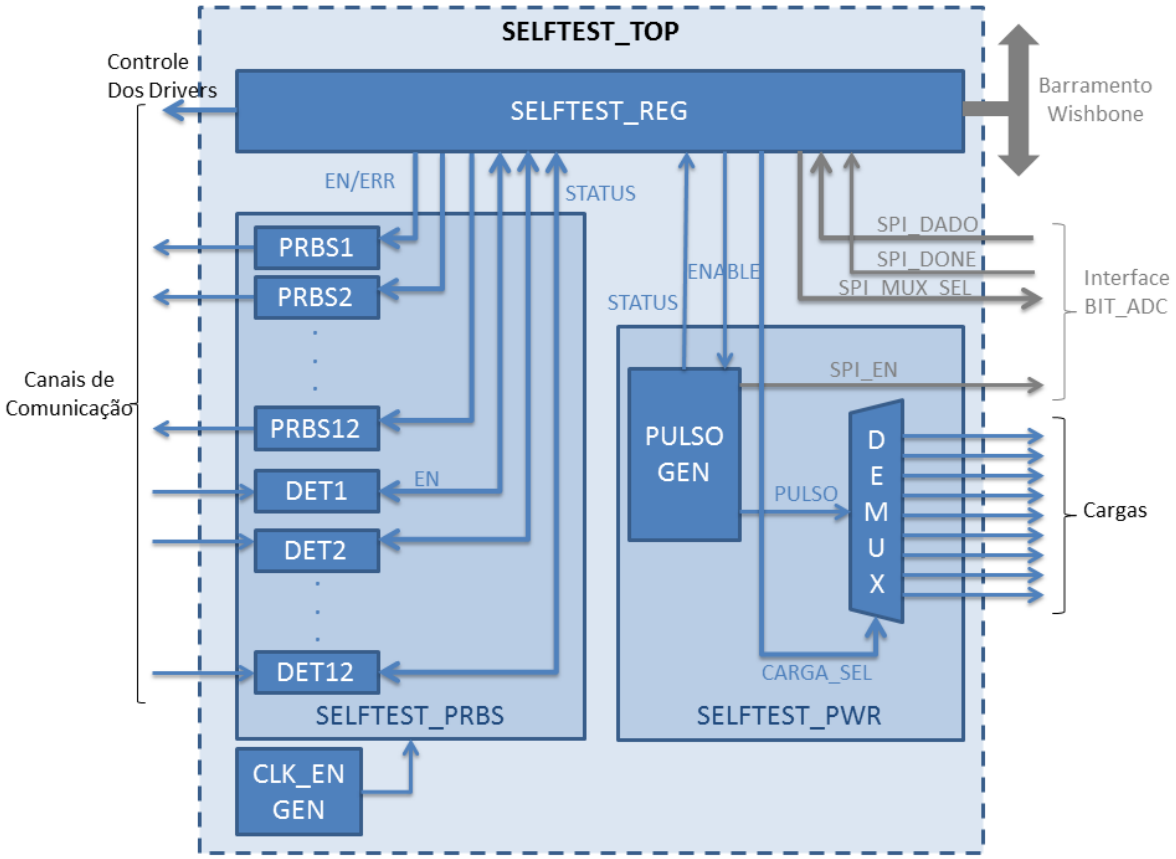
O bloco POWER_SEQ_CTRL é formado basicamente por uma máquina de estados com saídas Moore[3], exceto por uma saída do tipo Mealy[4] (a saída POWER_DONE é desabilitada se a entrada POWER_EN é desabilitada) (NAVABI, 2006). Se o bloco for reinicializado, a máquina volta ao estado inicial (desligado), onde todas as saídas são desabilitadas. Quando o processador de testes ativa o bit POWER_EN, a máquina de estados sai do estado desligado, selecionando o próximo estado (e, consequentemente, a sequência de *power up*) conforme o tipo de capacete. A máquina controla o tempo entre a habilitação de cada grupo de tensão através da base de tempo de 100 μs. O processador pode solicitar o início da sequência de *power down* mesmo durante o *power up*. A máquina de controle inicia a sequência de *power down* a partir do ponto que ela se encontra na sequência de *power up*.

4.1.7 SELFTEST

O bloco SELFTEST_TOP é estrutural, servindo para conectar os blocos SELFTEST_REG, SELFTEST_PRBS, CLK_EN_GEN e SELFTEST_PWR entre si e a outras unidades (barramento *Wishbone* e BIT_ADC). Seu diagrama de blocos é apresentado na Figura 20.

[3] Moore: As saídas dependem apenas do estado atual da máquina de estados.
[4] Mealy: As saídas dependem do estado atual da máquina de estados e das entradas.

Figura 20 – Diagrama de Blocos de SELFTEST.



Fonte: A Autora.

4.1.7.1 CLK_EN_GEN

O bloco CLK_EN_GEN é utilizado para dividir a frequência do *clock* por oito, gerando assim um sinal de *enable* com frequência de 18,75 MHz. Para isso, utilizou-se um *shift-register* de oito bits, com um bit em nível lógico alto e os outros sete em nível lógico baixo. Os bits são deslocados a cada borda de subida de SYS_CLK, sendo o bit mais significativo o sinal desejado. Este sinal é utilizado no bloco SELFTEST_PRBS para controlar a taxa dos geradores e receptores de PBRS. É necessário que se transmita com uma frequência mais baixa do que a do SYS_CLK pois estes dados passam por alguns transceptores que não suportam taxas tão elevadas (ver subseção 3.3.5.1).

4.1.7.2 SELFTEST_PRBS

O bloco SELFTEST_PRBS é formado por doze geradores de PRBS e doze detectores. Os geradores de PRBS são blocos formados por um LFSR de nove bits, sendo a sua

realimentação a operação lógica ou-exclusivo entre o bit 4 e o bit 8 do registrador (ver subseção 2.1.5). O LFSR é deslocado na borda de subida do sinal de *clock* somente se sua entrada de habilitação do *clock* estiver em nível lógico alto. Essa entrada é formada pela saída da operação lógica AND entre o sinal de 18,75 MHz e a habilitação do PRBS, vinda do bloco de registradores. Esta habilitação é individual para cada um dos geradores de PRBS.

Cada detector é, na verdade, formado por três detectores diferentes: um detector de PRBS, um detector de *all-zeros* e um detector de *all-ones*, sendo os dois últimos o mesmo bloco (DET_ALL), diferindo apenas na instanciação pelo valor de uma *generic*. A entrada de *clock enable* dos detectores é formada pela saída da operação lógica AND entre o *clock enable* de 18,75 MHz e a habilitação dos detectores, vinda do bloco de registradores. Esta habilitação é individual para cada um dos detectores.

O bloco DET_ALL, quando habilitado pelo seu *clock enable*, incrementa um contador a cada ciclo de *clock* em que o valor da sua entrada for diferente do valor que se quer detectar. O contador é incrementado até o valor três, e é zerado a cada pulso da base de tempo usada (neste caso, 1 μs). Se entre dois pulsos da base de tempo o valor da entrada for diferente do valor a ser detectado durante menos de três ciclos, a saída ALL é ativada, indicando que o valor da entrada está fixo no nível lógico de referência.

O bloco de detecção de PRBS contém um gerador de PRBS semelhante ao do emissor. A saída do gerador de PRBS é registrada em um *shift-register*, que é desabilitado após ter carregado nove bits. Este registrador é então comparado ao *shift-register* que contém os dados da entrada. Quando os dois registradores tiverem o mesmo valor, considera-se que o detector está sincronizado, ativando-se sua saída de indicação de sincronismo (esta indicação só é alterada mediante *clear*). Quando sincronizado, o detector é capaz de detectar erros, comparando sua entrada de dados à saída do seu gerador de PRBS.

4.1.7.3 SELFTEST_PWR

O bloco SELFTEST_PWR é formado por um gerador de pulsos e um demultiplexador. O gerador de pulsos (PULSO_GEN), ao receber um comando dos registradores, se não estiver enviando um pulso, habilita a saída do pulso e carrega a largura do pulso em um contador, definida através de uma *generic* com um valor que produza um pulso de 1,5 ms. Este contador é decrementado a cada borda de subida de *clock*. Quando o contador atingir um valor tal que indique que o envio do pulso iniciou 1 ms antes, um outro pulso, com largura de um ciclo de *clock*, é enviado ao bloco BIT_ADC, para que seja efetuada

a leitura da entrada correspondente à carga que foi habilitada (o bloco BIT_ADC deve estar no modo manual). Quando o contador atinge o valor zero, o bloco PULSO_GEN desabilita a saída do pulso da carga e indica que está pronto para enviar outro pulso.

O demultiplexador, a partir do seletor que também é enviado ao bloco BIT_ADC e da informação do tipo de capacete conectado, conecta sua entrada, proveniente do bloco PULSO_GEN, à saída correspondente a uma das cargas, mantendo as outras saídas em nível lógico baixo.

4.1.7.4 SELFTEST_REG

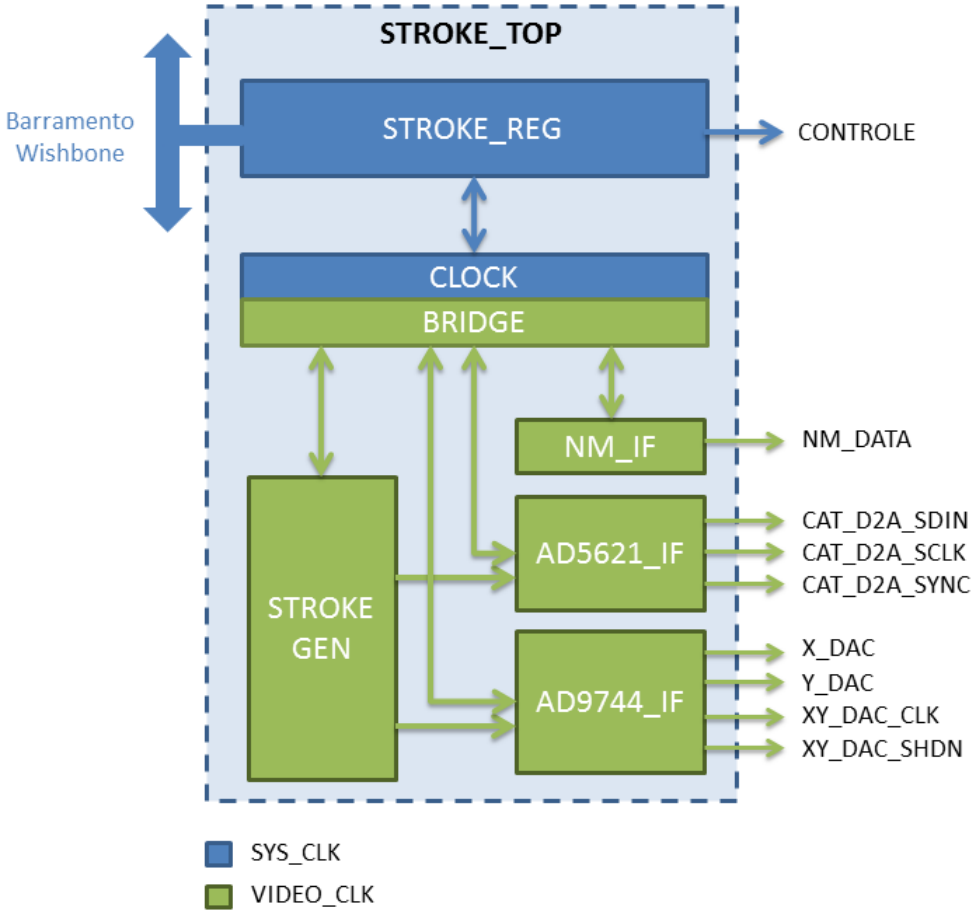
O bloco de registradores *Wishbone* é utilizado para configurar os transceptores dos canais de comunicação (habilitação do receptor, do transmissor e da resistência terminal, por exemplo), além de ser utilizado para controlar o funcionamento do bloco SELFTEST_PRBS (habilitação de cada gerador de PRBS, habilitação e reinicialização dos detectores). Os sinais de status do bloco *SELFTEST_PRBS* podem ser lidos pelo processador de testes através dos registradores, que indicam se os detectores estão (ou já estiveram, depois da última reinicialização) sincronizados, se houve algum erro e se o canal está recebendo apenas nível lógico baixo ou alto.

Através dos registradores, o processador seleciona o valor que deve ser lido através do bloco BIT_ADC. A escrita neste registrador gera o pulso de habilitação para o bloco PULSO_GEN. Quando o bloco BIT_ADC indica que a leitura foi efetuada, através de um pulso no sinal SPI_DONE, o bloco SELFTEST_REG registra o valor da entrada SPI_DADO, que poderá então ser lido pelo processador de testes, que irá avaliar se o valor lido está dentro de um limite aceitável. O processador de testes também pode monitorar o registrador de status proveniente do bloco SELFTEST_PWR para saber quando está sendo enviado um pulso e quando o bloco PULSO_GEN está disponível.

4.1.8 STROKE

O bloco STROKE_TOP é estrutural, servindo para interligar os blocos internos STROKE_REG, CLOCK_BRIDGE, STROKE_GEN, NM_IF, AD5621_IF e AD9744_IF. Seu diagrama de blocos é apresentado na Figura 21, onde pode-se observar que o bloco STROKE_TOP opera em dois domínios de *clock* distintos.

Figura 21 – Diagrama de Blocos de STROKE.



Fonte: A Autora.

4.1.8.1 CLOCK_BRIDGE

O bloco CLOCK_BRIDGE é necessário para fazer o cruzamento entre os dois domínios de *clock* presentes no bloco STROKE_TOP: SYS_CLK e VIDEO_CLK. Como explicado na subseção 2.1.4, quando não é respeitado o tempo de *setup* ou o tempo de *hold* de um *flip-flop*, sua saída pode se tornar imprevisível até a próxima borda de *clock*. Para evitar este problema, cada sinal enviado do domínio SYS_CLK para o domínio VIDEO_CLK passa por dois *flip-flops* cascateados que amostram o sinal na borda de subida de VIDEO_CLK (quando VIDEO_CLK_EN estiver habilitado). Da mesma forma, cada sinal enviado do domínio VIDEO_CLK para o domínio SYS_CLK passa por dois *flip-flops* cascateados que amostram o sinal na borda de subida de SYS_CLK.

Além dos registradores para cruzamento de domínio, existe uma RAM Dual Port (com duas interfaces, uma de escrita e uma de leitura) no bloco CLOCK_BRIDGE. Esta RAM foi colocada neste bloco pois ela é escrita pelo processador de testes através do bloco

de registradores, que opera no domínio de SYS_CLK, e é lida pelo bloco STROKE_GEN, que opera no domínio VIDEO_CLK. A RAM instanciada tem granularidade de 32 bits e endereçamento em 14 bits (16384 endereços). Para o cálculo do tamanho da RAM, considerou-se que para gerar um padrão como o mostrado na Figura 23 com a resolução máxima de 1280 por 1024, seria suficiente 16384 pontos (dois na potência 14), pois três quadrados do tamanho máximo podem ser representados por 13824 pontos.

A interface de escrita da RAM é formada por uma entrada de dados de 32 bits, uma entrada de endereço de 14 bits, uma entrada de *clock* (conectada a SYS_CLK), uma entrada de habilitação do *clock* (conectada a nível lógico alto) e uma entrada de habilitação de escrita (WEN, ou *write enable*). Todos os sinais desta inteface, exceto o SYS_CLK, são provenientes do bloco STROKE_REG. A interface de leitura da RAM é formada por uma saída de dados de 32 bits, uma entrada de endereço de 14 bits, uma entrada de *clock* (conectada a VIDEO_CLK), uma entrada de habilitação do *clock* (conectada a VIDEO_CLK_EN) e uma entrada de habilitação de leitura (REN, ou *read enable*). Todos os sinais da inteface de leitura, exceto o VIDEO_CLK e o VIDEO_CLK_EN, são provenientes do bloco STROKE_GEN (especificamente do bloco READ_CTRL).

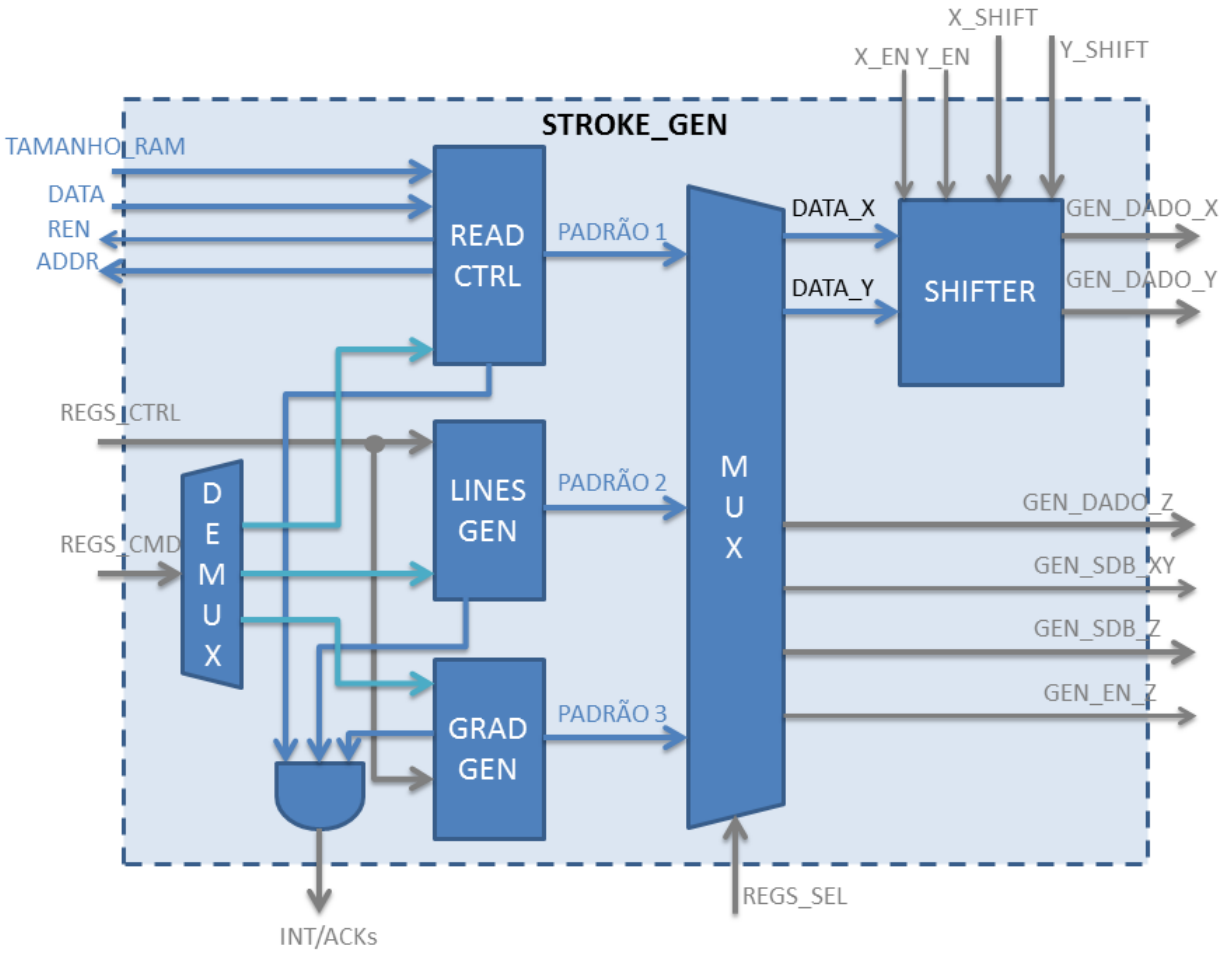
Futuramente, quando o bloco NM_IF for implementado, serão instanciadas outras três RAMs no bloco CLOCK_BRIDGE de mesmo tamanho, cujas interfaces de escrita também serão controladas pelo bloco STROKE_REG, e cujas interfaces de leitura serão controladas pelo bloco NM_IF. Os sinais de *clock* e *clock enable* serão os mesmos da RAM já instanciada.

4.1.8.2 STROKE_GEN

O diagrama do bloco STROKE_GEN é apresentado na Figura 22. O bloco READ_CTRL é responsável por controlar a leitura da memória RAM, convertendo os dados lidos em um padrão *stroke*. Os blocos LINES_GEN e GRAD_GEN são responsáveis por gerar internamente padrões *stroke*. O multiplexador é utilizado para que o processador de testes selecione o padrão *stroke* que será enviado aos blocos que fazem interface com os conversores (AD5621_IF e AD9744_IF). O demultiplexador é utilizado para enviar os comandos de início e fim de transmissão apenas para o bloco responsável pelo padrão selecionado (utiliza-se o mesmo seletor do multiplexador). O bloco SHIFTER é responsável por tornar um padrão estático móvel, adicionando um valor fixo nos eixos habilitados, que podem ser X ou Y (função adicional às que foram solicitadas nos requisitos).

Os dois blocos de geração de padrão interno recebem dados de controle dos regis-

Figura 22 – Diagrama de Blocos de STROKE_GEN.



Fonte: A Autora.

tradores com informações como a orientação do padrão (vertical ou horizontal) e a área a ser utilizada (valores iniciais e finais dos eixos X e Y). Os comandos para início e fim de transmissão de um padrão são enviados apenas a um bloco por vez. Como estes comandos são gerados pelo bloco STROKE_REG, que opera no domínio SYS_CLK, são enviados sinais de confirmação (ACK) como resposta a cada comando. Estes sinais de ACK são utilizados como entrada em um bloco OU lógico, juntamente com os sinais de interrupção que são gerados ao fim do envio de cada quadro.

O bloco READ_CTRL contém a máquina de estados responsável por ler a memória RAM e decodificar os comandos escritos, gerando assim o padrão 1 de teste. O processador, através dos registradores, pode escrever um comando de posição ou de intensidade, definido pelo bit menos significativo do dado escrito. Quando se trata de comandos de posição, a máquina apenas extrai os dados dos eixos X e Y, enviando-os para o bloco AD9744_IF, e no ciclo de *clock* seguinte faz uma nova leitura da memória, incrementando

o endereço. Quando se recebe um comando de intensidade, além de extrair o dado do eixo Z, a máquina aciona uma transmissão SPI, feita pelo bloco AD5621_IF, e aguarda a indicação, proveniente deste bloco, de que a transmissão foi finalizada. Somente após o recebimento desta indicação a máquina fará uma nova leitura da RAM. O tamanho do padrão é indicado ao bloco READ_CTRL através de um registrador, evitando assim que a máquina leia dados inválidos da memória.

O bloco LINES_GEN é formado por uma máquina de estados que gera linhas horizontais ou verticais em padrão *stroke* (padrão 2). Para isso, quando habilitada, a máquina executa os seguintes passos:

1. posiciona o cursor na posição inicial;
2. aciona o eixo Z (intensidade);
3. a cada ciclo de *clock*, incrementa em uma unidade a posição do eixo 1, que pode ser X ou Y, dependendo do sentido selecionado, até seu valor limite;
4. desliga do eixo Z;
5. incrementa em um valor definido pelos registradores a posição do eixo 2 (ortogonal ao eixo 1);
6. retorna o cursor do eixo 1 para a sua posição inicial;
7. repete os passos a partir do passo 2 até que se chegue no limite do eixo 2.

O bloco GRAD_GEN é formado por uma máquina de estados que gera um gradiente horizontal ou vertical em padrão *stroke* (padrão 3). Para isso, quando habilitada, a máquina executa os seguintes passos:

1. posiciona o cursor na posição inicial e atribui intensidade máxima a Z;
2. aciona o eixo Z;
3. a cada ciclo de *clock*, incrementa em uma unidade a posição do eixo 1, que pode ser X ou Y, dependendo do sentido selecionado, até seu valor limite;
4. desliga do eixo Z;
5. incrementa em uma unidade a posição do eixo 2 (ortogonal ao eixo 1);

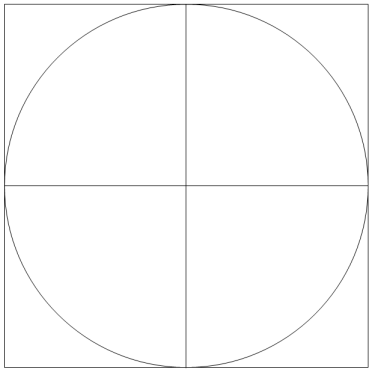
6. decrementa o valor do eixo Z, a partir do último valor ligado, em um valor definido através de um registrador;

7. retorna o cursor do eixo 1 para a sua posição inicial;

8. repete os passos a partir do passo 2 até que se chegue no limite do eixo 2.

Os três padrões, se habilitados, são redesenhados a cada borda de subida do sinal VSYNC (*Vertical Synchronization*, ou Sincronização Vertical), ou seja, os quadros são enviados na mesma taxa de VSYNC, que é de 50 Hz para HMDs do tipo DASH e de 60 Hz para HMDs do tipo JHMCS.

Figura 23 – Exemplo de Padrão *Stroke* de Teste.



Fonte: A Autora.

4.1.8.3 NM_IF

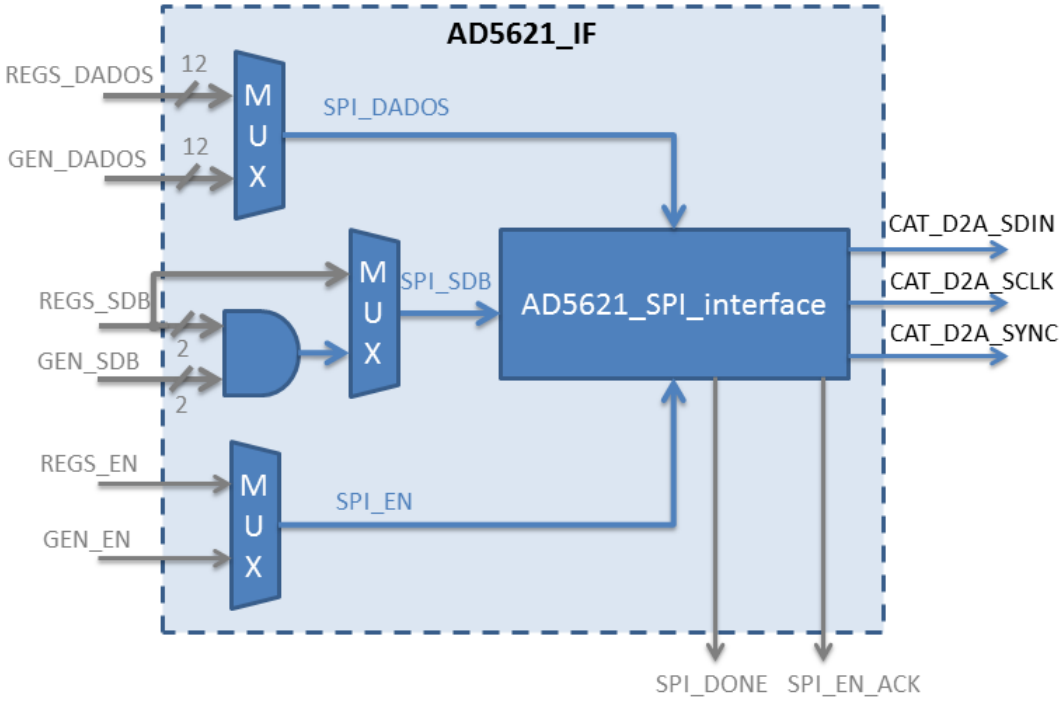
A estrutura do bloco NM_IF será um pouco mais simples do que a do bloco STROKE_GEN, pois possuirá apenas um bloco para ler as memórias RAMs correspondentes. A seleção do padrão de teste a ser lido será feita por demultiplexadores conectados às saídas de controle da leitura das RAMs e por multiplexadores que selecionarão as entradas, provenientes das RAMs. Seu funcionamento também irá diferir na maneira como os dados devem ser enviados à interface, cujo funcionamento é propriedade intelectual do cliente. Este bloco será implementado na próxima etapa do projeto.

4.1.8.4 AD5621_IF

Este bloco é formado por um multiplexador registrado de doze bits, um multiplexador registrado de um bit, um multiplexador registrado de dois bits, uma porta OU e um

bloco que faz a interface com o conversor AD5621 (AD5621_SPI_interface). Na Figura 24 pode-se observar o diagrama de blocos do AD5621_IF.

Figura 24 – Diagrama de Blocos de AD5621_IF.



Fonte: A Autora.

O multiplexador de doze bits permite que se selecione a fonte dos dados que serão enviados ao conversor (do bloco STROKE_GEN ou do bloco STROKE_REG). O multiplexador de dois bits permite que se selecione a fonte dos bits de *shutdown*, que pode ser diretamente do bloco STROKE_REG ou a saída da operação lógica OU entre os bits de *shutdown* do bloco STROKE_REG e os do bloco STROKE_GEN. O multiplexador de um bit seleciona a fonte, entre o bloco STROKE_REG e o bloco STROKE_GEN, que dispara uma operação de transmissão SPI. O seletor é o mesmo para todos os multiplexadores, sendo proveniente do bloco de registradores.

O bloco interno AD5621_SPI_interface consiste em uma máquina de estados que controla o ciclo de transmissão SPI, atuando como mestre, e um ODDR para gerar o *clock* da interface SPI, (instanciado da mesma forma utilizada no bloco CLOCK_SYS, gerando 27 MHz na saída). Além disso, este bloco controla o sinal de ACK enviado ao bloco de registradores, correspondente ao sinal SPI_EN.

A máquina de estados permanece em seu estado inicial até receber uma requisição

de transmissão SPI (através do sinal SPI_EN), quando os dados e os bits de *shutdown* são registrados em um sinal de dezesseis bits (doze bits de dados, dois bits de *shutdown* e dois zeros lógicos). Então, a máquina habilita o escravo SPI, através da saída CAT_D2A_SYNC, e envia os dados pela saída CAT_D2A_SDIN, fazendo uma operação de deslocamento a cada borda de subida do *clock* durante dezesseis ciclos, como mencionado na subseção 2.2.5. Após o fim da transmissão, a máquina desabilita o escravo e indica que terminou a transmissão (indicação utilizada pelo bloco STROKE_GEN).

4.1.8.5 AD9744_IF

Este bloco é semelhante ao bloco AD5621_IF, porém não possui um bloco específico para a interface SPI, pois a interface do conversor AD9744 é paralela. O bloco AD9744_IF é formado por dois multiplexadores registrados de doze bits cada (um para o conversor do eixo X, outro para o conversor do eixo Y), um multiplexador de um bit e uma porta OU para a lógica do bit de *shutdown* e um ODDR para a geração do *clock* do conversor (instanciado da mesma forma utilizada no bloco CLOCK_SYS, gerando 27 MHz na saída).

Os dois multiplexadores de doze bits permitem que se selecione a fonte dos dados que serão enviados ao conversor (do bloco STROKE_GEN ou do bloco STROKE_REG). O multiplexador de um bit permite que se selecione a fonte do bit de *shutdown*, que pode ser diretamente do bloco STROKE_REG ou a saída da operação lógica OU entre o bit de *shutdown* do bloco STROKE_REG e o do bloco STROKE_GEN. Desta maneira, se necessário, o processador de testes pode desabilitar os conversores enquanto o envio de dados estiver habilitado.

4.1.8.6 STROKE_REG

Este é o bloco de registradores *Wishbone*, utilizado para configurar as saídas de controle, como ganho e *offset*, por exemplo, e para selecionar o modo de operação entre automático e manual. No modo manual, os registradores são utilizados para enviar dados aos blocos AD9744_IF e AD5621_IF, gerando assim padrões *stroke*. Além disso, o processador escreve o padrão 1 na memória RAM através dos registradores (um registrador para o dado, um para o endereço onde este dado deve ser escrito e outro para indicar o tamanho total do padrão 1). Em modo automático, os registradores também são utilizados para selecionar o padrão a ser enviado entre os padrões 1, 2 e 3, além de permitir a configuração dos padrões 2 e 3 (orientação e distância entre linhas, por exemplo).

Todos os registradores que geram comandos automáticos que são enviados aos

blocos que operam no domínio de VIDEO_CLK permanecem habilitados até receberem a resposta correspondente (ACK). Garante-se, assim, que estes comandos permaneçam habilitados tempo suficiente para que sejam recebidos pelos blocos de destino, e que sejam desabilitados após seu processamento.

4.1.9 UART_DEBUG

Para realizar a função deste bloco, foi utilizado um *core* de propriedade intelectual da empresa desenvolvedora do projeto. O bloco possui uma UART, conversores de comandos e um conversor para mestre *Wishbone*.

4.2 Simulação

Para cada bloco de topo, foi feito um *test bench* (ou bancada de testes), que consiste em um arquivo VHDL não implementável, utilizado apenas para simulação. Estes arquivos contém uma sequência de testes a ser executada, baseada nas funções que o bloco deve implementar. Foram utilizados *scripts* TCL (*Tool Command Language*, ou Linguagem de Comando de Ferramentas) para compilar os blocos internos e os *test benches* e iniciar a execução das simulações.

Foram utilizados *cores* da empresa onde o projeto é desenvolvido nos *test benches* para geração de *clocks* e *resets* e para a simulação de mestres *Wishbone*. Foi necessário também criar alguns blocos para simulação, como os escravos SPI, por exemplo. Parte das verificações é feita de forma automática pelos *test benches*, através do uso de instruções *assert*, que retornam mensagens de erro no *console* da ferramenta de simulação caso a condição especificada não seja satisfeita, e parte da verificação é feita pelo usuário, através da inspeção visual das formas de onda geradas.

As simulações de grande parte dos blocos foram feitas utilizando-se a ferramenta Modelsim ALTERA Starter Edition 10.0c. A simulação do bloco CLOCK_SYS_TOP foi feita na ferramenta Vivado 2014.1, da Xilinx, pois este bloco contém componentes específicos do FPGA da Xilinx utilizado. Para o bloco STROKE_TOP, criou-se um bloco para simular o funcionamento dos ODDRs, pois, quando este bloco foi desenvolvido, a ferramenta da Xilinx ainda não estava disponível.

Visto que este trabalho é focado no desenvolvimento da lógica programável, será descrito apenas o *test bench* do bloco BIT_ADC_TOP. Os *test benches* dos outros blocos

foram feitos de maneira semelhante, porém contendo as adaptações necessárias para testar cada bloco.

4.2.1 BIT_ADC_TB

O *testbench* do bloco BIT_ADC_TOP contém um gerador de *clock*, um gerador de *reset*, um escravo SPI e um mestre *Wishbone*, além do próprio BIT_ADC_TOP. O *test bench* foi dividido em quatro casos de teste (*test cases*):

- operação em modo automático para os multiplexadores 1 e 2 e seu conversor AD;
- operação em modo automático para os multiplexadores 3 e 4 e seu conversor AD;
- operação em modo manual para os multiplexadores 1 e 2 e seu conversor AD;
- configuração das saídas discretas.

No primeiro teste, primeiramente o bloco é configurado para operar em modo automático, através dos registradores. Em seguida, o teste automático dos multiplexadores 1 e 2 é requisitado. O escravo SPI, quando selecionado, envia dados de 16 bits, que são carregados pelo *test bench*. Neste teste, ocorrem doze ciclos de recepção, então são carregados valores de 0 a 11 no escravo, incrementando-se ao término de cada recepção. Após o término das operações, os registradores de status e de dados são verificados para garantir que os dados foram corretamente recebidos. A verificação dos valores dos registradores é feita automaticamente, porém os *enables* e seletores dos multiplexadores devem ser verificados por inspeção visual da forma de onda gerada, que é apresentada na Figura 25.

No segundo teste, são repetidos os passos do primeiro teste, desta vez para os multiplexadores 3 e 4, alterando-se o número de ciclos de transmissão para 15. No terceiro teste, o bloco BIT_ADC_TOP é configurado para operar em modo manual, e são solicitadas duas leituras através da interface com o bloco SELFTEST. Este teste é todo verificado automaticamente. No último teste, são configurados, através dos registradores, os valores das saidas discretas, que são verificados automaticamente. Na Figura 26, é apresentado o *console* da ferramente Modelsim durante a execução desses testes.

4.3 Síntese

Para a síntese do topo do *design* e geração do arquivo de gravação (*bitstream*), foi criado um *script* TCL, que deve ser executado pela ferramenta Vivado (neste caso,

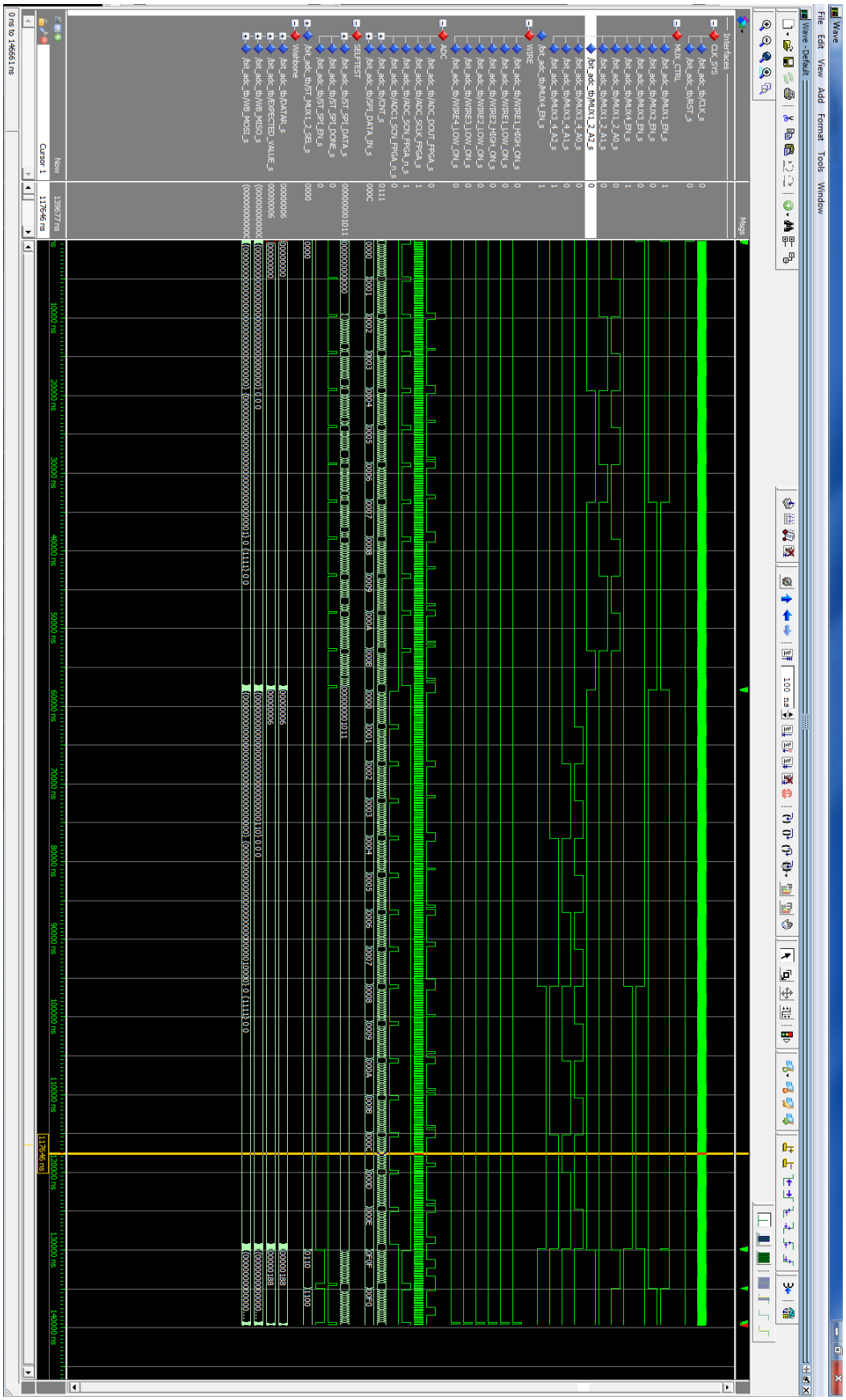
foi utilizado o Vivado 2014.1), o que pode ser feito tanto diretamente pelo *console* da ferramenta quanto por janela de comando do Windows. O *script* foi feito com base nos dados fornecidos pela Xilinx em seus Guias do Usuário (XILINX, 2013c) (XILINX, 2013d).

Primeiramente, são lidos todos os arquivos VHDL que fazem parte do projeto, através do comando *read_vhdl*. Em seguida, são definidos os valores dos registradores do bloco ID e é chamado um outro *script* TCL, previamente desenvolvido na empresa responsável pelo projeto, que atualiza os valores destes registradores. Então, é lido o arquivo de *constraints* (restrições), que deve estar no formato XDC.

No arquivo XDC foram primeiramente definidos o tipo, a localização e a tensão de cada porta do bloco *top level*, através do uso do comando *set_property*. Em seguida, foram especificadas as frequências de cada um dos *clocks*, necessárias para a verificação do *design*, pelo comando *create_clock*. Também foi criado um grupo de *clock*, para que a ferramenta ignore caminhos com cruzamento de domínio de *clock*, onde sabe-se que os tempos de *setup* e *hold* dos *flip-flops* são eventualmente violados, e foi atribuída a propriedade *ASYNC_REG* aos *flip-flops* de sincronização, o que faz a ferramenta tentar localizar estes registradores no mesmo *slice* do FPGA, reduzindo assim a incidência de metaestabilidade (XILINX, 2014).

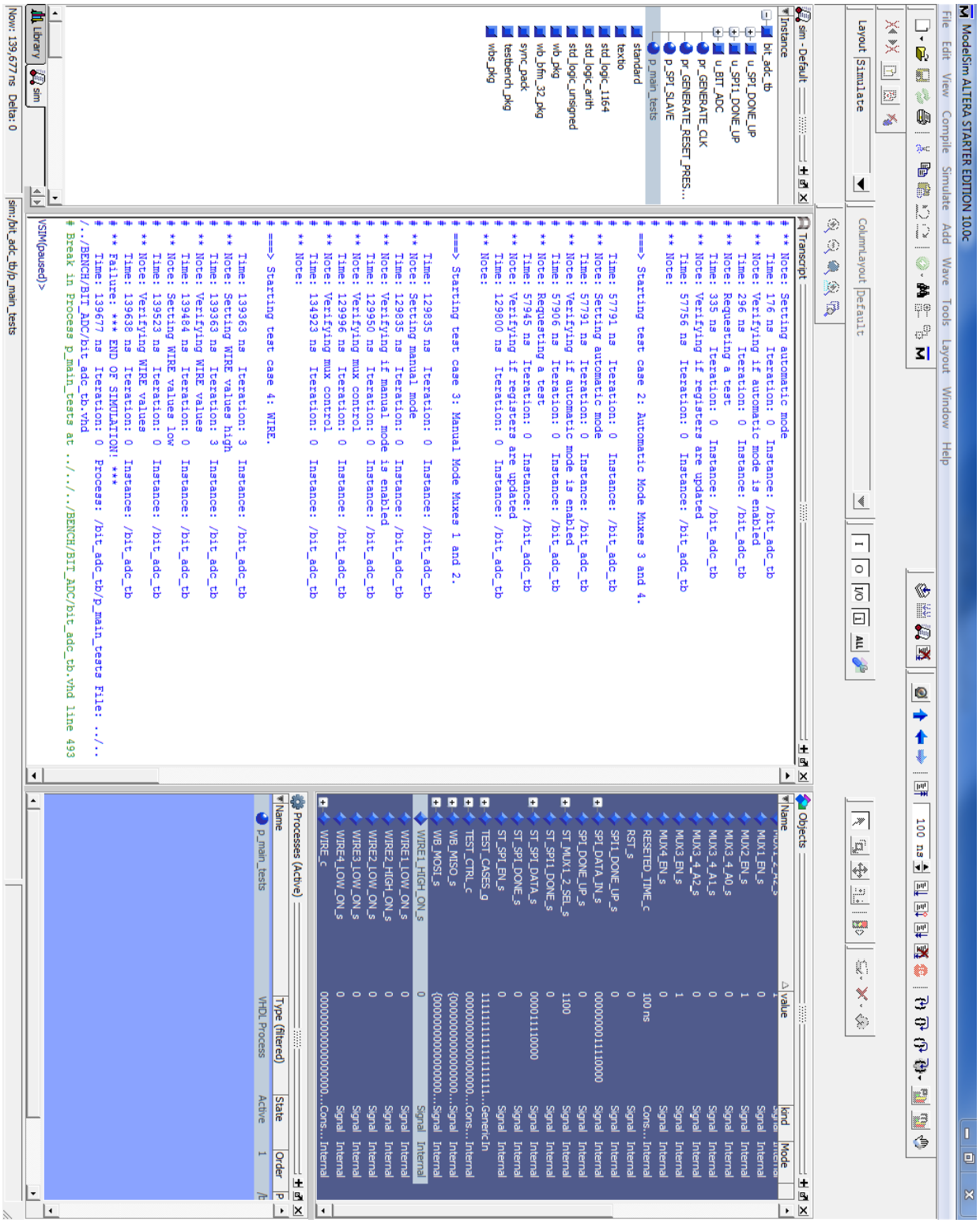
Após o término da leitura do arquivo XDC, o bloco é sintetizado, a partir do comando *synth_design*, e são gerados os relatórios correspondentes a esta etapa. É executada então a otimização do *design*, feita a partir do comando *opt_design*, onde são feitas a propagação de constantes e a remoção de instâncias não conectadas, por exemplo. Então, as portas e instâncias são localizadas no FPGA, pelo comando *place_design*. Se houver alguma violação de temporização após esta etapa, é executada a otimização física, onde algumas células podem ser replicadas para diminuir o *fanout*, melhorando a temporização, por exemplo. Então são gerados os relatórios pós-colocação, é feito o roteamento, através do comando *route_design*, e são gerados os relatórios finais. Por fim, é gerado o arquivo *bitstream* de gravação.

Figura 25 – Forma de Onda Gerada pelo *Test Bench* do Bloco BIT_ADC.



Fonte: A Autora.

Figura 26 – *Console* do Modelsim Durante a Simulação do Bloco BIT_ADC.



Fonte: A Autora.

5 Resultados

5.1 Simulações

Através das simulações, foram testados todos os requisitos de funcionamento de cada um dos blocos. Alguns dos blocos tiveram que ser retrabalhados durante a etapa de desenvolvimento da simulação, por não estarem apresentando o comportamento desejado, devido a problemas na descrição do *hardware* em VHDL. Todos os blocos, após estarem funcionando corretamente nas simulações, foram submetidos à revisão de outro engenheiro para que fossem detectados eventuais erros não percebidos pelo desenvolvedor do código VHDL e para que fossem indicadas possíveis melhorias no código. Após passar por estas últimas correções, cada bloco foi integrado ao bloco do topo. Quando foi finalizado o desenvolvimento dos blocos individualmente, passou-se à verificação da síntese.

5.2 Síntese, *Place* e *Route*

O *script* TCL foi executado com sucesso, sem apresentar erros ou alertas críticos (*critical warnings*). Não foi inferido nenhum *latch*, o que seria um sinal de problemas no código VHDL. O relatório de utilização dos recursos do FPGA, gerado após o *place*, mostra que foi utilizado muito menos de 75% dos recursos do FPGA, conforme apresentado na Tabela 4.

Tabela 4 – Utilização dos Recursos do FPGA.

Recurso	Utilizado	Disponível	Utilização (%)
Bonded IOB	145	400	36,25
Slice	1012	33450	3,02
Slice LUTs	1950	133800	1,45
Slice Registers	2826	267600	1,05
F7 *Muxes*	35	66900	0,05
Block RAM	13	365	3,56
OLOGIC	3	400	0,75
BUFGCTRL	4	32	12,70
PLLE2_ADV	1	10	10

Fonte: A Autora.

Os recursos mais utilizados foram os blocos de entrada e saída (IOB, ou *Input/Output Block*), que correspondem às portas do bloco do topo. Pode-se notar que está sendo utilizado menos de 2% das células lógicas (LUTs) e dos registradores. A ocupação de memórias RAM foi a prevista, referente ao bloco STROKE_GEN. Foram utilizadas três OLOGICs, que correspondem aos três ODDRs (dois do bloco STROKE_TOP e um do bloco CLOCK_SYS), e um PLLE2_ADV, conforme o esperado. Foram utilizados quatro BUFGCTRLs, que correspondem ao BUFGMUX de VIDEO_CLK, ao BUFG de REF_CLK, à saída do PLL (SYS_CLK) e à realimentação do PLL. Como pode-se notar, os BUFGs correspondentes aos *clocks* da *Ethernet* foram eliminados, já que não são conectados a nenhum bloco interno.

Em relação à temporização, os tempos de *setup* e *hold* de todos os registradores foram respeitados (exceto, é claro, dos registradores de sincronização de domínio de *clock*, mas como já mencionado na seção 4.3, eles são ignorados nestes relatórios). Isso foi possível graças à otimização física, pois os relatórios pós-colocação indicam violações de temporização.

O relatório dos domínios de *clock* considerados pela ferramenta após a otimização e o roteamento é mostrado na Tabela 5. A relação dos piores tempos de *setup* e *hold* encontra-se na Tabela 6. Pode-se notar que as menores folgas foram no domínio de SYS_CLK, pois é o *clock* mais utilizado no *design*, fazendo seu *fanout* ser alto, e, portanto, seu tempo de roteamento aumentar.

Tabela 5 – Relatório dos Domínios de *Clock* do FPGA.

Clock	Período (ns)	Frequência (MHz)
CLK_100MHZ_i	10,000	100,000
clkfbout	10,000	100,000
clkout0	6,667	150,000
ETH_RX_CLK_i	40,000	25,000
ETH_TX_CLK_i	40,000	25,000
FPGA_27M_IN_i	37,000	27.027
FPGA_54M_IN_i	18,500	54.054

Fonte: A Autora.

Na Tabela 5, pode-se observar que, apesar de só ter sido definida a restrição para a frequência da entrada do PLL (CLK_100MHZ_i), a ferramenta inferiu restrições para o *clock* da realimentação do PLL, clkfbout, e para a saída utilizada do PLL, clkout0.

Tabela 6 – Caminhos de *Clock* com Menor Folga de Tempo do FPGA.

Domínio de *Clock*	Menor Folga de *Setup* (ns)	Menor Folga de *Hold* (ns)
clkout0	0,904	0,013
FPGA_27M_IN_i	11,822	0,088
FPGA_54M_IN_i	2,572	0,088
clkout0	0,588	0,458

Fonte: A Autora.

5.3 Próximas Etapas

A próxima etapa do projeto a ser realizada são os testes em *hardware*, que serão feitos pelo cliente. Estes testes serão feitos com um PC, utilizando a interface UART (bloco UART_DEBUG) para acessar os registradores *Wishbone*. Para isso, paralelamente aos blocos de VHDL, foram desenvolvidos *scripts* em Lua para auxiliar na depuração. Durante esta etapa de testes, será prestado suporte ao cliente e poderão ser realizadas algumas modificações no código, se solicitadas. Com o término destes testes, serão considerados validados os blocos desenvolvidos nesta etapa, e então será iniciada a fase dois do projeto.

Na segunda fase, primeiramente, serão definidos os requisitos dos blocos VIDEO_IF e COMM. Em seguida, a arquitetura destes blocos será projetada, assim como dos blocos COMM_PROC, TEST_PROC, WB_BRIDGE, REGISTER_FILE e NM_IF. Então, os blocos serão desenvolvidos, simulados, sintetizados e testados em *hardware*. Nesta fase, também deve ser desenvolvido o *firmware* do processador de comunicação.

Na terceira e última fase, após todos os blocos de VHDL terem sido validados, será desenvolvido o *firmware* responsável pela execução dos testes. Os *firmwares* dos dois processadores utilizados serão carregados através da interface SPI que acessa uma memória *Flash* externa.

5.4 Possíveis Melhorias

Em relação ao desenvolvimento técnico do projeto, uma das melhorias propostas seria replicar o sincronizador de *reset* de SYS_RST, criando *resets* locais. Assim, seria possível diminuir a carga total deste sinal, melhorando sua temporização.

Uma das possíveis melhorias no planejamento das etapas deste projeto seria a redução do tempo entre o fim do desenvolvimento de um bloco e de seu *testbench* e a sua revisão, o desenvolvimento dos *scripts* em Lua e a execução dos testes em *hardware* correspondentes. Se estas etapas fossem executadas sequencialmente, seria possível uma maior interação entre os desenvolvedores e uma redução do tempo de desenvolvimento e de retrabalho.

Além disso, seria interessante poder acompanhar pessoalmente os testes em *hardware* do cliente, para auxiliar o cliente na compreensão do funcionamento dos blocos de registradores *Wishbone* e para depurar possíveis problemas.

6 Conclusão

Neste trabalho, apresentou-se o projeto e o desenvolvimento de parte dos blocos da lógica programável de um equipamento de teste de HMDs, dispositivos utilizados por pilotos para aumentar a segurança das operações das aeronaves, conforme visto no Capítulo 1.

Primeiramente, foram apresentados alguns conceitos essenciais para a compreensão do trabalho e componentes utilizados no projeto. Além disso, foram estudados os principais protocolos utilizados no projeto, como *Wishbone*, utilizado no barramento interno de comunicação entre os blocos de lógica, e SPI, utilizado na comunicação com conversores AD e DA externos.

Em seguida, foi feito um estudo do sistema. Foram apresentados os dois tipos de capacetes que o sistema deve testar, DASH e JHMCS. Na sequência, foi apresentado o diagrama de blocos da lógica a ser implementada, destacando-se os principais requisitos de cada bloco. Então, para os blocos de lógica a serem desenvolvidos nesta etapa do projeto, foi feito um estudo do circuito que cada um destes deve controlar, atentando-se às características dos principais componentes da placa, como conversores, multiplexadores, transceptores e osciladores. Este estudo foi muito importante para o bom desenvolvimento do projeto, pois dele foram extraídas informações essenciais, como limites de *clock*, modos de funcionamento dos componentes, atrasos, entre outras informações.

Então, foi descrita a arquitetura de cada um dos blocos de lógica programável desenvolvidos. Viu-se que no bloco CLOCK_SYS, que controla os *clocks*, resets e bases de tempo, foram utilizados diversos componentes próprios do FPGA, como BUFG, BUFGMUX, ODDR e PLL. Para o desenvolvimento deste bloco, foi muito importante ler os manuais do fabricante e conhecer os recursos disponíveis no FPGA, para aprender a utilizá-los da maneira correta e eficiente. Foi visto também que o uso de um barramento interno (no caso, um barramento *Wishbone*) foi essencial para este projeto, pois possibilitará a comunicação do processador de testes e da UART de depuração com todos os blocos internos.

Até o presente momento, já foi desenvolvida a lógica programável de todos os blocos da primeira fase do projeto. Todos estes blocos, em suas versões finais, apresentaram resultados satisfatórios nas simulações computacionais. Conforme mencionado, estas simulações foram feitas baseando-se nas funções principais de cada bloco. As simulações foram executadas com o auxílio das ferramentas Modelsim e Vivado 2014.1.

A ferramenta Vivado 2014.1 também foi utilizada para realizar a síntese, a colocação e o roteamento. No desenvolvimento do *script*, também foi muito importante o estudo da ferramenta e de seus recursos, pois assim foram descobertas algumas funções anteriormente desconhecidas pela equipe de desenvolvimento, como a possibilidade de se indicar os registradores de sincronização para que eles sejam colocados tão próximos quanto possível, diminuindo a probabilidade de incidência de metaestabilidade.

Observou-se que a lógica desenvolvida cumpriu os requisitos de ocupação do FPGA, mantendo-se bem abaixo do limite de 75% estipulado, e que os componentes próprios do FPGA foram inferidos corretamente pela ferramenta. Foi visto também que os requisitos de tempo foram cumpridos, porém com pouca folga no domínio SYS_CLK, devido ao grande número de blocos operando neste domínio de *clock*.

Durante o planejamento e a execução deste projeto, foram utilizados diversos conceitos vistos no curso de Graduação em Engenharia Elétrica, possibilitando que estes fossem aprofundados, estudando-se sua aplicação prática. Também foi possível acompanhar diversas etapas essenciais para projetos desenvolvidos no ambiente empresarial, desde a elaboração dos requisitos juntamente com o cliente, o estudo do sistema, o projeto da arquitetura e o desenvolvimento dos blocos e das simulações. Até o fim da primeira fase do projeto, ainda serão vistos aspectos da etapa de testes e depuração. Na próxima fase do projeto, já poderão ser aplicadas as melhorias propostas, visando ao aumento da eficiência do grupo e da qualidade do projeto.

Referências

ALTERA CORPORATION. *Understanding Metastability in FPGAs - v1.2*. 2009. Disponível em: <http://www.altera.com/literature/wp/wp-01082-quartus-ii-metastability.pdf>. Acesso em: 19.05.2014. Citado na página 26.

ANALOG DEVICES. *AD7910/AD7920 REV. C*. 2005. Disponível em: <http://www.analog.com/static/imported-files/data_sheets/AD7910_7920.pdf>. Acesso em: 26.03.2014. Citado 2 vezes nas páginas 43 e 45.

ANALOG DEVICES. *AD5601/AD5611/AD5621 (REV. G)*. 2013. Disponível em: <http://www.analog.com/static/imported-files/data_sheets/AD5601_5611_5621.pdf>. Acesso em: 16.04.2014. Citado na página 53.

ANALOG DEVICES. *AD9744 (REV. C)*. 2013. Disponível em: <http://www.analog.com/static/imported-files/data_sheets/AD9744.pdf>. Acesso em: 19.04.2014. Citado na página 52.

ANALOG DEVICES. *ADG758/ADG759 REV. B*. 2013. Disponível em: <http://www.analog.com/static/imported-files/data_sheets/ADG758_759.pdf>. Acesso em: 26.03.2014. Citado 2 vezes nas páginas 45 e 46.

AXELSON, J. *Serial Port Complete: Programming and Circuits for RS-232 and RS-485 Links and Networks*. Lakeview Research, 1998. ISBN 9780965081924. Disponível em: <http://books.google.com.br/books?id=YZqGTLckjLAC>. Citado na página 30.

BADAOUI, H.; FRIGNAC, Y.; FEHAM, M. Pseudo random binary sequences analysis for the modeling of optical dpsk transmission systems. *International Journal of Computer Science and Communication*, India, v. 1, n. 2, p. 369–372, Julho-Dezembro 2010. Citado na página 26.

BERNSTEIN, K. *High Speed CMOS Design Styles*. Springer US, 1998. ISBN 9780792382201. Disponível em: <http://books.google.com.br/books?id=R8KOoWCfh2cC>. Citado 2 vezes nas páginas 42 e 56.

BLANK, A. *TCP/IP Foundations*. Wiley, 2006. ISBN 9780782151138. Disponível em: <http://books.google.com.br/books?id=OB-k4OIiTxIC>. Citado na página 27.

CATSOULIS, J. *Designing Embedded Hardware*. O'Reilly Media, 2005. ISBN 9781449379032. Disponível em: <http://books.google.com.br/books?id=s55-xmGMGBAC>. Citado na página 33.

COLLINSON, R. *Introduction to Avionics Systems*. Springer, 2011. (SpringerLink : Bücher). ISBN 9789400707085. Disponível em: <http://books.google.com.br/books?id=aU8SMhzrScgC>. Citado na página 21.

COPPOLA, M. et al. *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC*. Taylor & Francis, 2008. (System-on-Chip Design and Technologies). ISBN 9781420044720. Disponível em: <http://books.google.com.br/books?id=SjEkzn65GXYC>. Citado na página 31.

DALLAS SEMICONDUCTOR. *Application Note 83: Fundamentals of RS-232 Serial Communications*. 1998. Disponível em: <http://www.fte.com/docs/max232.pdf>. Acesso em: 01.06.2014. Citado na página 30.

IEEE-SA. *IEEE Std 802.3^{TM}- 2012: IEEE Standard for Ethernet (Section 1)*. 2012. Disponível em: <http://standards.ieee.org/getieee802/download/802.3-2012_section1.pdf>. Acesso em: 08.05.2014. Citado na página 27.

IEEE-SA. *IEEE Std 802.3^{TM}- 2012: IEEE Standard for Ethernet (Section 2)*. 2012. Disponível em: <http://standards.ieee.org/getieee802/download/802.3-2012_section2.pdf>. Acesso em: 08.05.2014. Citado na página 27.

IEEE-SA. *IEEE Std 802.3^{TM}- 2012: IEEE Standard for Ethernet (Section 3)*. 2012. Disponível em: <http://standards.ieee.org/getieee802/download/802.3-2012_section3.pdf>. Acesso em: 08.05.2014. Citado na página 27.

KUGELSTADT, T. *SIGNAL CHAIN BASICS (Part 31): Digital interfaces – The SPI Bus*. 2009. Disponível em: <http://www.eetimes.com/document.asp?doc_id=1272534>. Acesso em: 24.03.2014. Citado na página 35.

LINEAR TECHNOLOGY CORPORATION. *LTC2854/LTC2855 REV. A*. 2007. Disponível em: <http://cds.linear.com/docs/en/datasheet/285455fb.pdf>. Acesso em: 13.04.2014. Citado na página 49.

LOXTON, R.; POPE, P. *Instrumentation: A Reader: A Reader*. Springer US, 1986. (Open University). ISBN 9780412534003. Disponível em: <http://books.google.com.br/books?id=RwsoQbHYjvwC>. Citado na página 30.

MICHELI, G. D.; BENINI, L. *Networks on Chips: Technology and Tools*. Elsevier Science, 2006. (Systems on Silicon). ISBN 9780080473567. Disponível em: <http://books.google.com.br/books?id=IHHTmSBcoGIC>. Citado na página 39.

MOTOROLA, I. *SPI Block Guide V04.01*. 2004. Disponível em: <http://read.pudn.com/downloads67/doc/240308/SPI%20SPEC/S12SPIV4.pdf>. Acesso em: 19.03.2014. Citado 2 vezes nas páginas 33 e 35.

NAVABI, Z. *Digital Design and Implementation with Field Programmable Devices*. Kluwer Academic Publishers, 2006. ISBN 9781402080128. Disponível em: <http://books.google.com.br/books?id=cPv4fHOxcVoC>. Citado na página 64.

OPENCORES; SILICORE. *Specification for the: WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*. 2002. Disponível em: <http://cdn.opencores.org/downloads/wbspec_b3.pdf>. Acesso em: 16.03.2014. Citado 2 vezes nas páginas 32 e 34.

RASH, C. *Helmet-mounted Displays: Sensation, Perception, and Cognition Issues*. U.S. Army Aeromedical Research Laboratory, 2009. ISBN 9780615283753. Disponível em: <http://books.google.com.br/books?id=G_aMQwAACAAJ>. Citado 3 vezes nas páginas 37, 38 e 39.

SPITZER, C. *Avionics: Elements, Software and Functions*. Taylor & Francis, 2006. (The Avionics Handbook, Second Edition). ISBN 9780849384394. Disponível em: <http://books.google.com.br/books?id=1gwoomBxPpcC>. Citado na página 21.

STENSBY, J. *Phase-Locked Loops: Theory and Applications*. Taylor & Francis, 1997. (Phase-locked Loops: Theory and Applications). ISBN 9780849394713. Disponível em: <http://books.google.com.br/books?id=Q07rP2xmSnAC>. Citado na página 42.

TEXAS INSTRUMENTS INCORPORATED. *AM26LV31C, AM26LV31I (REV. G)*. 2005. Disponível em: <http://www.ti.com/lit/ds/symlink/am26lv31.pdf>. Acesso em: 13.04.2014. Citado na página 49.

TEXAS INSTRUMENTS INCORPORATED. *AM26LV32C, AM26LV32I (REV. E)*. 2005. Disponível em: <http://www.ti.com/lit/ds/slls202e/slls202e.pdf>. Acesso em: 13.04.2014. Citado na página 49.

TEXAS INSTRUMENTS INCORPORATED. *SN65LVDS179, SN65LVDS180, SN65LVDS050, SN65LVDS051 - HIGH-SPEED DIFFERENTIAL LINE DRIVERS AND RECEIVERS (REV. P)*. 2009. Disponível em: <http://www.ti.com/lit/ds/symlink/sn65lvds179.pdf>. Acesso em: 14.04.2014. Citado na página 50.

TONCICH, D. *Data Communications and Networking for Manufacturing Industries*. Chrystobel Engineering, 1993. ISBN 9780646105222. Disponível em: <http://books.google.com.br/books?id=RIFcwwbj1f8C>. Citado na página 29.

TRIMBERGER, S. *Field-Programmable Gate Array Technology*. Springer US, 1994. ISBN 9780792394198. Disponível em: <http://books.google.com.br/books?id=-jHkdnr8ezIC>. Citado na página 23.

VALLABHANENIA, S. et al. *DESIGN OF AN ALL-DIGITAL PLL (ADPLL) CORE ON FPGA*. 2010. Disponível em: <http://klabs.org/mapld04/abstracts/sharma_a.doc>. Acesso em: 30.03.2014. Citado 2 vezes nas páginas 23 e 24.

WEBSTER, J. *Electrical Measurement, Signal Processing, and Displays*. Taylor & Francis, 2010. (Principles and Applications in Engineering). ISBN 9780203009406. Disponível em: <http://books.google.com.br/books?id=as4ZJBYZ8k8C>. Citado na página 31.

WOODS, R. et al. *FPGA-based Implementation of Signal Processing Systems*. Wiley, 2008. ISBN 9780470713778. Disponível em: <http://books.google.com.br/books?id=O2-RPrJUHJAC>. Citado na página 23.

WYATT, D.; TOOLEY, M. *Aircraft Communications and Navigation Systems*. Taylor & Francis, 2013. ISBN 9781136079023. Disponível em: <http://books.google.com.br/books?id=0zc68hAA5dgC>. Citado na página 21.

XILINX. *Xilinx 7 Series FPGA and Zynq Libraries Guide for HDL Designs - v14.4*. 2012. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/7series_hdl.pdf>. Acesso em: 14.05.2014. Citado 3 vezes nas páginas 25, 42 e 56.

XILINX. *LogiCORE^{TM}IP AXI Ethernet Lite MAC v2.0: Product Guide for Vivado Design Suite*. 2013. Disponível em: <http://www.xilinx.com/support/documentation/ip_documentation/axi_ethernetlite/v2_0/pg135-axi-ethernetlite.pdf>. Acesso em: 01.06.2014. Citado na página 39.

XILINX. *LogiCORE^{TM}IP MicroBlaze^{TM}Micro Controller System - v1.4*. 2013. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals_j/xilinx14_5/pg048-microblaze-mcs.pdf>. Acesso em: 01.06.2014. Citado na página 39.

XILINX. *Vivado Design Suite Tcl Command Reference Guide - v2013.1*. 2013. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug835-vivado-tcl-commands.pdf>. Acesso em: 30.05.2014. Citado na página 77.

XILINX. *Vivado Design Suite User Guide: Using Tcl Scripting - v2013.4*. 2013. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_4/ug894-vivado-tcl-scripting.pdf>. Acesso em: 30.05.2014. Citado na página 77.

XILINX. *UltraFast Design Methodology Guide for the Vivado Design Suite - v2014.1*. 2014. Disponível em: <http://www.xilinx.com/support/documentation/sw_manuals/ug949-vivado-design-methodology.pdf>. Acesso em: 30.05.2014. Citado na página 77.