REM

Sincronização de Múltiplas Câmaras e Controlo de Iluminação Sobre Uma Plataforma FPGA

RELATÓRIO DE ESTÁGIO DE MESTRADO

Ricardo André Martins de Sousa

MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES E REDES DE ENERGIA

UNIVERSIDADE da MADEIRA

A Nossa Universidade

www.uma.pt

outubro | 2014

Sincronização de Múltiplas Câmaras e Controlo de Iluminação Sobre Uma Plataforma FPGA

RELATÓRIO DE ESTÁGIO DE MESTRADO

Ricardo André Martins de Sousa
MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES E REDES DE ENERGIA

ORIENTAÇÃO
Fernando Manuel Rosmaninho Morgado Ferrão Dias

CO-ORIENTAÇÃO
Martin Wäny

Resumo

O objetivo deste projeto foi o de realizar a sincronização de pelo menos quatro câmaras individuais, ajustando dinamicamente o *frame rate* de operação de cada câmara, tendo por base a família de sensores de imagem CMOS NanEye da empresa Awaiba, numa plataforma FPGA com interface USB3. Durante o projeto analisou-se, com a assistência de um supervisor da Awaiba, o sistema *core* de captura de imagem existente, baseado em VHDL. Foi estudado e compreendido o princípio do ajuste dinâmico do *frame rate* das câmaras. Tendo sido então desenvolvido o módulo de controlo da câmara, em VHDL, e um algoritmo de ajuste dinâmico do *frame rate*, sendo este implementado junto com a plataforma de processamento e interface da FPGA. Foi criado um módulo para efetuar a monitorização da frequência de operação de cada câmara, medindo o período de cada linha numa *frame*, tendo por base um sinal de relógio de valor conhecido. A frequência é ajustada variando o nível de tensão aplicado ao sensor com base no erro entre o período da linha medido e o período pretendido. Para garantir o funcionamento conjunto de múltiplas câmaras em modo síncrono foi implementada uma interface *Master-Slave* entre estas.

Paralelamente ao módulo anteriormente descrito, implementou-se um sistema de controlo automático de iluminação com base na análise de regiões de interesse em cada *frame* captada por uma câmara NanEye. A intensidade de corrente aplicada às fontes de iluminação acopladas à câmara é controlada dinamicamente com base no nível de saturação dos pixéis analisados em cada *frame*. Foram desenvolvidas e implementadas variantes do algoritmo de controlo e o seu desempenho foi avaliado em laboratório.

Os resultados obtidos na prática evidenciam que a solução implementada cumpre os requisitos de controlo e ajuste da frequência de operação de múltiplas câmaras. Mostrou ser um método de controlo capaz de manter um erro de sincronização médio de 3,77 μs mesmo na presença de variações de temperatura de aproximadamente 50 °C. Foi também demonstrado que o sistema de controlo de iluminação é capaz de proporcionar uma experiência de visualização adequada, alcançando erros menores que 3% e uma velocidade de ajuste máxima inferior a 1 s.

Palavras-chave:

Sincronização de múltiplas câmaras, teoria de regulação e realimentação, sensores de imagem CMOS, controlo automático de iluminação, iluminação LED, FPGA, interface de alta velocidade USB3.

Abstract

The goal of this project was to achieve synchronization of at least four individual cameras, by dynamically adjusting each camera's operational frame rate, based on Awaiba's NanEye CMOS image sensor family, and an USB3 interfaced FPGA platform. During the project, the existing core system for image capture was analyzed with the assistance of an Awaiba supervisor. The principle of the dynamic adjustment of the cameras frame rate was studied and understood. Having then developed the camera control module, in VHDL, and an algorithm for dynamic adjustment of the frame rate, which is implemented together with the processing platform and the FPGA's interfacing core. A module was created to perform the operational frequency monitoring of each camera, by measuring the length of each line in a frame based on a well-known sampling signal. The frequency is adjusted by varying the voltage level applied to the sensor based on the time error between the measured and the desired line period. To ensure the joint operation of multiple cameras in synchronous mode, a Master-Slave interface was implemented.

In parallel to the above described control system, an automatic lighting control based on the analysis of regions of interest in each frame captured by a camera NanEye, was implemented. The intensity of the current applied to the light sources coupled to the camera is dynamically controlled based on the saturation level of the pixels in each analyzed frame. Variants of the control algorithm were developed and implemented and its performance was evaluated in a laboratory environment.

The results show that in practice the implemented solution meets the requirements for monitoring and adjusting the operational frame rate of multiple cameras. It proved to be a method of control capable of maintaining an average synchronization error of 3,77 μs even in the presence of temperature variations of approximately 50 °C. It was also shown that the lighting control system is able to provide a smooth viewing experience in the presence of changing ambient light conditions, obtaining errors below 3% and a maximum adjustment speed below 1 s.

Keywords:

Synchronization of multiple cameras, control and feedback theory, CMOS image sensors, automatic lighting control, LED lighting, FPGA, high-speed USB3 interface.

Agradecimentos

Gostaria de agradecer a todas as pessoas que direta ou indiretamente contribuíram para que eu pudesse concluir com sucesso este trabalho. Para todos os que participaram na minha vida e que, de uma maneira ou de outra, moldaram a pessoa que sou hoje.

Ao Professor Morgado Dias pelo apoio prestado ao longo da execução deste trabalho, e que pela sua disponibilidade, paciência e dedicação permitiu melhorar e enriquecer a qualidade deste trabalho.

Ao Doutor Martin Wäny, pelo desafio proposto e pela oportunidade de realizar este trabalho num ambiente industrial com toda a sua colaboração e apoio.

A todos os profissionais da Awaiba Lda, e em especial ao Engenheiro Pedro Santos, por toda a paciência, dedicação e atenção dispensada. Gostaria de agradecer toda a sua disponibilidade para partilhar os seus conhecimentos, que foram relevantes no desenvolvimento deste trabalho e pela sua preocupação durante a realização do mesmo.

A todos os meus colegas, e em particular ao Rodolfo Rodrigues, por toda a amizade e companheirismo. Pela troca de ideias que de uma forma ou de outra ajudaram na conclusão e enriquecimento do meu trabalho.

Aos meus pais, que sempre me apoiaram e ajudaram neste percurso. Pelo seu incentivo que me ajudou a chegar a este ponto na minha educação e na minha vida. Sem eles isto tudo não seria possível.

À minha namorada, Neuza Ferreira, de uma maneira especial, por todos os momentos passados, todo o apoio, afeto e atenção ao longo da realização desta tese bem como durante o decorrer de todo o meu percurso académico.

Aos meus irmãos, restante família e amigos por toda a compreensão, amizade e apoio prestado ao longo da minha vida.

Lista de Abreviaturas

ADC – *Analog to Digital Converter*

ADPLL – *All Digital Phase Locked Loop*

APS – *Active Pixel Sensor*

BB-PFD – *Bang-Bang Phase Frequency Detector*

CCD – *Charge Coupled Device*

CFA – *Color Filter Array*

CLB – *Configurable Logic Block*

CMOS – *Complementary Metal-Oxide Semiconductor*

DAC – *Digital to Analog Converter*

DCM – *Digital Clock Manager*

DCO – *Digitally Controlled Oscillator*

DLL – *Delay-Locked Loop*

DMX – *Digital Multiplex*

DN – *Digital Number*

DPS – *Diagram Pixel Sensor*

DRAM – *Dynamic Random Access Memory*

FIFO – *First Input First Output buffer*

FPGA – *Field Programmable Gate Array*

fps – *frames* por segundo

FT – *Função de Transferência*

FVAL – *Frame Valid*

hsync – *Horizontal Synchronization*

IO – *Input Output*

IOB – *Input/Output Block*

LED – *Light Emitting Diode*

LF – *Loop Filter*

LSB – *Least Significant Bit*

LUT – *Look-Up Table*

LVAL – *Line Valid*

LVDS – *Low Voltage Differential Signaling*

OTF – *Optical Transfer Function*

PC – *Personal Computer*

PFD – *Phase Frequency Detector*

PI – *Proporcional Integral*

PID – *Proporcional Integral Diferencial*

PLL – *Phase Locked Loop*

POF – *Plastic Optical Fiber*

PP – *Pixel Period*

PPS – *Passive Pixel Sensor*

PWM – *Pulse Width Modulation*

RAM – *Random Access Memory*

RGB – *Red Green Blue*

ROI – *Region of Interest*

RTL – *Register Transfer Level*

SoC – *System on a Chip*

SPI – *Serial Peripheral Interface*

SR – *Shift Register*

TDC – *Time-to-Digital Converter*

TOF – *Time-of-Flight*

USB – *Universal Serial Bus*

VCO – *Voltage Controlled Oscillator*

VHDL – *VHSIC Hardware Description Language*

vsync – *Vertical Synchronization*

Lista de Símbolos

$Avg(\varphi_{Error})_{ADPLL}$ – Erro de fase médio entre sinais de relógio obtido em simulação utilizando uma ADPLL.

$Avg(\varphi_{Error})_{DisposcopeUSB3}$ – Erro de fase médio entre *frames* na plataforma DisposcopeUSB3.

$Avg(\varphi_{Error})_{NanoUSB2}$ – Erro de fase médio entre *frames* na plataforma NanoUSB2.

$Avg(\varphi_{Error})_{Simulink}$ – Erro de fase médio entre *frames* obtido em simulação para o *Frequency Comparator*.

$Above$ – *Flag* de estado que indica se a correção do nível de iluminação está a ser feita no sentido descendente.

$Above_Aux$ – Sinal $Above$ atrasado por um ciclo de relógio.

$Above_OR$ – Resultado da operação lógica OU aos sinais atrasados da *flag* de estado $Above$.

$Above_dly1$ – Sinal $Above$ atrasado por dois ciclos de relógio.

$Above_dly2$ – Sinal $Above$ atrasado por três ciclos de relógio.

$Below$ – *Flag* de estado que indica se a correção do nível de iluminação está a ser feita no sentido ascendente.

Br – Intensidade luminosa da imagem captada por uma câmara digital.

b – Fator linear obtido a partir da relação entre a luminância detetada e a intensidade de luz emitida pelas lâmpadas.

C – Capacitância de um par de fios de transmissão.

C_{01} – Primeira variável do somatório de correções.

C_{0X} – Variável do somatório de correções com índice X.

C_L – Capacitância equivalente conectada à saída do estágio.

C_{obj} – Nível de contraste alvo.

C_w – Definição de contraste de Weber.

Clk_Cnt – Valor da contagem do período da linha numa *frame* proveniente da câmara local em número de ciclos do sinal de relógio.

Col_Cnt – Contagem das colunas de uma *frame*.

D – Separação entre os fios condutores.

D_{LED} – Palavra de comando escrita no LED para controlar o nível de intensidade luminosa.

d – Diâmetro dos fios condutores.

E – Iluminância incidente numa câmara digital.

End_Col_Addr – Endereço da última coluna da região de interesse.

End_Line_Addr – Endereço da última linha da região de interesse.

$FVAL_XOR_Ref$ – Diferença de fase em módulo entre a *frame* local e a *frame* master dada em número de *clocks*.

$Frame_Diff_Sign$ – Valor da correção do erro de fase das *frames* dada pelo processo frame_diff_eval em número de ciclos do sinal de relógio.

f – Frequência de uma sinusoide em Hz.

f_0 – Constante representando a interceção onde $V_{CC} = 0$.

f_{max} – Frequência de operação máxima do oscilador.

f_{min} – Frequência de operação mínima do oscilador.

f_{out} – Frequência de oscilação de um oscilador em anel.

g – Nível de cinzento (*gray value*) obtido à saída de uma câmara digital para uma dada iluminância incidente.

I – Corrente elétrica que percorre o cabo de alimentação do sensor.

I_{LED} – Corrente que percorre o díodo emissor de luz.

Inf_Thold – Nível de referência inferior utilizado no histograma de imagem.

Ini_Col_Addr – Endereço da primeira coluna da região de interesse.

Ini_Line_Addr – Endereço da primeira linha da região de interesse.

$Jump01_SR$ – Equivalente a C_{01}.

$Jump0X_SR$ – Equivalente a C_{0X}.

K_D – Ganho da componente diferencial de um compensador PID.

K_I – Ganho da componente integrativa de um compensador PID.

$K_{LED\ Driver}$ – Sensibilidade do LED Driver dada em mA/V.

K_P – Ganho da componente proporcional de um compensador PID.

K_{VCO} – Ganho em frequência do oscilador em função da tensão aplicada.

L – Indutância de um par de fios de transmissão.

L_{av} – Luminância relativa medida pela câmara no interior da montra.

L_x – Contribuição da luz proveniente do exterior da montra para a luminância relativa.

$LVAL_IN$ – Sinal de período de linha.

$Line_Cnt$ – Contagem das linhas de uma *frame*.

l – Comprimento do par de fios de transmissão.

$Max(\varphi_{Error})_{ADPLL}$ – Erro de fase máximo entre sinais de relógio obtido em simulação utilizando uma ADPLL.

$Max(\varphi_{Error})_{DisposcopeUSB3}$ – Erro de fase máximo entre *frames* na plataforma DisposcopeUSB3.

$Max(\varphi_{Error})_{NanoUSB2}$ – Erro de fase máximo entre *frames* na plataforma NanoUSB2.

$Max(\varphi_{Error})_{Simulink}$ – Erro de fase máximo entre *frames* obtido em simulação para o *Frequency Comparator*.

$Min(\varphi_{Error})_{ADPLL}$ – Erro de fase mínimo entre sinais de relógio obtido em simulação utilizando uma ADPLL.

$Min(\varphi_{Error})_{DisposcopeUSB3}$ – Erro de fase mínimo entre *frames* na plataforma DisposcopeUSB3.

$Min(\varphi_{Error})_{NanoUSB2}$ – Erro de fase mínimo entre *frames* na plataforma NanoUSB2.

$Min(\varphi_{Error})_{Simulink}$ – Erro de fase mínimo entre *frames* obtido em simulação para o *Frequency Comparator*.

$\max(R, G, B)$ – Valor máximo das cores da imagem recebida na câmara.

$\min(R, G, B)$ – Valor mínimo das cores da imagem recebida na câmara.

N – Número de estágios no oscilador em anel.

P – Potência elétrica típica consumida pelo sensor.

P_{obj} – Nível de intensidade de iluminação alvo.

$Pixel_Data$ – Valor do píxel lido.

$Pixel_Eval_Ok$ – Define a região de interesse permitindo a avaliação dos pixéis numa *frame* quando está ativo.

R – Parâmetro multiplicador dependente da responsividade espectral e do tamanho dos pixéis do sensor de imagem.

R_L – Resistência equivalente do estágio de atraso.

R_{cable} – Resistência do cabo de cobre utilizado para alimentar eletricamente o sensor de imagem.

S – Secção reta de um condutor circular com diâmetro d.

$Slave_LVAL_Ref$ – Valor do período da linha numa *frame* proveniente da câmara *Master* em número de ciclos do sinal de relógio.

Sup_Thold – Nível de referência superior utilizado no histograma de imagem.

$Sup_Thold_Sum_Ref$ – Número de pixéis acima do nível de referência superior.

T_{Frame} – Período de uma *frame*.

T_{Frame} – Período de uma *frame*.

T_{Line} – Período de uma linha contida numa *frame*.

$T_{Sensor\ A}$ – Temperatura em graus Celcius aplicada ao sensor A.

$T_{Sensor\ B}$ – Temperatura em graus Celcius aplicada ao sensor B.

$T_{System\ Clock}$ – Valor do período do sinal de relógio de sistema.

$T_{Target\ Freq}$ – Valor do período correspondente a um sinal com a frequência pretendida para o sensor de imagem.

T – Período da sequência do sinal PWM.

T – Temperatura em graus Celcius.

$Target_{Clk}$ – Número de transições do sinal de relógio para obter uma dada frequência de operação no sensor de imagem.

$Target_Clk_Cnt$ – Contagem alvo em número de ciclos de relógio pela qual o módulo de controlo *Frequency Comparator* se deverá ajustar para alcançar um dado *frame rate*.

$Tolerance$ – Tolerância no erro de sincronismo entre *frames* definida internamente na FPGA pelo programador e dada em número de ciclos do sinal de relógio.

Tp – Período "*high*" da sequência do sinal PWM.

t – Tempo de exposição do sensor de imagem à luz.

t_d – Atraso introduzido por cada estágio.

V_A – Amplitude de uma sinusoide.

V_C – Tensão de alimentação (controlo) aplicada ao sensor de imagem.

$V_{LED\ DAC}$ – Tensão aplicada pela DAC acoplada ao LED Driver.

$V_O(t)$ – Tensão de saída de uma sinusoide em função do tempo.

V_{drop} – Queda de tensão provocada pela linha de alimentação do sensor de imagem.

V_{max} – Tensão aplicada no oscilador para a qual a frequência de oscilação é máxima.

V_{min} – Tensão aplicada no oscilador para a qual a frequência de oscilação é mínima.

ε – Constante dielétrica do material isolante dos fios de transmissão.

μ – Permeabilidade magnética do condutor.

ρ – Resistividade do cobre.

τ_1 – Atraso entre a aplicação da palavra de comando na DAC e a estabilização da tensão elétrica aplicada no sensor.

τ_1 – Atraso introduzido pela LED DAC.

τ_2 – Atraso provocado pela linha de transmissão entre o sensor e a plataforma de controlo.

τ_{LED} – Tempo entre a aplicação da palavra digital na LED DAC e a real aplicação da corrente elétrica nos LEDs

τ_{V2I} – Atraso entre a aplicação da tensão na fonte de corrente controlada e a sua reflexão na intensidade de corrente que percorre os LEDs.

τ_{ctrl} – Atraso introduzido pelo sistema de controlo devido ao tempo de análise da *frame*.

τ_{sen} – Atraso introduzido pelo sensor de imagem.

φ – Fase de uma sinusoide.

φ_0 – Fase inicial de uma sinusoide.

$\varphi_{fsensor}$ – Desvio de fase provocado pela linha de transmissão entre o sensor de imagem e a plataforma de desenvolvimento.

φ_{out} – Fase de saída de um oscilador.

$\varphi_{Error_{DisposcopeUSB3}}$ – Erro de fase instantâneo entre *frames* na plataforma DisposcopeUSB3.

ω – Frequência angular de uma sinusoide em rad/s.

Índice

Resumo i
Abstract iii
Agradecimentos v
Lista de Abreviaturas vii
Lista de Símbolos ix
1. Introdução 1
1.1. Motivação 1
1.2. Objetivos 2
1.3. Organização e conteúdos 3
1.4. Contribuições originais 4
2. Estado da arte 5
2.1 Sensores de imagem CMOS 5
2.1.1 Introdução 5
2.1.2 Implementação típica de um sistema digital de imagem 5
2.1.3 CMOS vs CCD 6
2.1.4 Arquiteturas 8
2.1.5 Configurações de relógio 8
2.2 Sincronização de múltiplas câmaras 9
2.2.1 Sensores 3D 9
2.2.2 Mecanismos de sincronização de múltiplas câmaras 11
2.3. Controlo de iluminação 13
2.3.1 Introdução 13
2.3.2 Métodos de determinação do nível de iluminação 14
2.3.3 Métodos de controlo do nível de iluminação 16
2.4. *Field-programmable gate arrays* 18
2.4.1 Introdução 18
2.4.2 Arquitetura 19
2.4.3 Software de desenvolvimento 20

3. Sincronização de múltiplas câmaras sobre uma plataforma FPGA com interface USB3 21
3.1. Introdução 21
3.2. Análise do problema 22
3.2.1 Sensor de imagem NanEye 22
3.2.2 Topologia existente 28
3.2.3 Conclusões 36
3.3. Controlo dinâmico de frequência baseado no período de linha e de *frame* de uma imagem 36
3.3.1 Introdução 36
3.3.2 Topologia 37
3.3.3 Procedimento de design 40
3.3.4 Simulação em MATLAB/SIMULINK 47
3.3.5 Implementação na FPGA 49
3.3.6 Integração no *firmware* existente 50
3.3.7 Resultados obtidos 51
3.4. Conclusões 62
4. Algoritmo de controlo de iluminação sobre uma plataforma FPGA 65
4.1. Introdução 65
4.2. Análise do problema 66
4.2.1 Fonte de iluminação LED 66
4.2.2 Condicionantes 68
4.3 Topologia 73
4.3.1 Modelo teórico 74
4.4 Procedimento de design 75
4.4.1 Definição da ROI 75
4.4.2 Algoritmo de controlo da iluminação (*DAC Word Control*) 76
4.4.3 Algoritmo completo do bloco Illumination Control 82
4.5 Implementação na FPGA 83
4.6 Resultados obtidos 83
4.6.1 Região de interesse (ROI) 83

4.6.2 Ajuste do nível de iluminação 85
4.6.3 Fases de desenvolvimento 90
4.6.4 Recursos utilizados 92
4.7. Conclusões 93
5. Conclusão e trabalhos futuros 95
5.1. Conclusões gerais 95
5.2. Trabalhos futuros 97
Referências 98
A.1 Introdução 103
A.2 Topologia 103
A.2.1 PFD 104
A.2.2 Filtro digital 107
A.3 Simulação em MATLAB/SIMULINK 108
A.3.1 Implementação em simulink 108
A.3.2 Resultados Simulink 108
A.4 Implementação na FPGA 111
A.5 Conclusões 112
Anexo B - Blocos e Parâmetros utilizados na simulação em MATLAB/SIMULINK do módulo de controlo ADPLL 115
Anexo C - Blocos e Parâmetros utilizados na simulação em MATLAB/SIMULINK do módulo de controlo *Frequency Comparator* 121
Anexo D - Implementação em *hardware* dos sub-blocos do *Frequency Comparator* 127
Anexo E - Implementação em *hardware* dos Sub-blocos do *Illumination Control* 135
Anexo F – Teste de controlo de iluminação 147

1. Introdução

Neste capítulo são apresentadas as motivações por detrás da realização de um sistema de controlo para sincronização de múltiplas câmaras sobre uma plataforma FPGA utilizando uma interface USB3. Da mesma forma, são também explicadas as motivações por detrás da criação de um módulo de controlo a operar em paralelo ao anterior, mas aplicado à modulação de uma fonte de iluminação acoplada a uma câmara digital para adaptação às condições de luz. São definidos os objetivos bem como é apresentada a organização de conteúdos deste documento. Em acréscimo são enumeradas as contribuições originais para o trabalho.

1.1. Motivação

Aplicações com múltiplas câmaras estão em crescimento no campo da imagem digital quer em imagiologia médica, mas também cada vez mais em aplicações de eletrónica de consumo. Predominantemente, aplicações com duas câmaras são utilizadas para visão estéreo 3D, contudo aplicações com maior número de câmaras são um mercado emergente, sendo utilizadas em interfaces homem-máquina por controlo de gestos, visão panorâmica de 360º ou aplicações de vigilância e segurança [1], [2]. Contudo, em qualquer uma destas aplicações é necessário fazer o controlo de todas as câmaras de forma a garantir a captura de imagens sincronizadas.

Em particular, o sensor de imagem, NanEye, desenvolvido pela AWAIBA[1], é adequado para aplicações médicas devido ao seu pequeno tamanho (1 mm x 1 mm) e alto *frame rate* (frame rate nominal: 44 fps). Nomeadamente em endoscopia, onde idealmente se exigiria uma tecnologia minimalmente intrusiva garantindo ao mesmo tempo um instrumento seguro e uma imagem clara de alta resolução e qualidade, sem que isso implique dor ou trauma para o paciente [3]–[5]. Em aplicações médicas, nomeadamente em cirurgias ou endoscopia, as câmaras estão inseridas em ambientes com condições de iluminação muito variáveis, muitas vezes com pouca ou nenhuma luminosidade [3]. É de conceber então que um instrumento médico que utilize o módulo de câmara NanEye para este fim, necessite uma fonte de luz embebida com controlo ajustável de intensidade com base na incidência luminosa observada pelo sensor de imagem.

[1] O *core business* da AWAIBA é o desenvolvimento e fornecimento de sensores de imagem para aplicações especializadas, com foco na endoscopia médica, inspeção industrial, tratamento de imagens de alta velocidade, sensores de linha de alta velocidade, câmaras de bordo em automóveis e de imagem e análise científica. A AWAIBA oferece também sensores de imagem CMOS personalizados para estas aplicações, bem como produtos *standard* otimizados para áreas de aplicação bem definidas [6].

Estas questões motivaram o desenvolvimento de um sistema de controlo dinâmico da frequência de operação do sensor imagem NanEye, através da modulação da tensão de alimentação, de forma a garantir um *frame rate* constante, mesmo na ocorrência de variações de temperatura. Da mesma forma, suscitou-se o interesse na criação de um sistema a operar em paralelo com o anterior, que permitisse expor a câmara a diferentes níveis de iluminação através do controlo dinâmico de uma fonte de luz acoplada ao sensor de imagem sem provocar a saturação da imagem, mesmo para condições de iluminação adversas.

1.2. Objetivos

O objetivo do projeto inicial foi o de sincronizar, pelo menos, 4 câmaras individuais através do ajuste dinâmico do *frame rate* de operação de cada câmara, tendo por base a família de sensores de imagem NanEye, e uma plataforma FPGA com interface USB3. Complementarmente a este, pretendeu-se também criar um algoritmo de controlo dinâmico de uma fonte de iluminação LED sobre os drivers LED embebidos na placa de avaliação FPGA fornecida. Desta forma, durante a realização deste projeto, respeitou-se um plano de trabalhos que garantisse os objetivos subsequentes:

1. Investigação sobre o princípio do ajuste dinâmico do *frame rate* dos sensores de imagem.
2. Estudo e compreensão do código fonte VHDL existente para a captura de imagem, pré-processamento e envio através da interface USB, sendo realizada a identificação dos sinais de controlo relevantes para o controlo dinâmico das câmaras.
3. Desenvolvimento de um projeto de comando e controlo dinâmico da frequência de operação de múltiplas câmaras, independentemente do comprimento dos cabos que conectam os sensores ao módulo de controlo, garantindo simultaneamente a fiabilidade do controlo em condições ambientais variáveis.
4. Simulação do sistema conceptualizado no programa MATLAB/SIMULINK.
5. Desenvolvimento e integração do sistema de controlo na plataforma de interface e processamento e execução de testes sob várias condições de funcionamento.
6. Estudo sobre métodos de controlo de exposição à luz de sensores de imagem e controlo dinâmico de intensidade de uma fonte LED.
7. Análise das condicionantes à implementação do sistema de controlo de iluminação.

8. Desenvolvimento de um algoritmo de comando e controlo de uma fonte de iluminação LED, com ajuste dinâmico baseado na intensidade luminosa incidente no sensor de imagem.
9. Implementação, em VHDL, do algoritmo na plataforma FPGA e teste do seu desempenho em laboratório.

1.3. Organização e conteúdos

Esta dissertação está organizada em 5 capítulos (Introdução, Estado de arte, Sincronização de múltiplas câmaras sobre uma plataforma FPGA com interface USB3, Algoritmo de controlo de iluminação em VHDL, Conclusão e Trabalhos futuros), Referências e Anexos.

No Capítulo 1, Introdução, referem-se as motivações para o desenvolvimento e implementação do módulo de controlo de frequência e sincronização de múltiplas câmaras, bem como o sistema de controlo de iluminação tendo por base as imagens captadas por uma câmara. São também definidos os objetivos a alcançar, apresentada a organização do documento, e enumeradas as contribuições originais alcançadas no âmbito da realização desta tese.

No Capítulo 2, Estado da arte, é explicado o princípio de funcionamento das câmaras digitais modernas sendo referidas as características típicas dos sensores de imagem fabricados em tecnologia CMOS. Este tópico é relacionado com o conceito de visão computacional 3D e são referidas as principais técnicas utilizadas na sua geração. A problemática da sincronização de múltiplas câmaras é explicada e são apresentadas algumas soluções implementadas, para a resolução desta questão. Da mesma forma, é explicado o conceito de modulação da iluminação para melhoramento de imagem, sendo referidas técnicas utilizadas em trabalhos relacionados. Por fim, é descrito sucintamente o princípio de funcionamento das FPGAs e mencionadas as suas principais características.

No Capítulo 3, Sincronização de múltiplas câmaras sobre uma plataforma FPGA com interface USB3, é explicado todo o procedimento de desenvolvimento, implementação e teste do sistema de controlo de frequência e sincronização das câmaras digitais NanEye. Mais especificamente, é feita uma análise ao problema da sincronização, tendo em conta as câmaras e as duas plataformas FPGA utilizadas. São referidas as condicionantes e feita uma análise teórica ao sistema proposto como solução. São também explicadas as opções tomadas e a sua tradução numa implementação em *firmware*. No final do capítulo é descrito o procedimento de testes, são apresentados e discutidos os resultados obtidos (em laboratório e em simulação), e são apresentadas as principais conclusões.

No Capítulo 4, Algoritmo de controlo de iluminação em VHDL, é apresentada e discutida a implementação do módulo de controlo de uma fonte de iluminação acoplada a uma câmara digital, tendo por base a análise das imagens captadas para extração do nível de intensidade da iluminação. Em particular, é feita uma análise ao problema da deteção de níveis de iluminação a partir de um *stream* de vídeo, tendo em conta regiões de interesse na imagem, e consequente modulação de uma fonte de luz LED. São também explicadas as opções tomadas ao longo do desenvolvimento do módulo de controlo e analisados os resultados obtidos em laboratório. Por fim, são apresentadas as principais conclusões.

No Capítulo 5, Conclusão e trabalhos futuros, são realçadas as principais conclusões e sugeridos trabalhos futuros que possam dar continuidade e/ou acrescentar valor ao trabalho realizado.

Nos Anexos são apresentados os esquemas resultantes da implementação em firmware dos trabalhos realizados, bem como os esquemas utilizados nas simulações em MATLAB\SIMULINK. É também apresentado um método alternativo para controlo de frequência de múltiplas câmaras, tendo por base uma ADPLL.

1.4. Contribuições originais

As contribuições originais desta tese de mestrado são, na opinião do autor, as seguintes:

1. Desenvolvimento de um sistema de controlo dinâmico do *frame rate* de operação de um sensor de imagem digital auto-temporizado, tendo por base a medição do período de uma *frame* e o período de cada uma das linhas contidas numa *frame*, através de um sinal de relógio com frequência conhecida. Implementação e prova de conceito numa plataforma FPGA.
2. Desenvolvimento de um algoritmo de controlo da iluminação acoplada a uma câmara digital, aplicado a uma plataforma FPGA e com ênfase na minimização da utilização de recursos sem perda de qualidade e fiabilidade na experiência de visualização do vídeo captado.

2. Estado da arte

Neste capítulo são discutidos os conceitos teóricos investigados, no âmbito dos projetos de sincronização de múltiplas câmaras e controlo dinâmico de iluminação com base na análise de imagens. São apresentados trabalhos relacionados em ambas as áreas, e referidas as soluções implementadas. É também feita uma abordagem ao princípio de funcionamento das câmaras digitais bem como das plataformas FPGA.

2.1 Sensores de imagem CMOS

2.1.1 Introdução

Atualmente existem principalmente dois tipos de sensores de imagem de estado sólido: *Charge Coupled Device* (CCD) e *Complementary Metal Oxide Semiconductor* (CMOS). Até há poucos anos a tecnologia baseada em CCD dominava o mercado de sensores de imagem. Contudo, avanços recentes no desenho de sensores de imagem em tecnologia CMOS têm levado à sua adoção em vários produtos de consumo como câmaras para PC, telemóveis, e câmaras digitais. Ao se explorar a habilidade para integrar a captação de luz, com processamento analógico e digital ao nível dos pixéis, é possível criar novos sensores CMOS para aplicações de visão em robótica, testes biológicos, instrumentos médicos, entre outras aplicações, tornando-os numa alternativa cada vez mais vantajosa em detrimento dos CCDs [1].

2.1.2 Implementação típica de um sistema digital de imagem

Um sensor de imagem é normalmente incorporado num sistema digital de captura de vídeo tal como ilustrado na Fig. 2.1. O processo de captura de imagem inicia-se quando a luz proveniente da cena é focada na câmara através de uma lente. A área de um sensor de imagem consiste num *array* de pixéis, cada um contendo um fotodetetor, que converte a luz incidente em corrente, e em alguns circuitos de leitura, necessários para converter a corrente em carga elétrica (ou tensão) e para realizar a sua leitura para fora do *array* (processo designado por *readout*). Um *array* de microlentes é tipicamente colocado em cima do *array* de pixéis para aumentar a quantidade de luz incidente em cada fotodetetor. Para extrair a informação da cor é introduzido um filtro de cor (*Color Filter Array* – CFA) entre o *array* de microlentes e o *array* de pixéis do sensor de imagem. Os filtros de cor filtram a luz por gama de comprimento de onda, de forma que as intensidades de luz filtradas incluam informação acerca da cor da luz. Por exemplo, o padrão Bayer (mostrado na Fig. 2.2) fornece informação acerca da intensidade de luz nas regiões do comprimento de onda do vermelho,

verde, e azul (RGB). Consequentemente, cada pixel produz um sinal correspondendo a apenas uma das três cores primárias. Estes sinais elétricos analógicos são lidos do sensor de imagem e digitalizados por um conversor analógico digital (*Analog to Digital Converter* – ADC). A imagem originalmente capturada pelo sensor de imagem é então convertida para uma imagem a cores através de uma operação de interpolação espacial conhecida por *demosaicking*. São também realizadas correções de cor e aplicados algoritmos de compressão de imagem para posterior envio para outra plataforma [1].

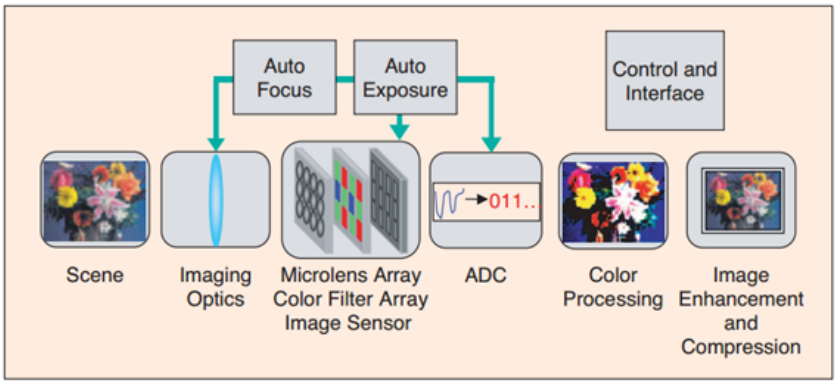


Figura 2.1 – Arquitetura típica de um sistema digital de captura de imagens [1].

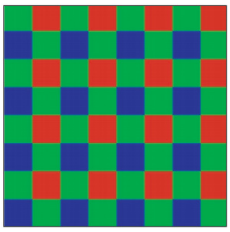
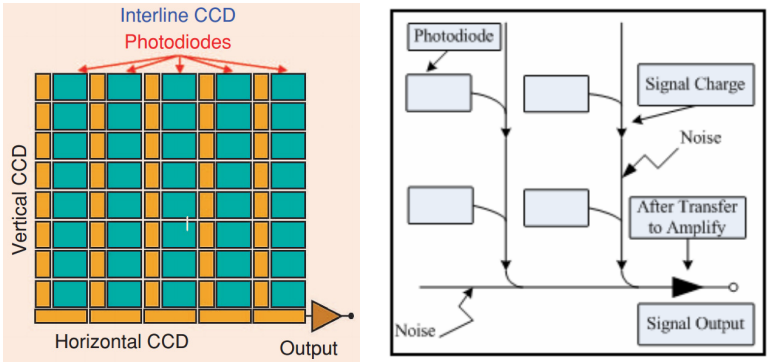


Figura 2.2 – Filtro de cor com padrão Bayer [1].

2.1.3 CMOS vs CCD

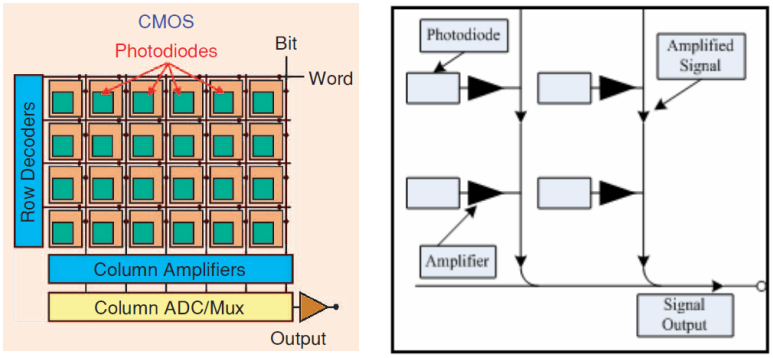
Os principais fatores distintivos entre os sensores de imagem CMOS e CCD estão relacionados com as suas arquiteturas de *readout*. Num CCD (ver Fig. 2.3 a)), a carga é transferida para fora do *array* através de CCDs verticais e horizontais, convertida em tensão através de um único amplificador e depois lida em série para a saída [1]. A particularidade desta configuração é que neste processo a luz captada por cada fotodíodo é amplificada por um único estágio (ver Fig. 2.3 b)).



a) b)

Figura 2.3 – Arquitetura de um sensor de imagem CCD [1], [7].

Num sensor de imagem CMOS, os sinais de tensão são tipicamente lidos linha a linha (*rolling shutter*) utilizando circuitos de seleção de linha e coluna (ver Fig. 2.4 a)) [1]. Contudo, ao contrário dos CCDs na arquitetura *Active Pixel Sensor* (APS) os fotodíodos/fotogates são acoplados a amplificadores individuais (ver Fig. 2.4 b)) [7].



a) b)

Figura 2.4 – Arquitetura de um sensor de imagem CMOS (APS) [1], [7].

Esta estrutura não só permite aos sensores CMOS uma leitura mais rápida e simples das cargas elétricas captadas pelo *array* de fotodíodos, mas também menor consumo energético comparativamente aos CCDs em parte porque estes últimos normalmente necessitam de mais eletrónica à volta do sensor. Além disto, ao contrário dos CCDs, cujo processo de fabrico envolve procedimentos complicados que os tornam caros, os sensores CMOS podem ser fabricados reutilizando os processos de relativamente baixo custo já existentes para produzir microprocessadores e outros chips. Por outro lado, os sensores CMOS são mais sensíveis ao ruído e poderão apresentar distorções causadas pelo efeito do *rolling shutter* [1].

2.1.4 Arquiteturas

Existem diferentes variantes na arquitetura de readout dos sensores de imagem CMOS. Na literatura são referidos três tipos: *Passive Pixel Sensor* (PPS), *Active Pixel Sensor* (APS), e *Diagram Pixel Sensor* (DPS). Aqui será apenas descrita a arquitetura APS, visto ser esta a mais popular e normalmente utilizada [1].

A arquitetura básica do APS é ilustrada nas Fig. 2.4 a) e b). A componente principal desta é o *array* APS, o qual é conectado ao descodificador de linha com o intuito de descodificar os sinais dos pixéis linha a linha. Os sinais descodificados são lidos para a coluna correspondente em primeiro lugar, que então converte os sinais analógicos para sinais digitais que mais tarde são lidos para fora do sensor através do *output*. Os sensores APS podem ser de 3 tipos: APS com fotodíodo, APS com fotogate, e APS com fotodíodo fixo (*pinned photodiode APS*). Destes os APS com fotodíodo e fotogate são os mais utilizados [8].

2.1.5 Configurações de relógio

Nos sensores de imagem é necessário existir um sinal de relógio para comandar a operação da câmara definindo o *pixel clock*. O *pixel clock* é um sinal de sincronismo utilizado para dividir o *stream* de dados de vídeo em pixéis [9]. Tipicamente são utilizados dois métodos para fornecer ao sensor este sinal de relógio.

2.1.5.1 Relógio Externo

A configuração mais comum na literatura implica fornecer um sinal de relógio externo ao sensor e um sinal de *trigger* para comandar a sua operação. Estes sinais são fornecidos por um módulo de comando externo (também designado por *frame grabber*) através de uma interface de comunicação (tal como I^2C [2]). Nesta configuração garante-se que todas as câmaras recebem um sinal de relógio com a mesma frequência e a mesma fase, assumindo que os cabos têm exatamente as mesmas dimensões. O sinal de relógio pode ser gerado através de uma *Phase Locked Loop* (PLL) usando como referência um oscilador de cristal ou outro sinal externo desde que este seja periódico [9]. Quando o sinal de *trigger* é ativado a câmara executa toda a sequência de captura de imagem, pré-processamento, e envio pela porta de dados [10].

---

[2] I^2C – Barramento com apenas duas linhas: uma linha de dados série (*Serial DAta* – SDA) e uma linha de sinal de relógio série (*Serial CLock* – SCL). Cada dispositivo pode ser endereçado individualmente através de uma relação *master*/*slave* [11].

Exemplos deste tipo de configurações podem ser encontrados em [9], [12], [13]. Em [12] é referido como o chip do sensor de imagem contém uma PLL embebida para gerar todos os sinais internos de relógio a partir de um único relógio externo com uma frequência entre 6 MHz e 27 MHz. Em [9] refere-se que nesta configuração existe uma troca dos sinais de relógio da câmara com o *frame grabber*. Inicialmente este fornece ao sensor um sinal de relógio externo. A câmara, por sua vez, gera um novo sinal de relógio (*pixel clock*) que é enviado, numa linha dedicada ou codificado no *stream* de dados, de volta para o *frame grabber*. Isto permite assegurar que o *stream* de dados está em fase com o sinal de relógio utilizado para fazer a amostragem dos mesmos. Neste caso, é referido que uma diferença de fase poderá ser proveniente dos atrasos internos provocados pelos circuitos do sensor de imagem. Para mais a PLL do sensor e a PLL do *frame grabber* poderão adicionar um *jitter* adicional [9], [12].

2.1.5.2 Relógio Interno

Alternativamente, o sinal de relógio pode ser gerado internamente no sensor, utilizando, por exemplo, um oscilador em anel controlado por tensão (*Voltage Controlled Oscilator* – VCO) [14]. Tanto o sensor referido em [14] como o sensor de imagem NanEye descrito em [15] geram o sinal de relógio internamente através de um VCO. Esta configuração permite eliminar pinos externos do sensor, possibilitando criar sensores de imagem com menores dimensões. A desvantagem é que tipicamente o sinal de relógio gerado internamente apresenta um nível de *jitter* assinalável sendo também a frequência do sinal de relógio, obtido pelo oscilador em anel, sensível à temperatura ambiente [16], [17].

2.2 Sincronização de múltiplas câmaras

2.2.1 Sensores 3D

A extração da distância para um objeto, em cada ponto de uma cena permite formar uma imagem 3D ou realizar a deteção de profundidade. As imagens de profundidade são úteis em diversas aplicações de visão computacional, tais como reconhecimento de objetos e de rosto, jogos de computador 3D, e mapeamento [1]. Existem vários geradores de imagens em 3D que empregam uma variedade de técnicas, tais como visão estereoscópica, a triangulação, interferometria, varrimento vertical con-focal (*Con-focal vertical scanning*), ou *time-of-flight* (TOF) [18].

Um dos objetivos propostos neste projeto é o de sincronizar perfeitamente a operação de múltiplas câmaras digitais, permitindo, entre outras aplicações, realizar um sistema de visão estereoscópica 3D com um tamanho menor do que

3 mm. Isto abrirá a possibilidade ao utilizador para ter uma noção de profundidade, além de altura e largura. Em [19] é descrito um sistema com duas câmaras colocadas sobre uma plataforma e espaçadas por 5,5 cm (ver Fig. 2.5). Isto significa que as câmaras filmam o objeto de ângulos ligeiramente diferentes tal como ilustrado na Fig. 2.6.

Figura 2.5 – Duas câmaras numa plataforma e alinhadas para visão estereoscópica [19].

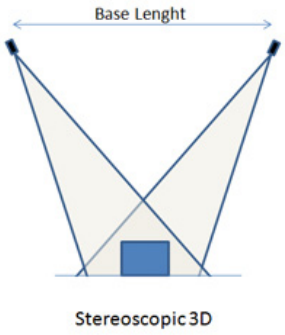
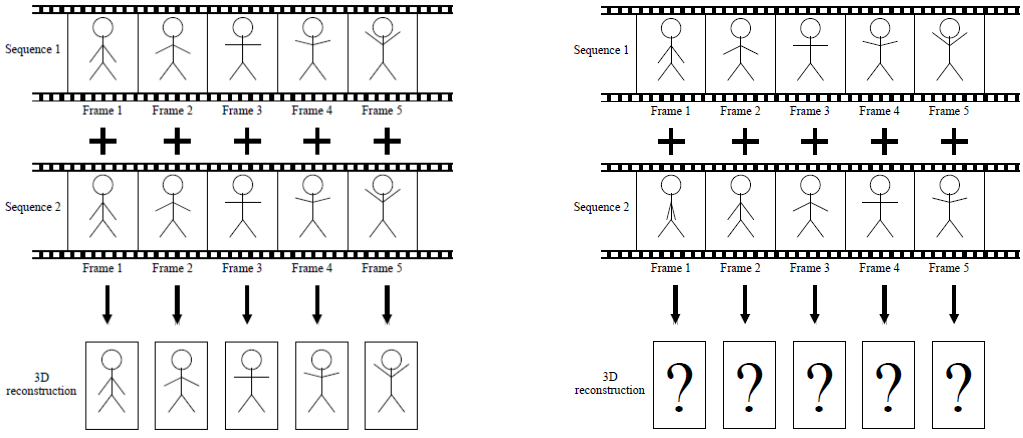


Figura 2.6 – Visão estereoscópica 3D [20].

Para que se possa realizar uma imagem 3D, com visão estereoscópica, é necessário que as câmaras utilizadas no sistema tenham as suas *frames* perfeitamente sincronizadas em fase e em frequência. Sem sincronização não é possível combinar as diferentes *frames* sem utilizar uma memória externa para armazenar as *frames* de vídeo completas [2], [19]. A Fig. 2.7 a) mostra dois *streams* de vídeo a serem fundidos numa imagem estereoscópica 3D. E a Fig. 2.7 b) mostra como *streams* de vídeo assíncronos não podem ser fundidos sem guardar as *frames* completas numa memória externa.



a) Sequências sincronizadas. **b) Sequências não sincronizadas**

Figura 2.7 – Fusão de dois *streams* de vídeo numa imagem estereoscópica 3D [2].

Se as *frames* estiverem sincronizadas, um algoritmo de reconstrução 3D funde as duas imagens captadas numa única imagem, e interpreta as diferenças

entre as duas imagens para determinar a profundidade. A capacidade de perceber a profundidade é o que permite ao utilizador do sistema ver onde os objetos estão em relação ao par de câmaras. Por exemplo para ter uma noção do seu movimento, e para permitir que se possa ver parcialmente em torno de um objeto sem mover o sistema de captura de imagem [21].

2.2.2 Mecanismos de sincronização de múltiplas câmaras

Como foi referido a visão estereoscópica requer a sincronização de fase e frequência entre as *frames* provenientes das câmaras acopladas ao sistema de captura [2], [19]. Em [2] é referido que a sincronização de vídeo pode ser realizada através de *software* ou *hardware*. Nos tópicos seguintes são explicadas as vantagens e desvantagens de cada um destes métodos e referidos alguns exemplos de implementação.

2.2.2.1 Sincronização por *software*

A sincronização por *software* utiliza pistas visuais ou características visíveis, como por exemplo, mudanças de iluminação e pontos em objetos em movimento ou trajetórias de objetos. Isto tem como vantagem permitir realizar a sincronização de vídeo sem *hardware* dedicado para garantir a sincronização das *frames* e sem a necessidade de ter as câmaras fisicamente próximas uma da outra abrindo a possibilidade da utilização de uma rede de câmaras baseadas em tecnologia sem fios [22]. Por outro lado, a sincronização por *software* exige uma quantidade significativa de memória e pós processamento além de que os algoritmos utilizados deverão estar adaptados à aplicação específica da rede de câmaras [2], [19].

Na literatura são referidas duas grandes classes de algoritmos de sincronização por *software*. A primeira destas é designada por Algoritmos Baseados em Características (*Feature-based algorithms*). Estes operam detetando características e seguindo objetos nas sequências de *frames*. Muitos destes métodos são baseados na geometria obtida com sistemas com múltiplos pontos de vista. Exemplos deste tipo de método são descritos em [23], [24] e [25]. Por norma, para medir o erro de sincronização, é calculado um erro algébrico ou geométrico com base na trajetória dos objetos ou na correspondência entre pontos ao longo da sequência capturada. A segunda classe é designada por Algoritmos de Alinhamento Direto (*Direct Alignment algorithms*). Estes algoritmos utilizam o nível de intensidade dos pixéis ou gradientes numa *frame* de vídeo para realizar a sincronização. Isto torna-os adequados para vídeos que contenham mudanças de luz significativas, como por exemplo, fogo-de-artifício, fontes de luz pulsadas, entre outros. Em [26] é proposta uma estratégia de modulação de uma fonte de luz com base numa

codificação Manchester que permite indexar as *frames* captadas. A fonte LED é modulada para que os pulsos de luz captados pela câmara contenham informação temporal referente à comunicação série e, consequentemente, permitam introduzir um *timestamp* em cada *frame* captada.

2.2.2.2 Sincronização por *hardware*

A sincronização por *hardware* utiliza interfaces de controlo e receção de dados para cada câmara sobre fios condutores. O controlo pode ser feito enviando um sinal de relógio externo e um sinal de *trigger*, no caso de a câmara necessitar um *clock* externo. Outras configurações implicam a utilização de memórias para armazenar os dados de vídeo e ajustar os desvios.

As vantagens de utilizar a sincronização por *hardware* são que o erro de sincronização é exatamente conhecido e que a quantidade de memória e pós processamento pode ser minimizada ou completamente eliminada. Por outro lado, estas vantagens são contrabalançadas pela necessidade de *hardware* dedicado e possivelmente dispendioso. Além disto as câmaras deverão ser capazes de receber e reconhecer os sinais de sincronização. Isto implica a utilização de fios entre o módulo de controlo e cada uma das câmaras (ou entre câmaras), o que não é viável em aplicações sem fios ou quando as câmaras estão muito distanciadas [2], [22].

Em [27] é demonstrado um sistema de processamento em tempo real de vídeo estereoscópico de alta definição com base numa FPGA. Nesta aplicação, a FPGA envia um sinal de *trigger* para cada câmara, ajustando o *frame rate* de forma a sincronizar os *streams* de vídeo. Contudo, os pixéis correspondentes a cada câmara não são recebidos necessariamente no mesmo instante, muito devido a pequenas diferenças de fase entre os dois *pixel clocks*. Para sincronizar as duas câmaras com um erro ao nível do píxel, foi necessário armazenar temporariamente os *streams* de vídeo em *First Input First Output buffers* (FIFOs) para compensar os desvios temporais.

O mesmo princípio é utilizado em [19]. No entanto, neste caso são utilizados descodificadores de vídeo externos, colocados entre as câmaras e a plataforma FPGA. Neste caso, assume-se que as câmaras operam ambas à mesma frequência e que todas as ligações elétricas têm exatamente as mesmas dimensões. Cada descodificador de vídeo captura o *stream* de vídeo correspondente e gera o seu próprio sinal de relógio com base no *hsync* da câmara, trazendo ambos os *streams* para o mesmo domínio de relógio. Ambos os *streams* são ligados a FIFOs, na FPGA, que atuam como *buffers* de dados para compensar o desalinhamento dos *streams*.

Outra aplicação para sistemas com múltiplas câmaras é a inspeção industrial em linhas de montagem. O trabalho descrito em [10] demonstra a

necessidade da utilização de um sistema de múltiplas câmaras para deteção de falhas em tubos transparentes de vidro a partir de múltiplos ângulos de visão. Sistemas de inspeção visual em tempo real geralmente consistem no uso de câmaras acionadas com o mesmo pulso comum e controladas por um sistema centralizado. A arquitetura descrita em [10] segue o mesmo princípio utilizando um sinal de *trigger* para acionar o processo de captura, pré processamento, e envio dos dados de vídeo, em cada câmara. A particularidade deste sistema é que todos os dados são enviados para uma câmara principal que depois se encarrega de os enviar para o sistema central localizado num computador.

Uma outra aproximação a este problema foi utilizada em [28]. Esta envolveu enviar um sinal de sincronização a cada câmara para iniciar o processo de captura de imagem. Contudo, a inovação neste sistema foi incluir um *timestamp* em cada *frame* capturada com um código único que traduzia o instante exato em que a *frame* foi capturada. Os dados de vídeo eram armazenados em memórias externas e no pós-processamento, os *timestamps* de cada *frame* eram recuperados, permitindo ao *frame grabber* recolher e alinhar as *frames*, garantindo a sincronização. Outros exemplos de sincronização por *hardware* são referidos em [29] e [30].

No entanto, todas estas implementações têm em comum o facto de utilizarem sinais de sincronização comuns (*triggers*) e/ou memórias para alinhar as *frames* capturadas. Como será demonstrado, estas implementações não serão opções viáveis tendo em conta que o *hardware* alvo apresenta certos condicionalismos. Em primeiro lugar como o sensor de imagem NanEye é auto-temporizado (relógio interno) e funciona autonomamente, não dispõe de um pino de *trigger* nem de *clock* externo. Como consequência apenas dispõe de 4 pinos: 2 para alimentação e 2 para dados (interface *Low Voltage Differential Signaling* – LVDS). Em segundo lugar, apesar das plataformas FPGA disporem de memórias externas (RAM ou Flash) e ser possível a instanciação de RAMs e FIFOs internamente, esta não foi considerada uma opção válida tendo em conta que neste projeto pretendeu-se criar um sistema de sincronização que modulasse ativamente o comportamento das câmaras. Além disso não era claro, até este ponto, se a capacidade memória necessária para armazenar as *frames* seria suficiente.

2.3. Controlo de iluminação

2.3.1 Introdução

Em muitas aplicações, e em particular em aplicações médicas, como é o caso de instrumentos de endoscopia ou braços robóticos para cirurgia, é necessária a integração de uma fonte de luz com o sensor de imagem para

proporcionar uma iluminação adequada à cena captada, sem pôr em causa uma experiência de visualização agradável para o utilizador [3], [4].

A fonte de luz emite normalmente um espectro alargado cobrindo a luz visível, infravermelho, e em alguns casos, ultravioleta. Para evitar que a radiação infravermelha entre no corpo, danificando tecidos ou causando queimaduras, são instalados filtros infravermelhos na fonte de luz. Esta pode ser dos mais variados tipos, desde lâmpadas de xénon, iodetos metálicos ou vapor de mercúrio, contudo no caso particular deste projeto será utilizado um *array* de LEDs de alto brilho e com uma cor branca fria. Em sistemas com controlo de iluminação automático, a fonte de luz é ligada a um processador dedicado com o objetivo de manter o nível de iluminação durante o procedimento. Contudo, alterar a intensidade da fonte de luz através da variação da tensão ou corrente de alimentação pode alterar a temperatura da cor da luz [4].

Alterar a intensidade de iluminação e alterar o tempo de exposição da câmara têm o mesmo efeito na imagem. Porém, alterar o nível de iluminação pode ser visto como um ajuste grosseiro e a alteração da exposição como um ajuste fino. Como consequência, o ajuste do nível de iluminação deverá ser realizado com precaução [31]. Contudo, no caso do projeto descrito nesta tese, o nível de exposição da câmara é considerado como fixo, para evitar perturbações na intensidade de luz captada pela câmara, decorrentes da variação do tempo de exposição e que possam comprometer a estabilidade do controlo. Em [31] é sugerida a utilização de um controlador do tipo Proporcional Integral Diferencial (PID) ou Proporcional Integral (PI). Contudo, este deverá ter uma reação muito suave para controlar o nível de iluminação de uma forma lenta e gradual. Isto poderá ser realizado incluindo uma constante de tempo elevada através de um ganho proporcional baixo (K_P) e um ganho de integração alto (K_I).

2.3.2 Métodos de determinação do nível de iluminação

Apesar de existirem modelos matemáticos para descrever o comportamento da iluminação, não existem, infelizmente, fórmulas para determinar o nível ótimo de iluminação. Esta grandeza depende, entre outros, de um conjunto de fatores em que se incluí: a distância entre a fonte de luz e o objeto (quanto mais próximo o objeto mais iluminação irá incidir sobre este); o campo de visão; tipo de fonte de luz; intensidade luminosa da fonte; contraste entre o objeto e o fundo da cena; nível de refletividade da superfície do objeto; cor da imagem; tipo de sensor de imagem (melhor ou menor qualidade); e parâmetros de funcionamento da câmara (*frame rate*, ganho, velocidade de varrimento – *shutter time*) [32]. Como consequência, não é possível definir um valor determinístico para o nível de iluminação ideal de uma cena, sendo este sempre um valor relativo e dependente dos fatores acima mencionados.

A medição do nível de iluminação a que uma câmara se encontra exposta pode ser realizada utilizando um aparelho de medição externo (tal como um luxímetro) ou diretamente através da imagem obtida com o sensor de imagem. Neste último caso, a informação da iluminação é capturada pelo sensor que a converte em sinais elétricos em posições discretas, representando os pixéis da imagem capturada. Nesta aproximação, cada píxel da imagem é considerado uma fonte de luz cujo nível cinzento (*gray value*) representa a quantidade de iluminância incidente relativa a cada píxel. Por outro lado, deve-se ter sempre em conta que um sensor de imagem CMOS não é um luxímetro ideal [33], [34].

Estimar as condições de iluminação com precisão é fundamental para ajudar o dispositivo de controlo a decidir como compensar o valor emitido pela fonte de luz para obter a melhor exposição luminosa. Na literatura foi possível encontrar algumas propostas para a resolução do problema da estimação do valor de iluminação de uma cena, utilizando as imagens capturadas por uma câmara [33], [35]–[37]. A relação simplificada entre a iluminância, E, e o nível de cinzento, g, no plano da imagem é [33],

$$g = R \cdot E \cdot t \ \ (DN), \tag{2.1}$$

onde t é o tempo de exposição, e R é um parâmetro multiplicador dependente da responsividade espectral do sensor e do tamanho dos pixéis. De (2.1) conclui-se que existe uma relação linear entre a iluminância e o nível de cinzento. Isto significa que é possível utilizar a escala de cinzentos, obtida por um sensor, como uma medida fiável da iluminância incidente [33]. No trabalho descrito em [33] foi provado experimentalmente a existência desta relação linear, testando a variação da curva do nível de cinzentos, g, em função da iluminância, E, para diferentes tempos de exposição, t. Após a correção das distorções introduzidas pelo sistema de captura de imagem e determinação empírica do parâmetro R, foi possível obter uma estimação do nível de iluminação.

Em [36] é proposto um método de estimação que usa a relação entre o valor médio e o valor mediano de uma imagem. O valor médio é apenas a componente média da soma de todos os pixéis numa imagem. O valor mediano é obtido ordenando o valor dos pixéis num *array* e selecionando o valor central. Em condições normais de iluminação, o nível de brilho em todos os pixéis segue uma distribuição uniforme ao longo de todo o espectro de cor e iluminação. Consequentemente, o valor médio varia pouco em relação ao valor mediano. Por outro lado, em condições de alto contraste ou em condições de sobre ou sub iluminação, o valor mediano diferencia-se significativamente do valor médio. Assim é possível obter uma estimação das condições de iluminação da imagem. Este método foi utilizado em conjunto com um mecanismo de auto exposição de forma a ajustar o valor da exposição da câmara com base nas condições de

iluminação. Contudo, revelou-se ser necessário realizar a fusão de múltiplas *frames* captadas a diferentes níveis de exposição para que se obtivesse uma imagem com um nível de exposição adequado [36].

Outra aproximação ao problema envolve a utilização de histogramas de imagem [37]–[39]. Estes fornecem informação acerca da distribuição da intensidade luminosa incidente nos pixéis ao longo da escala dos cinzentos (*gray scale*). Este conceito é abordado em maior detalhe na secção 4.2.2.1. No caso do trabalho proposto em [37], [38], a estimação da iluminação tem como propósito ajudar a definir o tempo ótimo de exposição da imagem. Já em [39] é proposto um algoritmo de estimação da iluminação de uma imagem para o propósito de determinação da distribuição espectral de potência de uma fonte de luz. Contudo, foi utilizado um histograma de cromaticidade, onde foi representada a intensidade de iluminação para cada cor do espectro. Esta informação foi utilizada por uma rede neuronal para determinar o comprimento de onda central da fonte de luz.

2.3.3 Métodos de controlo do nível de iluminação

Através da correta estimação do nível de iluminação incidente na câmara, é possível projetar métodos para realizar um controlo eficaz da intensidade de luz emitida de uma fonte, sem comprometer a experiência de visualização das imagens captadas. Em particular, aplicações com fontes de iluminação em LED são muito comuns [40]–[42]. O controlo deste tipo de fonte pode ser feito através de modulação por largura de impulso (*Pulse Width Modulation* - PWM) ou controlo de corrente [40].

2.3.3.1 Sistemas de controlo de iluminação generalizados

Na literatura são descritas algumas arquiteturas de controlo de iluminação, mas não necessariamente baseadas em sistemas de visão com câmaras [41], [43]. Em relação a sistemas de controlo de iluminação baseados em FPGA, tem-se como exemplo os trabalhos desenvolvidos em [42], [44]. Contudo, como o foco do trabalho incide no campo dos sistemas de controlo baseados em sensores de imagem, estes trabalhos são apenas referidos a título de exemplo.

2.3.3.2 Sistemas de controlo de iluminação com base em câmaras

Atualmente sistemas inteligentes de controlo de iluminação com base na análise de imagens captadas por sistemas de visão computacional são utilizados em aplicações como em edifícios e habitações inteligentes ou em parques de estacionamento [40]. Um dispositivo como uma câmara digital pode ser utilizado

como um substituto para sensores de iluminação ou de movimento que continuam a ser amplamente utilizados neste tipo de aplicações [40].

Em [40] é descrito um sistema de controlo de iluminação automática de um LED, tendo por base uma câmara para obter informação acerca de objetos em movimento e intensidade de iluminação da cena captada. A aplicação destinou-se a ambientes interiores e usa a informação obtida a partir do processamento de vídeo para detetar a posição de uma pessoa numa divisão e o nível de intensidade luminosa a que esta se encontra exposta, atuando no *array* de LEDs para que a posição atual e o caminho correspondente à trajetória esperada estejam sempre bem iluminados. Aqui o cálculo da intensidade luminosa do LED baseada na intensidade luminosa detetada na imagem segue o seguinte procedimento: Suponha-se que a intensidade luminosa da imagem recebida é Br e calculada como,

$$Br = \frac{\max(R,G,B) + \min(R,G,B)}{2} \ (DN), \tag{2.2}$$

com $\max(R,G,B)$ e $\min(R,G,B)$ o valor máximo e mínimo, respetivamente, das cores recebidas na câmara. Este fornece uma medida grosseira do nível de iluminação da imagem, ajudando a definir o valor escrito no LED, D_{LED}. Este pode tomar os valores 0 a 255 e é dado por,

$$D_{LED} = 255 - Br \ (DN). \tag{2.3}$$

O nível de iluminação do LED é imposto através de PWM. A intensidade de luz emitida pelo LED é dependente do *duty cycle* do sinal PWM. Considerando T o período da sequência do sinal, Tp pode ser visto como o período "*high*" do mesmo, com D_{LED} a continuar a ser o nível de intensidade de iluminação do LED,

$$D_{LED} = \frac{Tp}{T} \ (DN). \tag{2.4}$$

Outra aplicação de controlo de iluminação, que tem por base a deteção de intensidade de luz através de uma câmara é descrita em [34]. Neste, foi criado um sistema de controlo da iluminação numa montra de uma loja. Uma câmara digital de baixo custo foi utilizada como um medidor de iluminância utilizado pelo controlador para manter um nível de contraste mínimo entre o interior da montra e o exterior. Para tal, foi considerada a definição de contraste, C_w, de Weber,

$$C_w = \frac{L_{av} - L_x}{L_x}, \tag{2.5}$$

onde L_{av} é a luminância relativa medida pela câmara no interior da montra e L_x é a contribuição da luz proveniente do exterior da montra para a luminância

relativa [34]. Fixando um contraste alvo, C_{obj}, o nível de iluminação alvo para a iluminação, P_{obj}, é dado por,

$$P_{obj} = \frac{C_{obj}}{b} L_x \left(\frac{cd}{m^2}\right), \tag{2.6}$$

com b, um fator linear obtido a partir da relação entre a luminância detetada e a intensidade de luz emitida pelas lâmpadas, e P_{obj} a poder tomar valores de 0 a 1 (onde 1 significa intensidade máxima da lâmpada e 0 intensidade mínima) [34]. Com base neste valor, o sistema atua nos drivers da iluminação LED através do protocolo *Digital Multiplex* (DMX)[3].

Existem ainda outros métodos apoiados em algoritmos baseados na luminância média da imagem, na raíz quadrada da luminância média, e no valor máximo da luminância da imagem. Estes algoritmos têm como vantagem a sua simplicidade contudo não permitem um controlo tão fino da iluminação incidente (por exemplo, a imagem pode ter uma luminância média razoavelmente baixa mas pode mesmo assim haver grandes porções de pixéis saturados). Para uma modulação mais adaptativa da iluminação foram propostas arquiteturas baseadas em histogramas [31], [37], [45], [46].

2.4. *Field-programmable gate arrays*

2.4.1 Introdução

Uma FPGA é um circuito integrado que tem a capacidade de ser reconfigurado utilizando uma forma de especificação do circuito pretendido, que pode ser uma linguagem de descrição de *hardware* como VHDL ou Verilog. O seu princípio de funcionamento assenta sobre a utilização de blocos de lógica configurável (*Configurable Logic Blocks* - CLBs) que se encontram ligados numa ligação em matriz. Cada CLB é composto por *Look-Up Tables* (LUTs), *multiplexers* e também por flip-flops que, em conjunto, permitem implementar operações lógicas programáveis [47], [48]. Isto permite ao *designer* implementar funções combinatórias complexas, funções matemáticas, descodificadores, ou simples operações lógicas como AND e XOR [48]. Esta plataforma permite implementar sistemas de controlo com *hardware* dedicado (como é o caso do projeto descrito nesta tese) de uma forma mais prática, visto que as FPGAs evitam o alto custo inicial, os ciclos de desenvolvimento longos, e a inerente inflexibilidade de um circuito integrado específico para a aplicação desejada [49].

[3] DMX – É um *standard* para redes de comunicação digital que é normalmente utilizado para controlar sistemas de iluminação e efeitos.

2.4.2 Arquitetura

A arquitetura das FPGAs consiste em cinco elementos fundamentais programáveis:

- Os CLBs, já referidos, realizam uma vasta gama de funções lógicas bem como armazenamento de dados. Os LUTs implementam a lógica, os *multiplexers* permitem rotear a lógica dentro e para fora do CLB, e os flip-flops atuam como elementos de memória sincronizados com um sinal de relógio;
- Os *Input/Output Blocks* (IOBs) controlam o fluxo de dados entre os pins de entrada/saída suportando comunicação bidirecional bem como operação em *tristate*;
- Os elementos de memória são RAMs (ou memórias flash em FPGAs mais recentes) configuráveis para se obter diferentes capacidades de armazenamento de dados [47];
- Os *Multiplier Blocks* são multiplicadores embebidos e otimizados para operações de multiplicação sem vírgula flutuante. A utilização destes multiplicadores permite libertar recursos para outras operações. As novas famílias de FPGAs já contêm multiplicadores dedicados para operações com vírgula flutuante (embora este não seja o caso nas FPGAs utilizadas no decorrer desta tese) [47];
- Os *Digital Clock Manager Blocks* (DCM) permitem manipular os sinais de relógio de uma forma totalmente digital possibilitando ao utilizador implementar funções de atraso, distribuição, multiplicação, divisão e desfasamento [49], [50].

Estes elementos encontram-se organizados como indicado na Fig. 2.8 - no caso da Spartan-3 3S500E. Como se pode verificar, um anel de IOBs circunda a malha de CLBs. No centro, existem duas colunas de blocos RAM. Cada coluna é constituída por elementos de memória sendo cada um destes associado a um multiplicador dedicado. Os DCMs estão posicionados no centro, com dois no topo e outros dois em baixo.

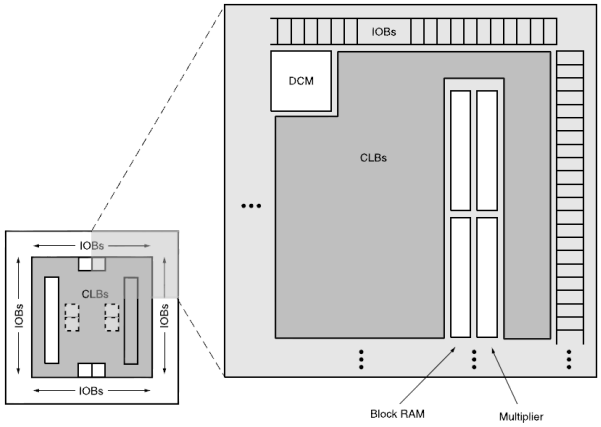


Figura 2.8 – Arquitetura interna da Spartan-3 3S500E [49].

2.4.3 Software de desenvolvimento

O *software* de design fornece ao utilizador uma interface gráfica para todos os estágios de desenvolvimento do *firmware* para a FPGA. Os fabricantes fornecem *software* de desenvolvimento que normalmente realiza quatro funções principais: desenvolvimento (em linguagem HDL ou através de esquema elétrico); simulação; síntese; e programação através de ligações como USB ou JTAG [47], [51]. Um exemplo deste tipo de *software* está ilustrado na Fig. 2.9. Neste caso é apresentada a interface inicial do ambiente de desenvolvimento ISE da Xilinx.

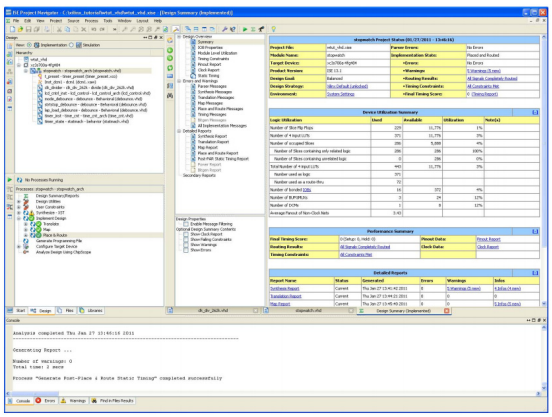


Figura 2.9 – Ambiente de desenvolvimento ISE disponibilizado pela Xilinx [51].

As FPGAs são particularmente indicadas para funções repetitivas e computacionalmente intensivas devido à capacidade que estas possuem para implementar paralelismo. Isto permite reduzir a latência das funções implementadas, algo fundamental em aplicações vídeo que implicam requisitos temporais restritos [52]. Contudo, o desenvolvimento em FPGA tem as suas desvantagens. Entre elas pode-se incluir a falta de flexibilidade na sua programação, a especialização necessária para projetar funções complexas, o tempo necessário para realizar o desenvolvimento das aplicações, entre outras [47], [52].

3. Sincronização de múltiplas câmaras sobre uma plataforma FPGA com interface USB3

No presente capítulo é discutida a implementação do módulo responsável pelo controlo de frequência ou *frame rate* dos sensores de imagem NanEye. Em particular é explicado o procedimento seguido no estudo, simulação e implementação, em linguagem VHDL, de um sistema de controlo dinâmico de frequência baseado no período de linha e de *frame*, denominado *Frequency Comparator*. São apresentados os resultados obtidos para a simulação e implementação dos módulos criados, e justificadas as opções tomadas no desenvolvimento dos mesmos. Para complementar o entendimento sobre o trabalho realizado, é feita também uma análise ao sensor de imagem, à arquitetura de controlo, e às plataformas de desenvolvimento utilizadas na implementação do projeto.

3.1. Introdução

A sincronização de sinais de vídeo provenientes de múltiplas câmaras em simultâneo é uma necessidade para aplicações baseadas em sensores de imagem. Para assegurar que a saída destas aplicações está correta, é essencial a existência de sincronização entre as sequências de vídeo. Caso isto não se verifique poderá haver dessincronização ou perda de *frames*, podendo levar em casos extremos a distorções nas sequências de vídeo visíveis pelo utilizador [2].

O aumento do número de câmaras de vídeo a operar simultaneamente num sistema de captura implica um grande volume de dados de vídeo disponíveis para processamento. Se for considerada a operação em simultâneo de 8 sensores de imagem a transmitir cerca de 40 *frames/s* de vídeo simultaneamente, implica que a aplicação de *software* terá que processar 320 *frames* num único segundo, envolvendo um custo computacional assinalável tal como referido em [53]–[55]. Além de que para realizar um controlo mais rápido e fino da frequência de operação do sensor é necessário atuar num nível mais baixo: o nível do *hardware*. Desta forma minimizam-se os atrasos e consequentemente o erro do sistema de controlo sintetizando circuitos dedicados para processar o erro e atuar na correção, paralelamente.

Uma forma de implementar este controlo ao nível de *hardware* é através de uma plataforma FPGA convertendo funções computacionalmente intensivas em circuitos RTL (*Register Transfer Level*). A sua lógica configurável permite a criação de arquiteturas de controlo dedicadas compostas por circuitos lógicos paralelos, minimizando os tempos de atraso e permitindo, em alguns casos, realizar aceleramento de *software* por várias ordens de magnitude superior aos

mais modernos microprocessadores. Por outro lado, o *design* de circuitos RTL é complexo e demorado, os tempos de compilação são longos e a depuração de um aplicativo FPGA é significativamente mais complexa do que a depuração de *software* [52].

3.2. Análise do problema

3.2.1 Sensor de imagem NanEye

O sensor de imagem NanEye, visível na Fig. 3.1, é uma pequena câmara que consiste num sensor de imagem CMOS com uma área de 1 mm x 1mm e que pode ser adaptada a uma lente ótica miniaturizada. O dispositivo é montado num cabo de fita plana de dimensão até 2 m de comprimento que se conecta à estação base Awaiba (Awaiba *Base Station*). Por sua vez, esta está conectada a um *Personal Computer* (PC) com um cabo USB padrão medindo até 5 m [56]. A estação base é o *hardware* entre a câmara e o PC, composto essencialmente por uma plataforma FPGA, e faz a deserialização do fluxo de dados recebido da câmara. O Awaiba *Viewer* baseado em Windows é uma ferramenta de *software* capaz de capturar imagens da câmara via USB e fazer correções de imagem como *offset*, correção de ganho, reconstrução de cor, etc.

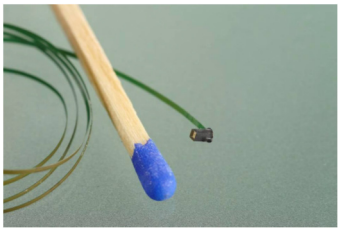


Figura 3.1 – Sensor de Imagem NanEye [57].

3.2.1.1 Arquitetura

A arquitetura do sensor de imagem NanEye pode ser observada na Fig. 3.2. Esta é uma câmara digital com uma leitura de *frames* (denominado por *readout*) totalmente autónoma e auto temporizada, conversão para 10 *bit* e transmissão série dos dados via LVDS (*Low-Voltage Differential Signaling*), tudo integrado num SoC (*System on a chip*) com uma dimensão de 1x1x1,5 mm [15]. O sensor é alimentado por uma tensão contínua entre 1,6 V e 2,4 V nos pinos V_{DD} – V_{SS}. É gerado um sinal de relógio internamente no sensor, cuja frequência é diretamente proporcional à tensão aplicada. Este controla o *readout* da luz capturada na matriz de pixéis e convertida em tensão elétrica. Os valores são quantificados no seu equivalente digital através de uma ADC e enviados para a estação base via LVDS (pinos *out_n* e *out_p*). Não existindo a necessidade de componentes externos próximos do sensor para que este possa operar, nem mesmo condensadores externos [56].

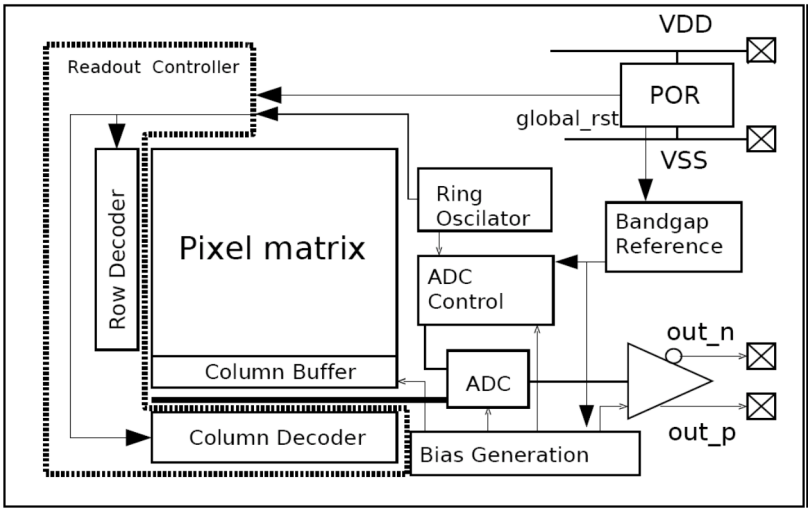


Figura 3.2 – Diagrama de blocos do sensor de imagem CMOS NanEye [57].

3.2.1.2 Modelo teórico

A. <u>Oscilador em anel</u>

Em teoria, este sensor pode ser modelado por um oscilador em anel controlado por tensão. Este tipo de oscilador é composto por um número de atrasos repartidos em estágios (ou andares), com a saída do último estágio a realimentar a entrada do primeiro. Para alcançar a oscilação, o anel deve proporcionar uma mudança de fase de π e ter um ganho em tensão unitário na frequência de oscilação. Cada atraso deverá proporcionar um desvio de fase de π/N, onde N é o número de estágios de atraso, e a saída do último estágio deverá ser o inverso da saída anterior. Isto implica que o oscilador em anel deverá conter um número ímpar de estágios [58]. Um exemplo de um oscilador em anel está ilustrado na Fig. 3.3, com o estágio de atraso na Fig. 3.4.

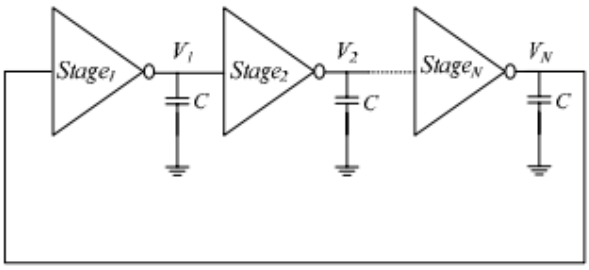


Figura 3.3 – Oscilador em anel [59].

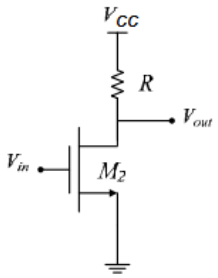


Figura 3.4 – Estágio de Atraso [59].

Assumindo que todos os estágios inversores são idênticos, a frequência de oscilação do anel, f_{out}, pode ser escrita como,

$$f_{out} = \frac{1}{2Nt_d}\ (Hz), \tag{3.1}$$

onde N é o número de estágios no oscilador e t_d é o atraso introduzido por cada um destes. Por sua vez, o atraso em cada estágio pode ser expresso por,

$$t_d = R_L \cdot C_L\ (s), \tag{3.2}$$

com C_L representando a capacitância equivalente conectada à saída do estágio. E R_L a resistência equivalente do estágio de atraso. Combinando (3.1) com (3.2) obtém-se,

$$f_{out} = \frac{1}{2NR_LC_L}\ (Hz). \tag{3.3}$$

Contudo, deste resultado pode tornar-se complicado extrair o valor da frequência de oscilação principalmente devido a não linearidades e elementos parasitas presentes em cada estágio do oscilador [58].

B. VCO – Oscilador controlado por tensão

Sendo o sensor essencialmente um oscilador controlado por tensão (VCO – *Voltage Controlled Oscillator*), é possível modelar este como sendo um sistema cuja saída em frequência, f_{out}, é uma função de uma variável de controlo em tensão, V_C, tal como ilustrado pela Fig. 3.5 [60].

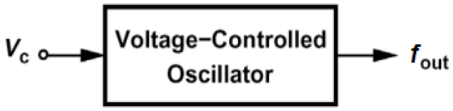


Figura 3.5 – Oscilador controlado por tensão [60].

Considerando o sensor de imagem como um VCO ideal, pode-se obter a sua frequência de saída como uma função linear da tensão de controlo,

$$f_{out} = f_0 + K_{VCO}V_C\ (Hz), \tag{3.4}$$

em que f_0 é uma constante representando a ordenada na origem, e K_{VCO} caracteriza a sensibilidade/ganho do oscilador, expresso em Hz/V, e representado como o declive da reta vista na Fig. 3.6 [60].

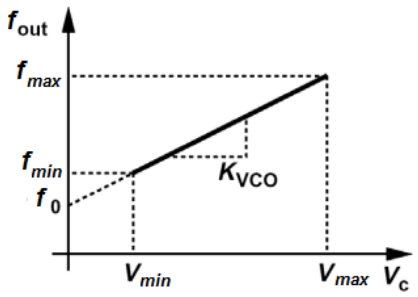


Figura 3.6 – Definição de K_{VCO} [60].

Nesta pode-se observar que o intervalo de operação do oscilador (ou *tuning range*) encontra-se limitado entre f_{min} e f_{max}, à tensão de controlo V_{min} e V_{max}, respetivamente. Na realidade, K_{VCO} é a derivada parcial da frequência de oscilação em ordem à tensão de controlo [61],

$$K_{VCO} = \frac{\partial f_{out}}{\partial V_C} \left(\frac{Hz}{V}\right). \tag{3.5}$$

Alternativamente também pode ser escrito como [62],

$$K_{VCO} \approx \frac{f_{max} - f_{min}}{V_{max} - V_{min}} \left(\frac{Hz}{V}\right). \tag{3.6}$$

Esta análise permitiu perceber a relação entre a frequência de saída de um VCO, constituído por um oscilador em anel, em função da tensão de controlo. Contudo, falta analisar o seu comportamento em função do tempo. Considere-se uma sinusoide com fase φ e amplitude V_A [60],

$$V_O(t) = V_A \cos(\varphi(t)) \ (V), \tag{3.7}$$

em que,

$$\varphi(t) = 2\pi f t = \omega t \ (rad), \tag{3.8}$$

sendo f a frequência da sinusoide em Hz e ω a frequência angular da sinusoide em radianos/s. À medida que a onda se propaga no tempo a fase varia linearmente tal como ilustrado na Fig. 3.7.

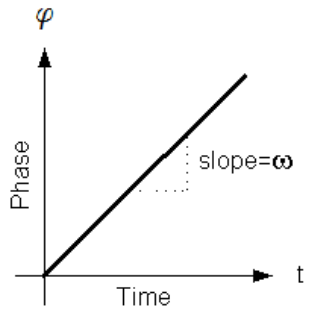


Figura 3.7 – Ilustração da variação de fase de uma sinusoide ao longo do tempo [63].

Quanto mais inclinada for esta reta mais rapidamente a fase varia, consequentemente maior será a frequência da sinusoide, o que pode ser traduzido matematicamente na derivada da fase em função do tempo,

$$\omega = \frac{d\varphi}{dt} \left(\frac{rad}{s}\right); \tag{3.9}$$

$$f = \frac{1}{2\pi} \cdot \frac{d\varphi}{dt} \ (Hz). \tag{3.10}$$

Da mesma forma, a fase é o integral da frequência,

$$\varphi = \int \omega \, dt = \int 2\pi f \, dt \, (rad). \tag{3.11}$$

Como num VCO a frequência do sinal à saída é dada por (3.4), resulta que a sua forma de onda é dada por,

$$V_O(t) = V_A \cos\left(2\pi f_0 + K_{VCO} \int V_C \, dt + \varphi_0\right) \, (V), \tag{3.12}$$

com φ_0 a fase inicial da sinusoide. Contudo ao analisar a saída de um VCO, a corrente e a tensão instantânea da forma de onda são ignoradas, sendo apenas a fase relevante para a análise,

$$\varphi_{out} = \int 2\pi f \, dt = K_{VCO} \int V_C \, dt \; (rad). \tag{3.13}$$

À entrada do VCO temos uma tensão mas à saída temos uma fase. Pela Fig. 3.8, observa-se que o nível da inclinação da rampa de fase é diretamente proporcional ao nível de tensão aplicado.

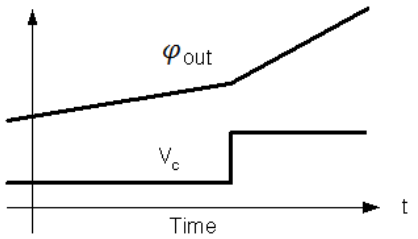


Figura 3.8 – Inclinação da curva da fase em função da tensão aplicada [63].

Daqui conclui-se que um VCO é essencialmente um integrador. Por outro lado, em termos de frequência, pode ser modelado como um filtro passa baixo com ganho K_{VCO}. Sabendo que o operador Laplace para a integração é $1/s$, deduz-se a função de transferência do VCO como sendo,

$$\frac{\varphi_{out}(s)}{V_C(s)} = \frac{K_{VCO}}{s} \; \left(\frac{rad}{V}\right). \tag{3.14}$$

C. K_{VCO} experimental

Como não foram fornecidos dados acerca da sensibilidade do sensor de imagem, foi necessário medir o K_{VCO} experimentalmente. Para tal variou-se a tensão aplicada ao sensor de 1,6 V a 1,8 V, medindo consecutivamente a frequência do sinal de dados proveniente do mesmo. Na Fig. 3.9, pode-se observar o comportamento da frequência do sinal de relógio proveniente do sensor de imagem em função da tensão aplicada, em 2 sensores de imagem NanEye 2C (a vermelho e a verde) e num sensor NanEye 2B (a azul).

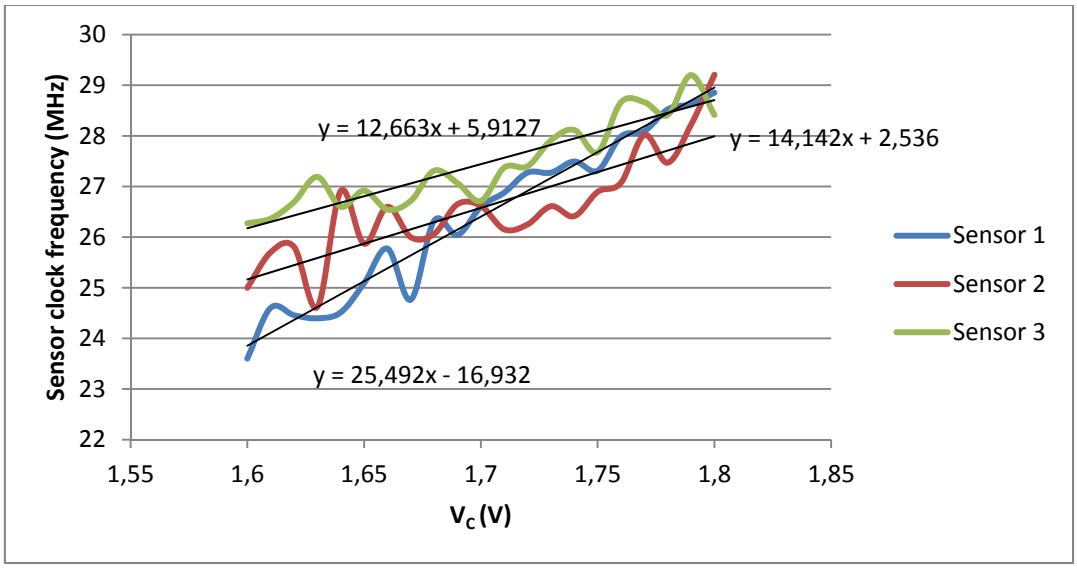


Figura 3.9 – Frequência de oscilação do sinal de relógio do sensor em função da tensão aplicada.[4]

Como expectável, a frequência de oscilação do sensor aumenta proporcionalmente com a tensão de alimentação aplicada com uma dada sensibilidade K_{VCO}. Contudo, observou-se que mesmo para sensores semelhantes, a inclinação da curva (sensibilidade do sensor) varia consideravelmente. Isto deve-se a variações no processo de fabrico dos sensores CMOS a uma escala nanométrica. Estas variações são causadas por imperfeições no processo de fabrico tais como, defeitos na máscara de litografia, irregularidades na lente ou impurezas [64]. Inclusive em [64] foi demonstrado como as variações no processo de fabrico têm um efeito drástico nas métricas de performance tal como a frequência de oscilação central de um VCO. Isto implica, desde logo, uma condicionante a qualquer sistema de controlo que seja criado com o propósito de controlar a frequência de oscilação do sensor. Deste modo, a arquitetura de controlo deverá ser necessariamente independente da sensibilidade do sensor.

Por outro lado, o conhecimento do valor aproximado de K_{VCO} continua a ser importante para a realização do modelo teórico do sensor e de toda a arquitetura de controlo, quer como para a realização de simulações e testes. Com base nas medidas realizadas, determinou-se a sensibilidade de cada sensor através da expressão 3.6. Dos resultados calculou-se o valor médio da sensibilidade como sendo,

$$K_{VCO} = 17{,}432 \left(\frac{MHz}{V}\right). \tag{3.15}$$

[4] Medições realizadas a uma temperatura ambiente de aproximadamente 24 °C.

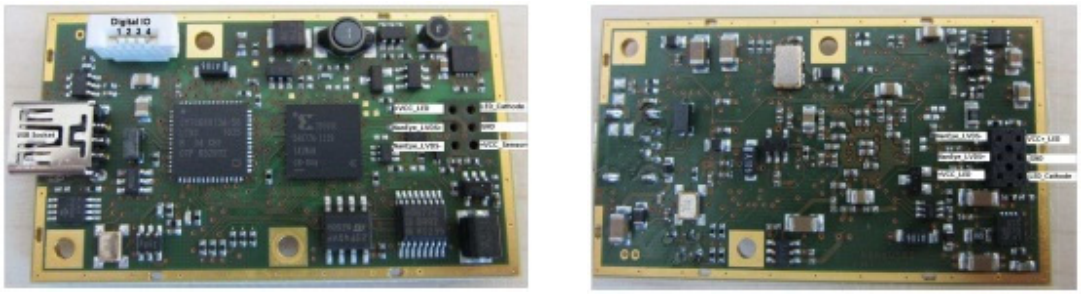
3.2.2 Topologia existente

A correta receção e transmissão dos dados provenientes do sensor de imagem exige todo um sistema digital responsável pela descodificação, armazenamento, e tratamento dos dados para posterior envio para uma plataforma como um PC. Isto implica necessariamente a existência de *hardware* para realizar a ponte entre a câmara digital e a interface do utilizador, alojado no computador. Para preencher este papel é utilizada uma FPGA devido à capacidade que esta possui para realizar processamento paralelo (aspeto fundamental em aplicações de alto débito) mantendo uma certa flexibilidade, bastando carregar as atualizações via um *update* do *firmware* (código VHDL) [52].

3.2.2.1 Plataformas de desenvolvimento e arquitetura do sistema

A. Plataforma NanoUSB2 (Spartan 3E)

Durante o decorrer do trabalho foram utilizadas duas plataformas de desenvolvimento distintas. Numa primeira fase foi utilizada a plataforma denominada NanoUSB2 (ver Fig. 3.10 a) e b)), composta por um chip XILINX Spartan FPGA XC3S500E-4CPG132C e uma interface USB 2.0 [65].



a) Vista de cima. **b) Vista de baixo.**

Figura 3.10 – Plataforma de desenvolvimento NanoUSB2 [65].

Além do conector MiniUSB e do chip FPGA existe na plataforma o conector para o NanEye e para os LEDs, quatro pinos digitais I/O (designados por *Push Buttons*), duas DACs de 12 bit, além de toda a eletrónica envolvente como capacidades de filtragem de ruído [65].

A topologia de *hardware*, já existente aquando do início do projeto descrito neste capítulo, tinha a estrutura ilustrada na Fig. 3.11. O sensor de imagem comunica com a NanoUSB2 via uma interface LVDS, transmitindo um *stream* de dados codificados em código Manchester, o que permite adicionar o sinal de relógio do sensor aos dados. Na FPGA, este fluxo de dados é descodificado e convertido do formato série para o formato paralelo. Os dados de vídeo são armazenados numa DRAM (*Dynamic Random Access Memory*) e

posteriormente enviados para o controlador de USB, FX2, constituído por uma FIFO. Devido ao protocolo de comunicação existente nas interfaces USB, são utilizadas FIFOs para adaptar a taxa binária nos barramentos da FPGA para a taxa binária da porta USB [66]. O protocolo utilizado pelo USB 2.0, no modo *Bulk Transfer* permite que a informação seja transmitida em blocos com um tamanho máximo de 512 *bytes* (ou 1024 *bytes* no USB 3.0) [67], [68]. Assim, se for necessário transmitir com alto débito uma imagem através desta interface, preenche-se a FIFO com os dados e apenas quando esta está completamente preenchida é que estes são transferidos para o chip FX2 da Cypress, que se encarrega de os colocar na porta USB. Deste modo, maximiza-se o débito evitando a transferência de pacotes com tamanho inferior.

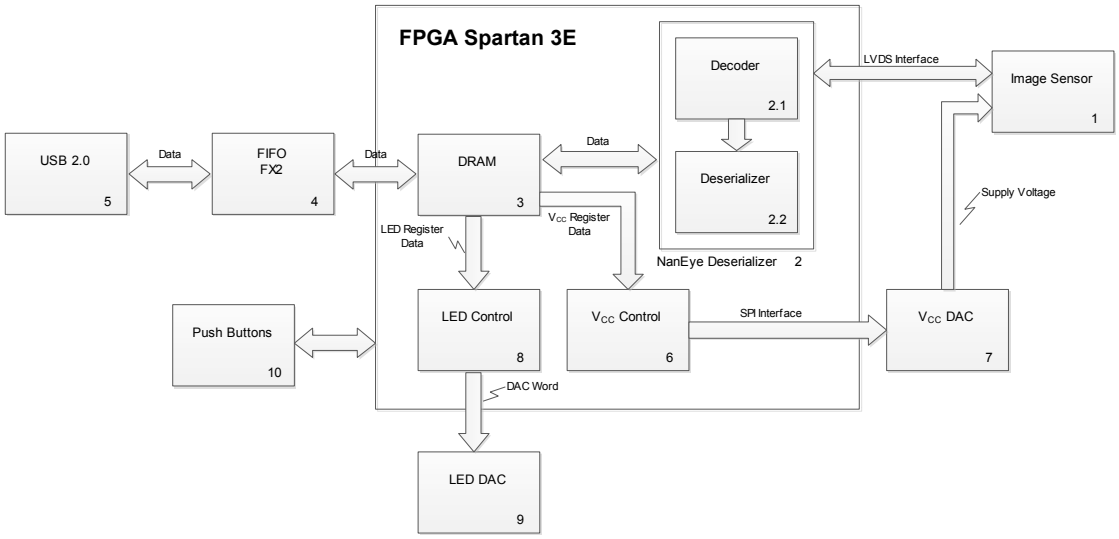
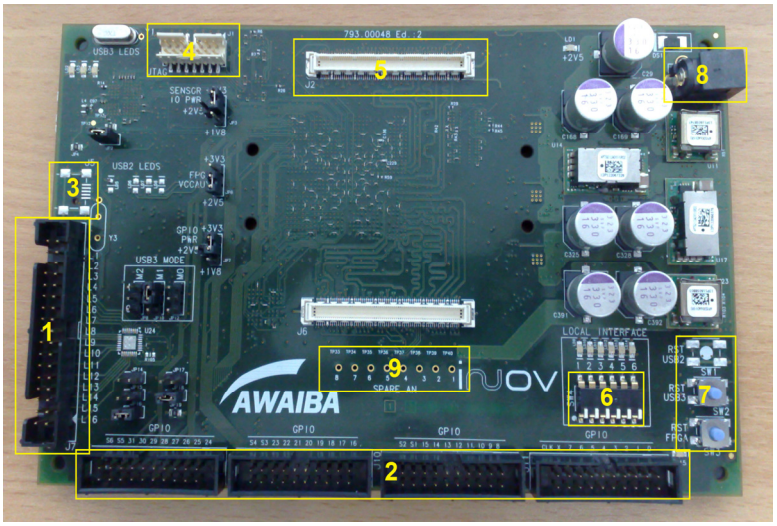


Figura 3.11 – Arquitetura interna na plataforma NanoUSB2.

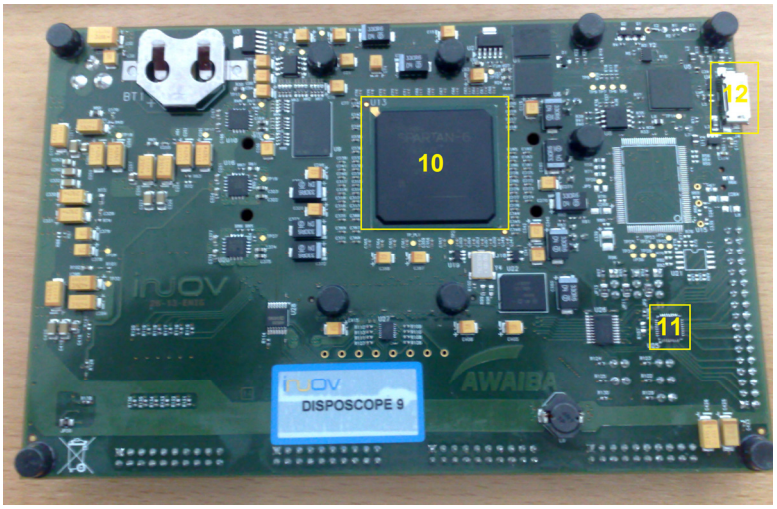
Através da mesma interface, a FPGA recebe simultaneamente os registos de controlo de configuração da câmara, controlo de iluminação (*LED Control*) e controlo da tensão aplicada no sensor (V_{CC} *Control*), estes últimos atuam nas duas DACs de 12 *bit*, LED DAC e V_{CC} DAC, respetivamente. Existe ainda uma interface para o exterior através de 4 pinos I/O designados por *Push Buttons*, utilizados primariamente para *debugging*.

B. Plataforma DisposcopeUSB3 (Spartan 6)

Posteriormente foi utilizada a plataforma denominada DisposcopeUSB3 (ver Fig. 3.12 a) e b)), composta por um chip XILINX Spartan-6 XC6SLX150-FG900 FPGA e uma interface de dados USB 3.0 [69].



a) Vista de cima.



b) Vista de baixo.

Figura 3.12 – Plataforma de desenvolvimento DisposcopeUSB3 [69].

Tabela 3.1 – Legendas.

	Descrição
1	*IO connection*
2	*GPIO connection*
3	*USB2 connection*
4	*JTAG connection*
5	*Hirose FX10*
6	*Switch Control*
7	*Push Buttons*
8	*Power Supply 12 V / 60 W*
9	*IO connection*
10	*Spartan-6 FPGA*
11	*LED Driver*
12	*USB3 connection*

Nesta placa a interface de dados utiliza a tecnologia USB 3.0, contudo o carregamento do *firmware* para a FPGA é feito através de um conector JTAG. O *array* programável é uma Xilinx Spartan FPGA XC6SLX150-2FGG900 com capacidade superior, permitindo a implementação de uma arquitetura mais complexa [50]. Além destas características, a DisposcopeUSB3 possui: um conjunto variado de *General Purpose Input Output ports* (GPIO), utilizados para *debugging* ou interface com dispositivos externos; vários interruptores e botões de pressão; dois conectores Hirose FX10 para interface com os sensores; além de DACs e um LED driver capaz de controlar até 16 fontes de luz externas [69].

Nesta plataforma, a arquitetura interna de *hardware*, já existente aquando do início do projeto, tinha a organização ilustrada na Fig. 3.13. Note-se no entanto, que o exemplo da figura apenas ilustra a conexão de uma câmara

individual à FPGA. Para *N* câmaras os componentes da arquitetura são replicados *N* vezes e ligados aos módulos externos da mesma forma.

O sensor de imagem comunica com a DisposcopeUSB3 via uma interface LVDS e Hirose FX10, transmitindo um *stream* de dados codificados em código Manchester. Na FPGA, este fluxo de dados é descodificado e convertido do formato série para o formato paralelo. Os dados das imagens são enviados para o módulo *Image Out*, e formatados em blocos de 8 *bit* para transmissão para o controlador de USB, FX3 [70]. Uma forma de aumentar a taxa de transmissão através da interface de dados é utilizar uma FIFO para cada câmara. Ao preencher-se totalmente as FIFOs, forma-se um único bloco para transmissão via USB, cujos dados estão divididos em vários *slots*, um para cada FIFO. Assim, considerando como exemplo 4 câmaras a operar simultaneamente, é possível transmitir o *stream* de vídeo destas simultaneamente através de uma única interface USB, dividindo o pacote de dados em 4 partes.

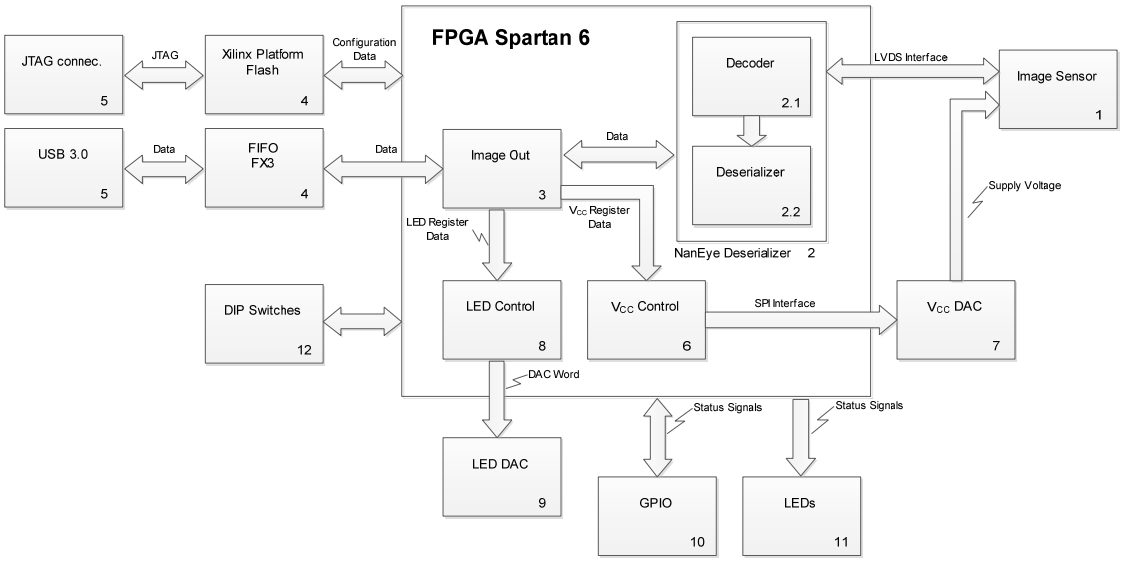


Figura 3.13 – Arquitetura interna na plataforma DisposcopeUSB3 (considerando apenas uma câmara).

No caso da plataforma Disposcope a programação é realizada via interface JTAG, ficando o *firmware* armazenado numa memória *flash*. No entanto, os registos de configuração e dados de vídeo continuam a ser transmitidos via a interface USB 3.0. O controlo de iluminação e controlo da alimentação do sensor continuam a ser realizados através da atuação em DACs de 12 *bit*. As restantes interfaces são os botões de pressão e interruptores (DIP Switches), indicadores luminosos (LEDs), e portos de entrada/saída de uso geral (GPIO).

3.2.2.2 Componentes relevantes

No contexto do projeto do sistema de controlo dinâmico do *frame rate* dos sensores de imagem, os módulos relevantes, independentemente da plataforma utilizada, são essencialmente o bloco de deserialização (*Deserializer)*, o V_{CC} *Control*, V_{CC} DAC, e o sensor de imagem em si. Como ilustrado na Fig. 3.14, do *Deserializer* obtém-se a saída de dados de vídeo, e os sinais de sincronização *vsync* (*vertical synchronization*) e *hsync* (*horizontal synchronization*), e após modificações decorrentes do trabalho realizado, também o sinal de relógio do sensor de imagem (ver Anexo A secção A.4). O V_{CC} Control permite a escrita na DAC segundo o protocolo SPI (*Serial Peripheral Interface*) [71]. Este recebe os registos de controlo e formata os comandos numa palavra binária de 24 *bit* para escrita na DAC. Esta última interpreta a palavra recebida e alimenta o sensor com uma tensão elétrica, V_{CC}. O sensor transmite os dados via uma interface LVDS com codificação Manchester.

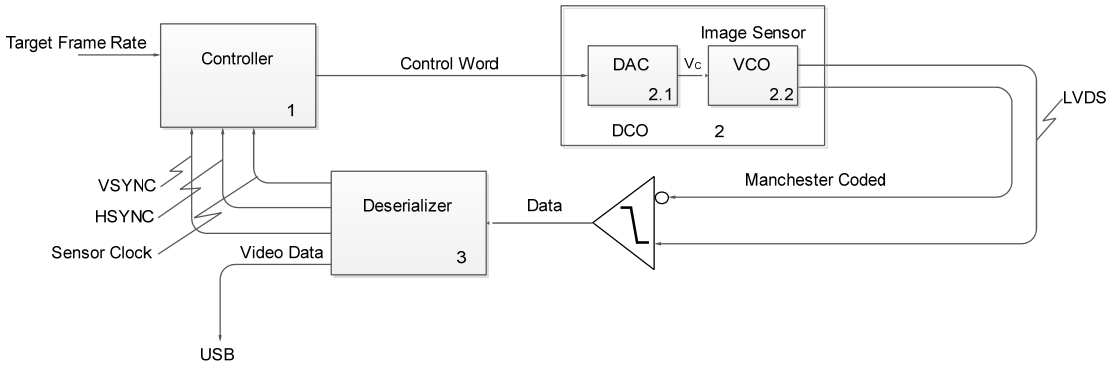


Figura 3.14 – Esquema dos elementos constituintes do sistema fechado de controlo.

3.2.2.3 Condicionantes

A. <u>Atrasos no sistema</u>

Qualquer arquitetura de controlo utilizada deverá ter em conta os atrasos na malha de controlo, particularmente o atraso introduzido pelos componentes do sistema, mas também o atraso introduzido pelo cabo que conecta o sensor à FPGA. Para simplificar a análise apenas são considerados: o atraso entre a aplicação da palavra de comando na DAC e a aplicação da tensão no sensor, τ_1; e o atraso entre a transmissão dos dados pelo sensor e a receção dos mesmos na plataforma, τ_2. Consultando [71] determinou-se,

$$\tau_1 = 4{,}4\ \mu s \tag{3.16}$$

como o atraso máximo. Por outro lado τ_2 é uma função do comprimento do par de fios de transmissão, l, da separação entre os fios condutores, D, do diâmetro dos fios, d, da permeabilidade magnética do condutor, μ, e da constante

dielétrica do material isolante dos condutores, ε. A indutância de um par de fios de transmissão pode ser escrita como [72],

$$L = \frac{\mu}{\pi} \times \operatorname{arcosh}\left(\frac{D}{d}\right)\ (H) \tag{3.17}$$

e a capacitância como,

$$C = \pi\varepsilon / \operatorname{arcosh}\left(\frac{D}{d}\right)\ (F). \tag{3.18}$$

Visto que o cabo de cobre pode ter um comprimento que pode variar entre 0 m e 2 m, considerou-se para efeitos de cálculo o pior caso:

- $l = 2\ m$
- $D = 188\ \mu m$
- $d = 71\ \mu m$
- $\varepsilon = 2{,}4791 \times 10^{-12}\ F/m$
- $\mu = 1{,}2566 \times 10^{-6}\ H/m$

Substituindo as grandezas obtém-se,

$$L = 6{,}5166 \times 10^{-7}\ H; \tag{3.19}$$

$$C = 4{,}7805 \times 10^{-12}\ F. \tag{3.20}$$

Com base nestes resultados calculou-se o atraso da linha como sendo,

$$\tau_2 = \sqrt{L \times C} \times l = 3{,}53\ ns. \tag{3.21}$$

Conclui-se que o atraso provocado pela linha de transmissão não é significativo quando comparado com τ_1. Contudo optou-se por considera-lo na análise da dinâmica do sistema devido à influência dos atrasos na estabilidade de um *loop* de controlo.

B. <u>Quedas de tensão</u>

Outro efeito que o comprimento do cabo de transmissão irá provocar é uma queda de tensão na linha. Levando à diminuição da tensão efetivamente aplicada no sensor. Em primeiro lugar determinou-se a resistência do cabo de cobre com resistividade ρ e secção reta S como sendo,

$$R_{cable} = \rho \frac{l}{S} = 8{,}4866\ \Omega. \tag{3.22}$$

Considerando as características indicadas em [15] determinou-se a queda real de tensão no cabo, V_{drop}, com base na corrente,

$$I = \frac{P}{V_{CC}} = \frac{7{,}6\ mW}{2{,}1\ V} = 3{,}62\ mA, \tag{3.23}$$

como sendo,

$$V_{drop} = R_{cable} I = 0{,}031\ V, \tag{3.24}$$

com I a corrente elétrica que percorre o cabo de alimentação, e P, o consumo de potência do sensor. Isto indica que se fosse aplicada uma tensão nominal de 2,1 V, para um cabo com comprimento de 2 metros, a tensão realmente aplicada ao sensor seria de 2,069 V, o que representa uma queda de 1,46 % em relação ao valor inicial.

C. Dependência da temperatura

Sendo o sensor de imagem, em termos de modelo, um oscilador em anel, é de esperar que a sua frequência de oscilação varie com a temperatura visto que o atraso em cada circuito CMOS depende da temperatura [16]. Na realidade existem algumas aplicações, como em [17], que fazem uso deste fenómeno para realizar medições de temperatura com elevada precisão.

D. Múltiplos sensores

A sincronização de múltiplos sensores em tempo real implica que cada sensor individual deverá estar a operar à mesma frequência. Contudo isto não é suficiente para garantir a sincronização, visto que poderá haver um desvio de fase entre as *frames*, o que implica que qualquer sistema de controlo deverá procurar compensar este desvio através do aumento ou diminuição relativa da alimentação do sensor. Para mais, cada sensor poderá estar ligado à estação base por cabos com comprimentos totalmente distintos e a operar sob condições de temperatura muito diferentes.

3.2.2.4 Modelo teórico

A. Deserializador

O sensor de imagem NanEye contém uma área de captação de luz equivalente a uma matriz de 250x250 pixéis, totalizando 62,5k pixéis por *frame* transmitida. Contudo o protocolo de funcionamento do sensor descrito em [15] especifica que a transmissão de uma *frame* é na realidade constituída por 250 linhas de dados mais 3 linhas reservadas para a configuração do sensor. Em cada uma das linhas são transmitidos 249 PP (*Pixel Period*) mais 3 PP a 0, com cada PP codificado a 12 *bit*. Deste modo é possível modelar o comportamento do *Deserializer* no domínio da frequência como o subsistema da Fig. 3.15.

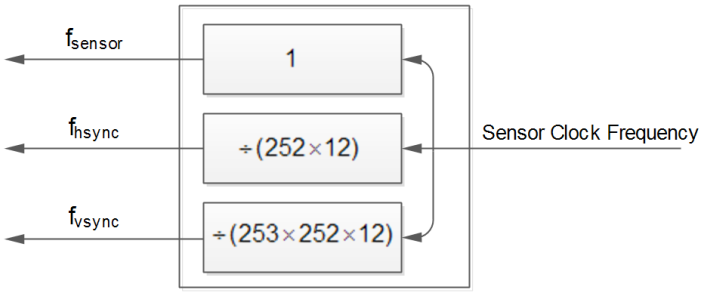


Figura 3.15 – Modelo teórico do *Deserializer*.

Em que f_{sensor} é a própria frequência do sensor, logo não necessita qualquer conversão daí o fator multiplicativo unitário. A saída f_{hsync} corresponde à frequência de linha logo a frequência do sensor é divida pelo equivalente a 252 PP multiplicados por 12 *bit*. Por fim, a saída f_{vsync} é a frequência das *frames* (ou *frame rate*) e é obtida através da divisão da frequência do sensor por 252 PP multiplicados por 12 *bit* e pelo equivalente a 253 linhas.

B. Digital to Analog Converter

Uma das propriedades fundamentais de uma DAC é a sua resolução. A resolução indica o número de *bits* disponíveis no domínio digital, n, para representar um sinal de saída analógico. Outro fator importante é a sua tensão de referência, V_{ref}. Esta estabelece a gama de saída do sinal analógico e a amplitude do passo por cada LSB (*Least Significant Bit*). No caso específico da DAC descrita em [71], $n = 12\ bit$ e $V_{ref} = 2{,}5\ V$, como tal a saída de uma DAC ideal pode ser definida como,

$$V_{out}(t) = \frac{V_{ref} Ctrl_{Word}(t)}{2^n} = \frac{2{,}5 Ctrl_{Word}(t)}{4096} = K_{DAC} Ctrl_{Word}(t)\ (V), \qquad (3.25)$$

com $Ctrl_{Word}$ a palavra digital de controlo que pode tomar o valor 0 a 4095. Note-se que nesta aproximação não é tido em conta o erro de quantização da DAC.

C. Malha de controlo

Sabendo que a DAC de 12 *bit* consegue aplicar uma tensão máxima de 2,5 V, pode-se representar o modelo teórico da malha de controlo tal como ilustrado na Fig. 3.16. Em que $error(\varphi_{Frame})$ representa o erro de fase entre a *frame* do sensor ligado localmente ao *loop* e a *frame* proveniente de uma outra câmara. Note-se no entanto, que esta é, nesta fase, apenas uma representação do modelo teórico da malha de controlo. Como tal, nesta não é indicada qual a técnica utilizada no bloco 1 (*Controller*) para gerar uma palavra de controlo para a DAC. As propostas para resolver o problema do controlo desta malha são apresentadas adiante, na secção 3.3 e no Anexo A.

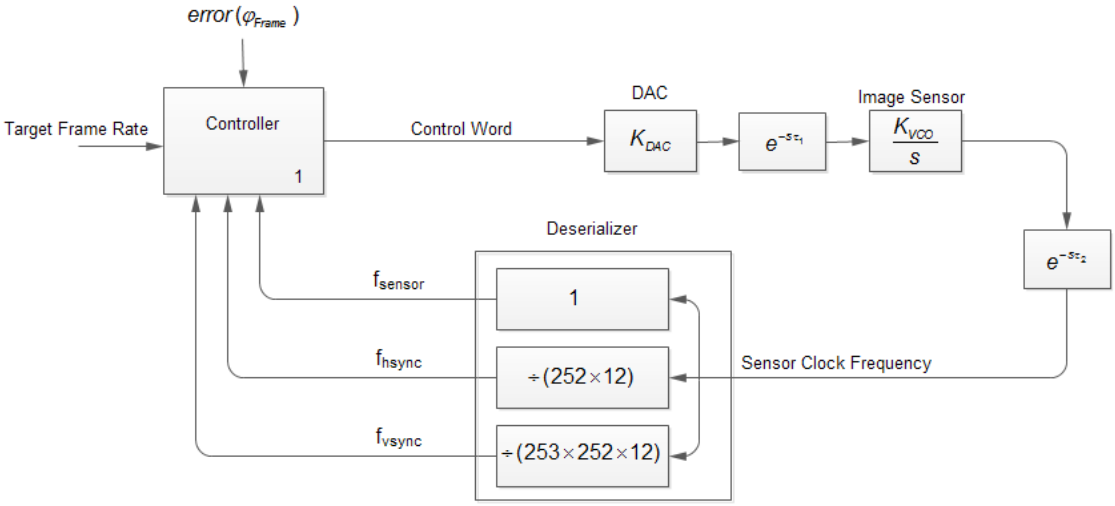


Figura 3.16 – Modelo teórico do sistema de controlo de frequência do sensor de imagem.

3.2.3 Conclusões

Verificou-se que o sensor de imagem NanEye pode ser modelado como um oscilador controlado por tensão. Como tal, a frequência de operação deste é diretamente controlada pela tensão aplicada, sofrendo contudo de uma dependência com a temperatura. Para mais, devido ao processo de fabrico, existem diferenças na sensibilidade do sensor mesmo entre câmaras idênticas.

O controlo do *frame rate* do sensor é feito a partir de uma plataforma de hardware com base numa FPGA. Isto permite implementar um módulo de controlo que introduza o mínimo de atraso possível e controlo preciso da frequência de operação do sensor. Este sistema de controlo deverá ser independente do sensor utilizado, do comprimento do cabo, da temperatura ambiente, e do número de sensores ligados ao módulo de controlo.

3.3. Controlo dinâmico de frequência baseado no período de linha e de *frame* de uma imagem

3.3.1 Introdução

Numa fase inicial, considerou-se o uso de uma PLL totalmente digital como solução para o problema proposto, tendo sido inclusivamente implementado um protótipo cujo procedimento é explicado no Anexo A. Contudo, apesar de uma PLL poder ser uma solução para realizar a sincronização de sinais, neste caso em particular, esta arquitetura apresentou um número de desvantagens (referidas na secção A.5) que impossibilitaram o seu uso. Em detrimento desta solução, optou-se por considerar um sistema de controlo distinto baseado não no sinal de relógio extraído a partir dos dados do sensor de

imagem mas sim nos sinais *vsync*[5] e *hsync*[6], extraídos da deserialização do *stream* de dados. Apesar de ser um método mais rudimentar permite uma implementação mais simples e robusta (comparativamente à solução descrita no Anexo A) numa plataforma FPGA, trazendo vantagens em termos de capacidade ocupada, portabilidade e replicação, garantido simultaneamente a sincronização dos dados provenientes de múltiplos sensores, independentemente da sua versão, temperatura ambiente e comprimento dos cabos conectores.

3.3.2 Topologia

A arquitetura do sistema de controlo de frequência baseado no período de linha e de *frame* de uma imagem é uma proposta original do presente trabalho. É motivada pela necessidade de criar um sistema de sincronização de múltiplas câmaras que use uma única referência fixa, neste caso, um sinal de relógio com frequência constante.

A topologia desta proposta é apresentada na Fig. 3.17. À semelhança da topologia apresentada na secção A.2 do Anexo A, este é também um sistema realimentado. Neste caso, contudo, a sua função é ajustar a frequência de operação da câmara baseando-se no sinal de período de linha, *Line Valid* (LVAL), e ajustar a fase das *frames* com base no sinal de período de *frame*, *Frame Valid* (FVAL). É constituído por um comparador de frequência (no contexto deste trabalho designado por *Frequency Comparator*), um oscilador controlado digitalmente (DCO) e um deserializador.

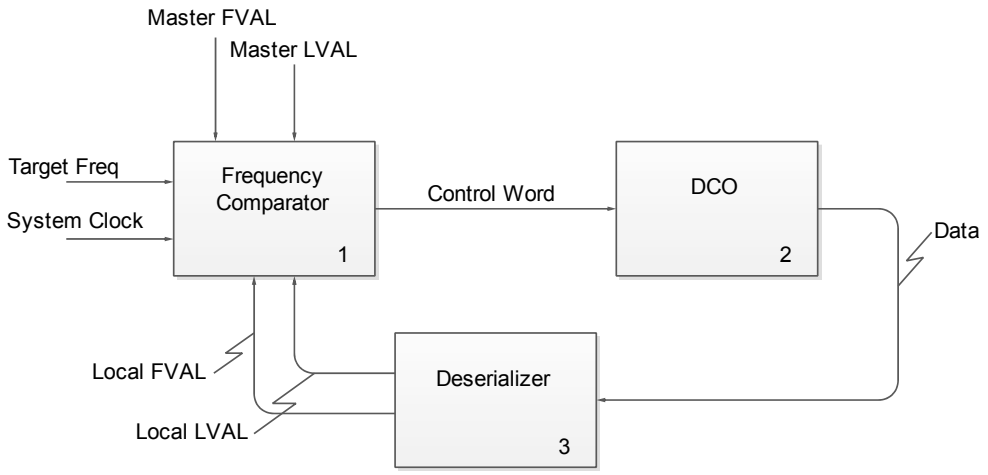


Figura 3.17 – Diagrama de blocos do sistema de controlo de *frame rate*.

[5] *Vsync* – Acrónimo para *Vertical Synchronization* - Sincronização Vertical. No contexto do presente trabalho correspondente ao sinal de período de *frame* do vídeo.
[6] *Hsync* – Acrónimo para *Horizontal Synchronization* - Sincronização Horizontal. No contexto do presente trabalho correspondente ao sinal de período de linha do vídeo.

O *stream* de dados proveniente do sensor de imagem é, numa primeira fase, convertido do formato série para paralelo, e depois descodificado permitindo que à saída do deserializador se obtenham os sinais de período de linha e de *frame*. O bloco *Frequency Comparator* é utilizado para comparar o período do sinal LVAL com um período de referência definido externamente, que corresponderá a um dado *frame rate*. Paralelamente é feita uma comparação com um sinal FVAL externo para determinar o erro de fase entre a *frame* local e a *frame* externa proveniente de outra câmara. Destas comparações resulta uma palavra de controlo utilizada para controlar o DCO e modular a tensão aplicada ao sensor. Sendo que o DCO é uma DAC acoplada ao sensor de imagem que para efeitos de análise comportamental pode ser modelado como um VCO.

Em relação aos trabalhos relacionados, previamente introduzidos no estado da arte [10], [19], [27]–[30], esta arquitetura distingue-se por realizar um controlo ativo da sincronização aumentando ou diminuindo o *frame rate* de cada sensor até que os vários *streams* de vídeo sejam síncronos no tempo. Isto tem a clara vantagem de descartar o uso de elementos de memória para armazenar as *frames*. Para mais torna, teoricamente, o sistema de controlo independente do comprimento do cabo, um fator problemático em [19]. Esta técnica de controlo vai também de encontro à própria arquitetura do sensor uma vez que este é auto-temporizado e como tal não necessita de um sinal de relógio externo nem sinal de *trigger*, dispondo apenas de 4 pinos para conexão.

3.3.2.1 Modelo teórico

O comportamento do Comparador de Frequência apresenta não linearidades devido ao uso de contadores digitais na arquitetura interna do módulo. Isto torna complicada a sua representação segundo um modelo teórico devido à dificuldade na representação matemática de sistemas não lineares. Por este facto, a ilustração do sistema de controlo teve que ser realizada segundo o modelo da Fig. 3.18. Neste a arquitetura do módulo de controlo é representado como o algoritmo descrito na secção 3.3.3 e caracterizado pelo fluxograma da Fig. 3.26.

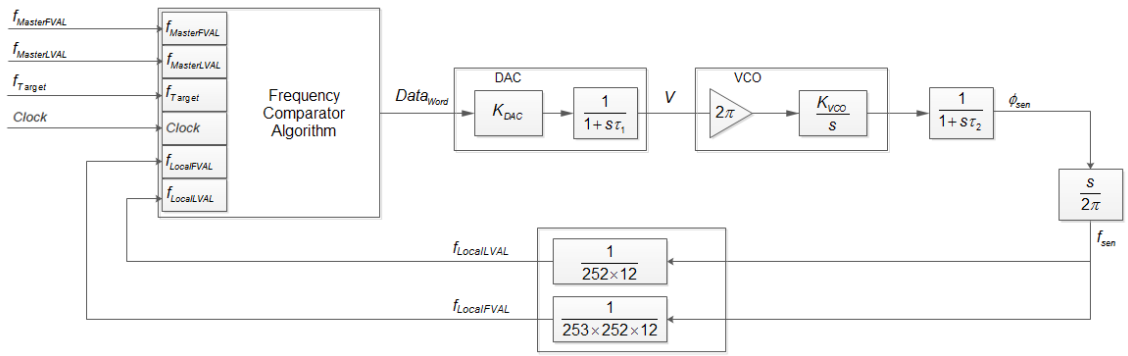


Figura 3.18 – Modelo do Comparador de Frequência.

Contudo esta representação não permite realizar a análise da estabilidade do sistema de controlo. Pelo que foi necessário efetuar uma linearização do modelo de controlo limitando a representação do sistema a um modelo de regime permanente. Neste modelo assume-se que a sincronização em frequência já está realizada (mesmo *frame rate*) sendo apenas analisado o comportamento do sistema na presença de desvios de fase (captura de fase). Como tal, o modelo resultante pode ser representado por uma diferença de fase, pelo ganho e o atraso imposto pela DAC, pela função de transferência do VCO, e pelo atraso provocado pela linha de transmissão (Fig. 3.19).

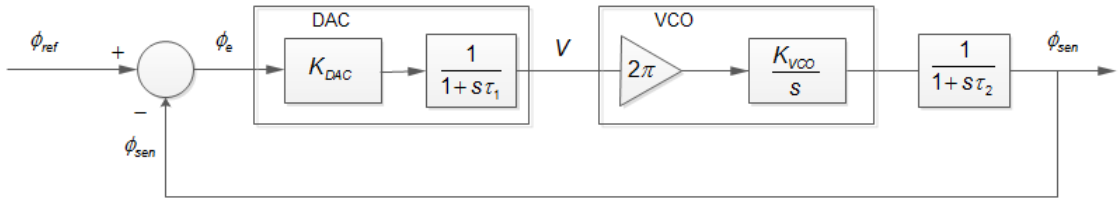


Figura 3.19 – Modelo teórico do Comparador de Frequência em regime não transitório.

Considerando o ganho do sistema em malha aberta,

$$H(s) = 2\pi K_{DAC}\frac{1}{1+s\tau_1}\frac{K_{VCO}}{s}\frac{1}{1+s\tau_2}; \tag{3.26}$$

$$H(s) = \frac{6{,}685\times 10^{4}}{1{,}553\times 10^{-14}s^3 + 4{,}404\times 10^{-6}s^2 + s}, \tag{3.27}$$

é possível extrair os pólos e zeros como sendo,

Tabela 3.2 – Pólos e Zeros do sistema de controlo em regime permanente (Comparador de Frequência).

Zeros	Não existem
Pólos	$s = 0$
	$s \approx -227248$
	$s \approx -2{,}83353\times 10^{8}$

Com base nestes resultados representou-se o diagrama de Bode da Fig. 3.20.

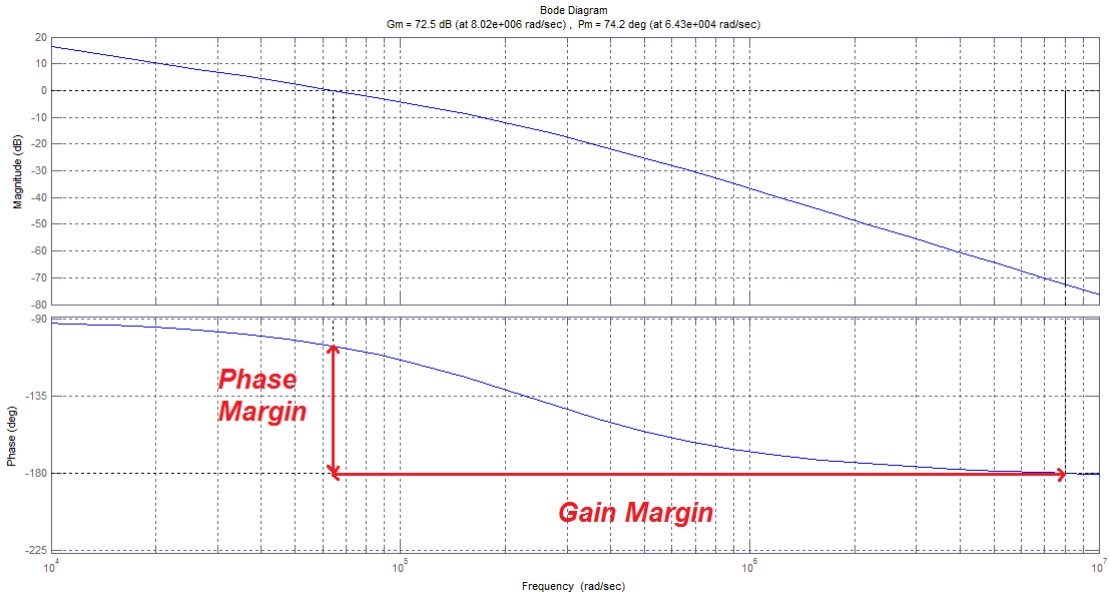


Figura 3.20 – Diagrama de Bode do sistema em malha aberta.

Determinou-se a margem de ganho, G_m, como 72,5 dB e a margem de fase, P_m, como 74,2 º. Esta análise evidencia um sistema com larga margem de estabilidade quando o ganho introduzido pelo comparador de frequência é unitário.

3.3.3 Procedimento de design

Numa fase inicial fez-se o controlo da frequência de operação de apenas uma câmara através do ajuste da tensão aplicada. Como se pode observar pela Fig. 3.17, à saída do *Deserializer* pode-se obter os sinais LVAL e FVAL provenientes da câmara. Numa primeira fase, analisando apenas o LVAL, faz-se uma contagem do período deste utilizando para tal as transições do sinal de relógio comum a todos os componentes da FPGA (*System Clock*). Este processo é ilustrado no diagrama da Fig. 3.21.

Como o sinal de relógio tem uma frequência constante, dada por um cristal oscilador de 48 MHz, pode-se deste modo ter uma referência fixa pela qual se pode determinar o erro da frequência de operação do sensor. Para tal compara-se o número de contagens obtido através deste método com o número de contagens expectável para uma determinada frequência. Sabendo que numa linha são transmitidos 249 PP mais 3 PP a 0, ambos codificados a 12 *bit*, obtém-se a relação,

$$Target_{Clk} = \frac{(12\ bits\ \times\ 252\ PP)\ \times T_{Target\ Freq}}{T_{System\ Clock}}, \tag{3.28}$$

sendo $T_{Target\ Freq}$, o valor do período correspondente a um sinal de relógio com a frequência pretendida para o sensor, e $T_{System\ Clock}$ o período do sinal de relógio de sistema.

Para uma melhor compreensão considere-se o exemplo seguinte: pretende-se que o sensor opere a 44 fps (*frames* por segundo) utilizando como referência um sinal de relógio de sistema com uma frequência de 48 MHz. Segundo o protocolo de funcionamento do sensor NanEye 2C [15] uma *frame* contém um total de 250 linhas mais 3 linhas reservadas para a configuração do sensor, portanto o período de uma linha é dado por,

$$T_{Line} = \frac{T_{Frame}}{253\ lines}\ \ (s), \tag{3.29}$$

substituindo os valores,

$$T_{Line} = \frac{1/44\ fps}{253\ lines} = 89{,}83\ \mu s. \tag{3.30}$$

Pegando neste resultado e substituindo na relação anteriormente deduzida em 3.28 obtém-se,

$$Target_{Clk} = \frac{T_{Line}}{T_{System\ Clock}} = \frac{89{,}83\ \mu s}{1/48\ MHz} \cong 4312\ clocks, \tag{3.31}$$

sendo o resultado o número de contagens que o sistema terá que efetuar para que o sensor de imagem transmita 44 *frames* num segundo dadas as condições acima especificadas.

Se a contagem efetuada estiver acima deste valor significa que a câmara está a operar a uma frequência inferior ao desejado pelo que o módulo de controlo deverá aumentar o valor da tensão aplicada ao sensor para compensar. Por outro lado, se a contagem estiver abaixo do especificado, significa que a câmara está a funcionar a um *frame rate* demasiado elevado pelo que a tensão aplicada deverá diminuir. Assim o sistema poderá ajustar a operação do sensor para qualquer valor dentro da sua gama de funcionamento, permitindo este método que em teoria o sistema de controlo seja independente da temperatura ambiente e do comprimento do cabo que liga a câmara ao módulo de controlo.

Contudo, apesar de este método permitir ajustar a frequência de operação de uma câmara individual para qualquer valor desejado, não garante a sincronização, em fase, de diferentes câmaras. Como ilustrado na Fig. 3.22, ao conectar múltiplos NanEyes em paralelo e colocando-os em funcionamento utilizando apenas o método acima descrito verificar-se-ia que apesar de os mesmos poderem estar a funcionar à mesma frequência, existiria uma diferença de fase entre as *frames* lidas a partir de cada um dos sensores.

Para ultrapassar este problema é necessário então fazer um ajuste na frequência de operação dos sensores para que gradualmente seja ajustada a sua fase com um dado FVAL de referência. Para tal, pode-se fazer pequenos ajustes à grandeza $Target_{Clk}$, somando ou subtraindo um valor por defeito a esta referência, como ilustrado na Fig. 3.23. Considerando um dos sensores como a referência (*Master*) e um segundo sensor como dependente (*Slave*), pode-se ajustar a fase das *frames* entre as câmaras detetando a transição de estado do sinal FVAL local e comparando com a transição do sinal FVAL proveniente de uma outra câmara considerada como a referência, ou *Master*.

Caso a transição do flanco descendente no FVAL local ocorra antes do sinal proveniente do *Master*, significa que o sinal da câmara local está em avanço logo é necessário diminuir a frequência de operação do sensor aumentando o $Target_{Clk}$ por um valor adequado (inicialmente e no caso do exemplo o ajuste é feito de 1 em 1). Por outro lado, se a transição de estado no FVAL local ocorrer depois do sinal *Master*, significa que a *frame* local está em atraso em relação ao *Master* FVAL. Desta forma, é necessário aumentar a frequência de operação diminuindo o $Target_{Clk}$ para corrigir o erro.

Contudo este método apesar de garantir a sincronização em frequência e em fase das *frames*, não tem em conta a velocidade de ajuste consoante a amplitude do erro. Para melhor compreensão do problema tome-se o seguinte exemplo: a *frame* do sensor local (*Slave*) encontra-se em avanço em relação à *frame* proveniente do sensor definido como *Master* por o equivalente a 100 períodos do sinal de relógio. Tal como ilustrado na Fig. 3.24, se a correção da fase for feita por 1 período de *Clock* somado ao $Target_{Clk}$, então verificar-se-ia que o sistema demoraria um tempo considerável a corrigir o erro de fase.

Por este motivo foi necessário criar um mecanismo que permitisse ao sistema de controlo fazer um ajuste rápido da fase e ao mesmo tempo manter a estabilidade e um erro dentro do aceitável. Para tal foi então necessário monitorizar a amplitude do erro entre as duas *frames*. Se o erro fosse muito elevado o ajuste seria mais rápido (algo como 30 períodos do sinal de relógio somados ou subtraídos ao número de ciclos de relógio alvo, $Target_{Clk}$), se por outro lado o erro fosse pequeno o ajuste seria muito mais suave garantido pouca oscilação e um erro reduzido. Como é ilustrado na Fig. 3.25, para alcançar este efeito fez-se uso de uma porta XOR e um contador. À entrada da porta XOR foram colocados os dois sinais de *frame*, *Master* FVAL e *Slave* FVAL. Através desta operação lógica pode-se obter um sinal à saída cujo *duty cycle* é proporcional ao erro de sincronização das duas *frames*. Assim, através de um contador pode-se medir efetivamente a amplitude do erro detetando as transições de um sinal de relógio dentro do período *High* do sinal FVAL XOR.

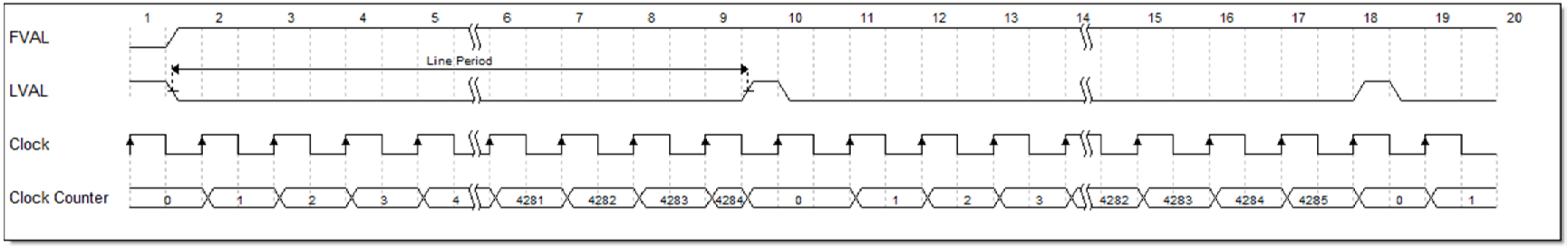


Figura 3.21 – Diagrama temporal da contagem de período de linha.

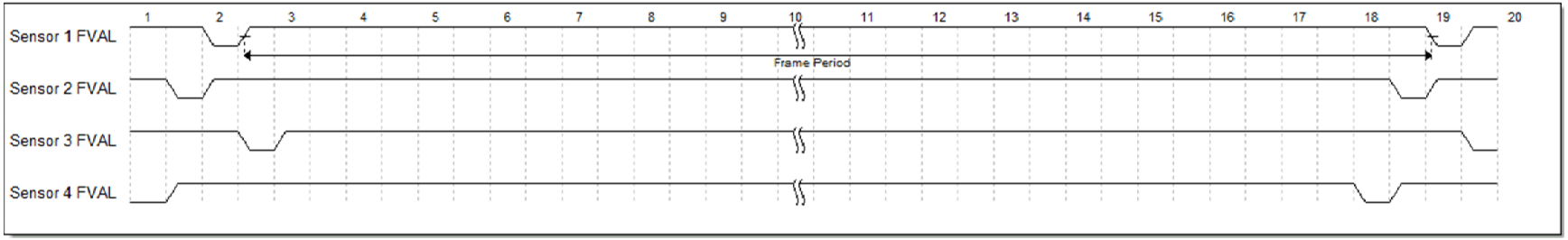


Figura 3.22 – Diagrama temporal da diferença de fase entre *frames*.

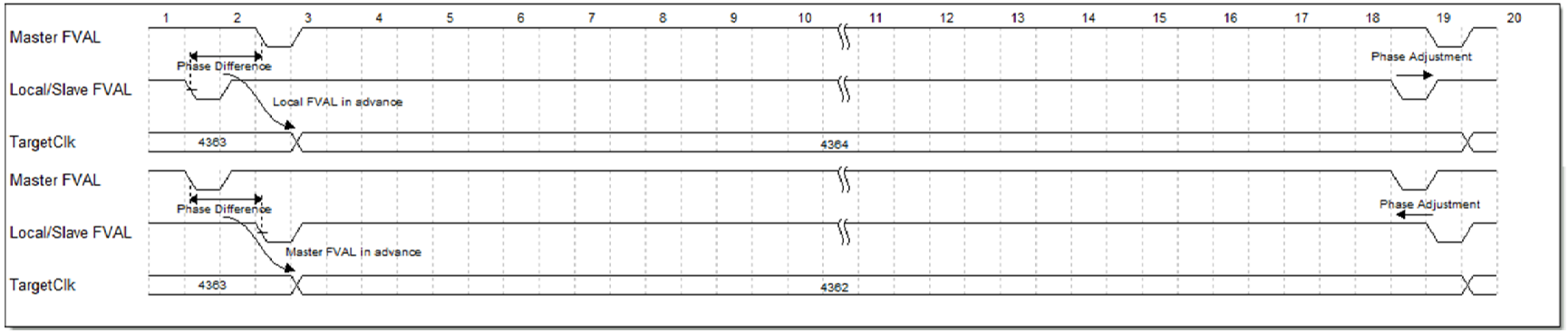


Figura 3.23 – Diagrama temporal do ajuste de fase das *frames*.

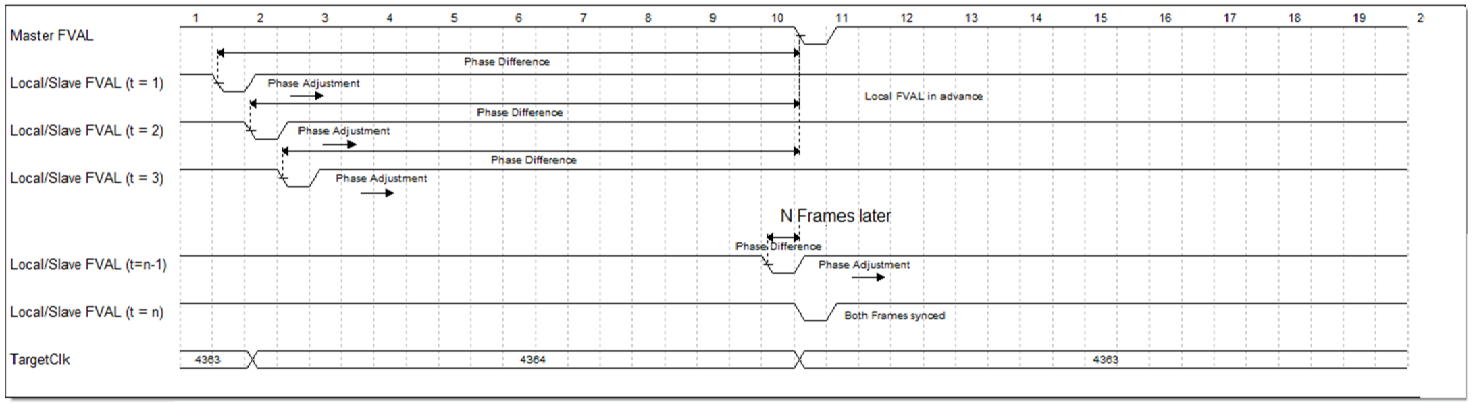


Figura 3.24 – Diagrama temporal do ajuste (lento) de fase entre *frames*.

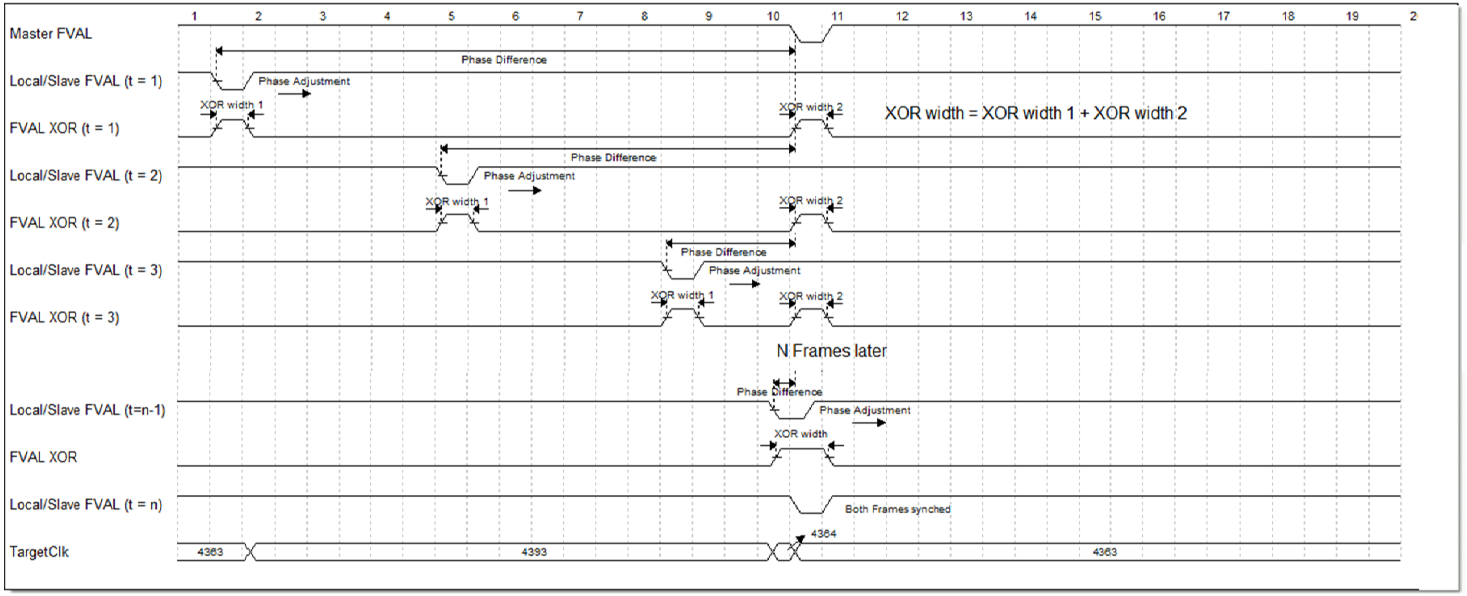


Figura 3.25 - Diagrama temporal do ajuste (rápido) de fase entre *frames*.

Definindo-se um valor de transição entre erro elevado e erro pequeno, pode-se na prática implementar um sistema de correção discreto utilizando passos de correção fixos. Experimentalmente definiu-se a amplitude da correção e o limite de transição entre o passo elevado e o passo mais suave.

No caso do exemplo da Fig. 3.25, o passo rápido é de 30 caso o "XOR *width*" seja maior que o limite estabelecido (neste caso uma vez e meia o período low do sinal Master FVAL), e o passo lento de 1 se menor que o limite. O valor 30, utilizado como passo rápido, foi determinado empiricamente como sendo o valor ótimo que permitia o ajuste mais rápido na presença de um desfasamento elevado sem que o sistema se tornasse instável. Na prática, a utilização destes dois níveis de ajuste provou ser suficiente para garantir a rapidez necessária para a aplicação em causa (o sincronismo é alcançado dentro de 2 s). Contudo, se as exigências da aplicação mudassem poder-se-ia justificar a inclusão de mais níveis de ajuste para diminuir o tempo necessário para alcançar a sincronização das *frames*. Este processo e todo o método de controlo descrito nesta secção encontra-se ilustrado no fluxograma da Fig. 3.26.

Atribuindo-se a uma das câmaras o papel de *Master*, implica que o *frame rate* desta seja diretamente controlado a partir da interface gráfica recebendo os comandos a partir de um PC controlado pelo utilizador. As restantes câmaras conectadas ao módulo são configuradas como *Slaves*, e não são diretamente controladas a partir da interface, em vez disso recebem os sinais de controlo (*Master* FVAL e *Master* LVAL) a partir da câmara *Master*. O sinal de *frame* é utilizado na sincronização em fase segundo o método descrito anteriormente.

O sinal de linha é utilizado pelas câmaras em modo *Slave* para medir a frequência de operação da câmara *Master*, através da contagem do período do sinal. Com base no resultado é ajustada a tensão aplicada ao sensor local de modo a sincronizar a frequência deste com a câmara *Master*.

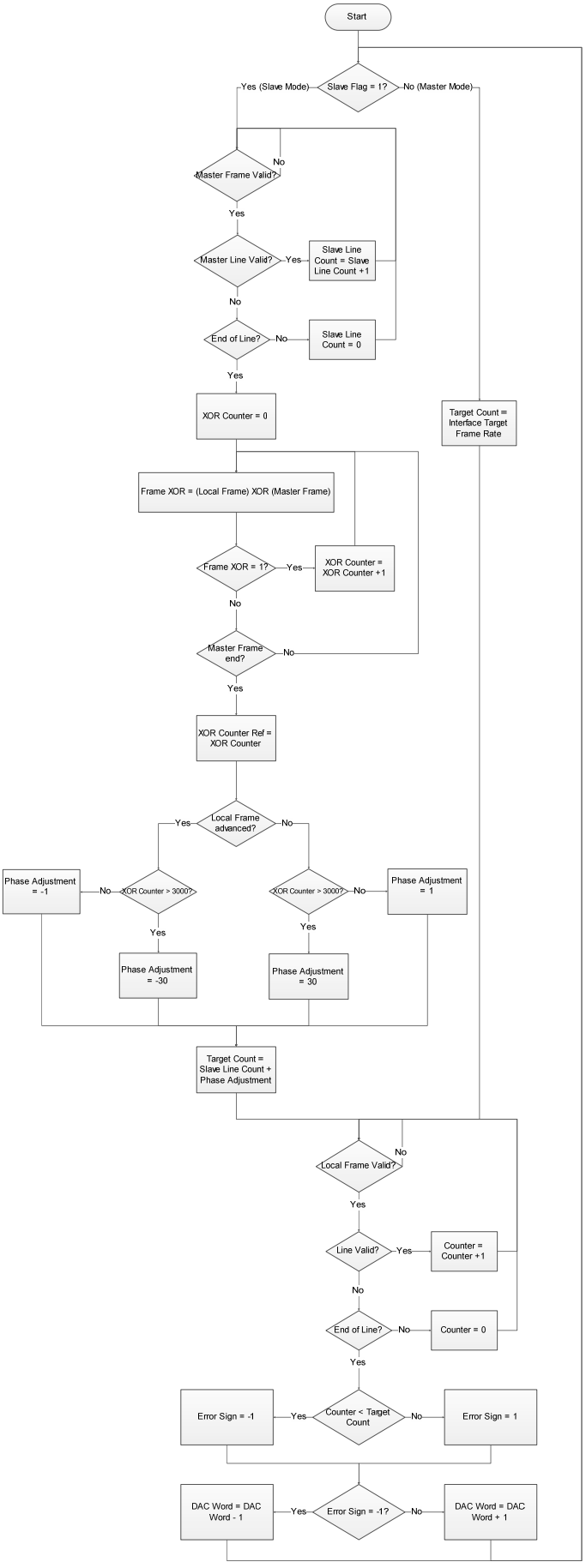


Figura 3.26 – Fluxograma do módulo de controlo *Frequency Comparator*.

3.3.4 Simulação em MATLAB/SIMULINK

Para um melhor entendimento da dinâmica de funcionamento do sistema efetuou-se uma simulação do modelo de controlo no *software* MATLAB/SIMULINK. Procurou-se realizar um modelo de simulação próximo da implementação prática para proporcionar uma melhor compreensão da dinâmica do sistema de controlo já direcionada para implementação prática na plataforma FPGA.

A estrutura geral do modelo implementado na simulação pode ser observada na Fig. 3.27. Este é constituído pelo *Frequency Comparator*, pelo sensor de imagem modelado por um VCO, e por um *Deserializer*. A este modelo é também ligado um bloco representativo de uma câmara *Master*. No Anexo C podem ser consultados os diagramas de blocos individuais de cada um destes componentes.

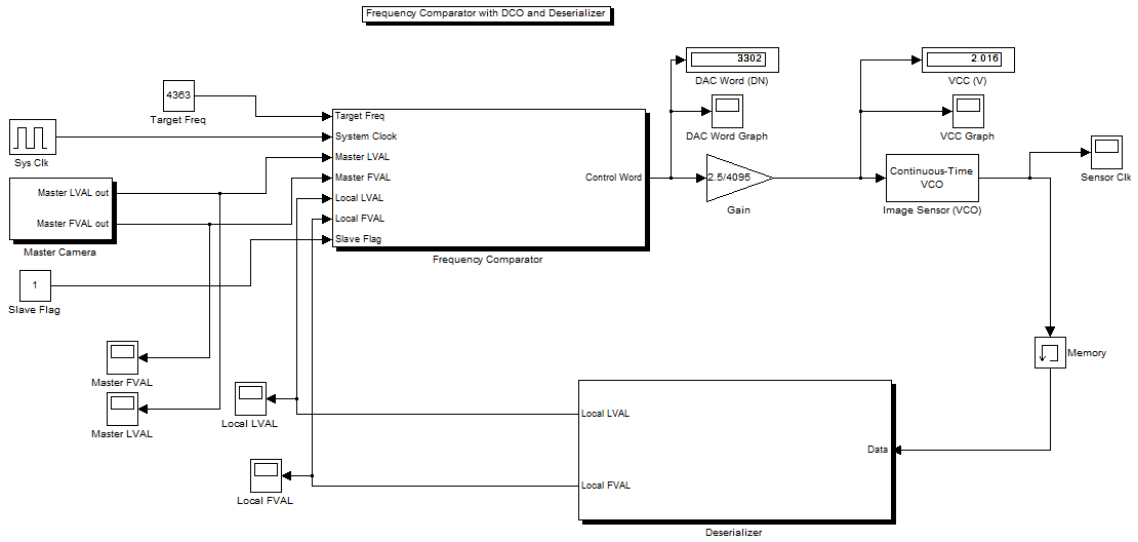


Figura 3.27 – Modelo de simulação da arquitetura de controlo utilizando o *frequency comparator*.

Da mesma forma, os parâmetros utilizados durante a simulação encontram-se nas Tab. C.1 a C.4 do Anexo C. No gráfico da Fig. 3.28 é possível observar o ajuste do nível de tensão para compensar o erro de fase e frequência.

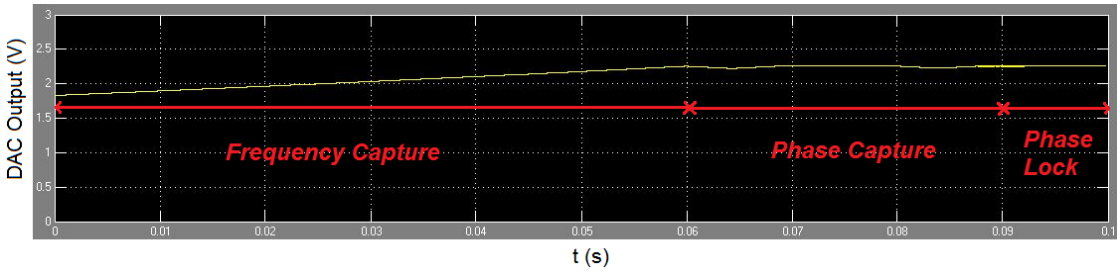


Figura 3.28 – Variação do nível de tensão e ajuste em torno da referência.

Até aos 0,06 s é realizada a captura de frequência, dos 0,06 s até cerca dos 0,09 s o sistema faz a captura de fase, e no restante funcionamento o ajuste

de fase encontra-se estável (*Phase Lock*). Nas Fig. 3.29 até 3.31, encontra-se ilustrado o ajuste de frequência e fase entre a *frame* local e a *frame* proveniente de outra câmara. O tempo de ajuste varia com as condições iniciais do sistema, no entanto para este teste em particular, realizado com os parâmetros das Tab. C.1 a C.4, obteve-se um tempo máximo de 0,2 s.

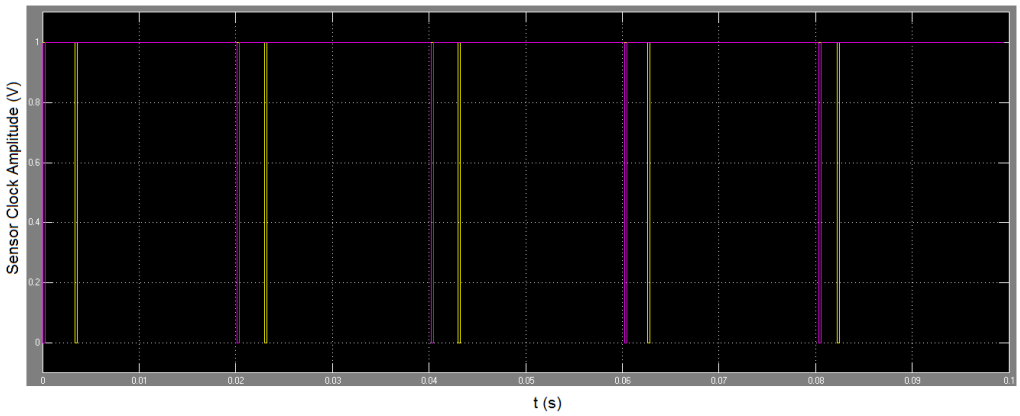


Figura 3.29 – Captura de Frequência.

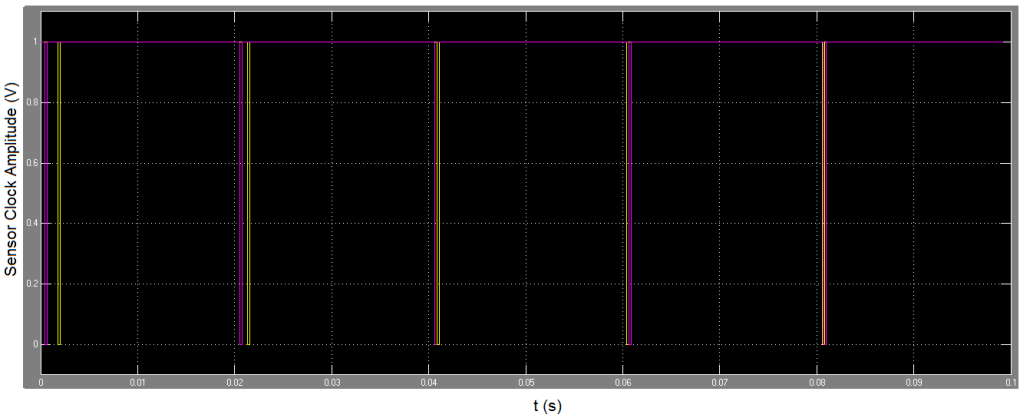


Figura 3.30 – Captura de fase.

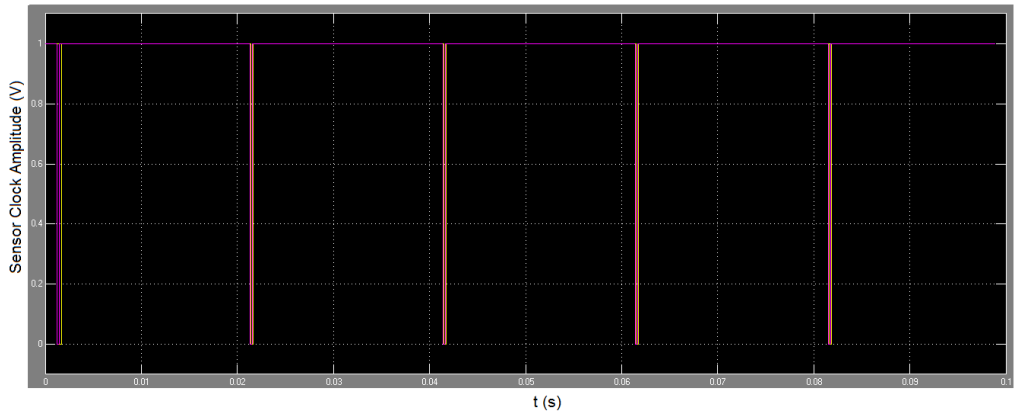


Figura 3.31 – Sincronização de fase (*Phase Lock*).

Quanto ao erro do ajuste de fase, fez-se uma ampliação das formas de onda tal como ilustrado nas Fig. 3.32 até 3.35. Realizaram-se um conjunto de

medições sob diferentes condições de funcionamento tendo compilado os erros obtidos na Tab. 3.3.

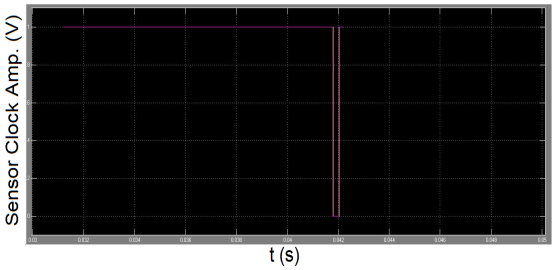


Figura 3.32 – Ampliação do erro de fase (1).

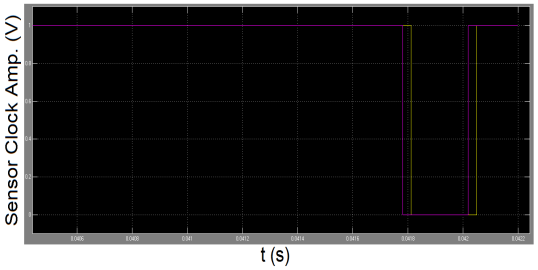


Figura 3.33 – Ampliação do erro de fase (2).

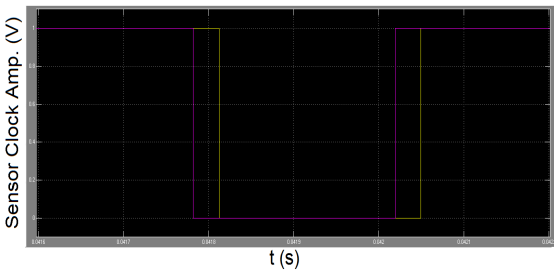


Figura 3.34 – Ampliação do erro de fase (3).

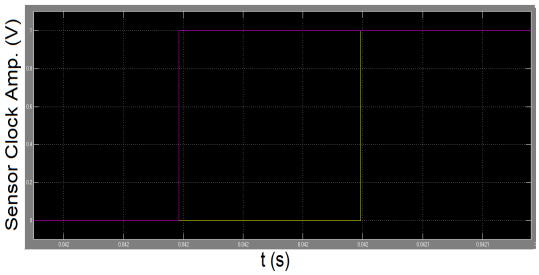


Figura 3.35 – Ampliação do erro de fase (4).

Tabela 3.3 – Erros de fase entre *frames* obtidos via simulação (*Frequency Comparator*).

MATLAB/SIMULINK	
$Max(\varphi_{Error})_{Simulink}$	95,00 µs
$Min(\varphi_{Error})_{Simulink}$	16,67 µs
$Avg(\varphi_{Error})_{Simulink}$	54,63 µs

Da observação dos resultados conclui-se que o sistema de controlo apresenta o comportamento originalmente pretendido, a captura de frequência e fase permite um ajuste rápido e suave do *frame rate* da câmara e com um erro médio baixo tendo em conta o período de uma *frame*.

3.3.5 Implementação na FPGA

A implementação prática na plataforma FPGA foi realizada com base na linguagem de descrição de *hardware* VHDL. Apesar de muito prática e versátil uma plataforma FPGA tem as suas desvantagens. No caso há que ter em conta a sincronização de todos os blocos através de um sinal de relógio comum, e operações matemáticas como divisão ou a multiplicação devem ser limitadas a valores inteiros ou evitadas tanto quanto possível para evitar o consumo de demasiados recursos da FPGA [52]. Todos estes aspetos e outros são inerentes aos sistemas digitais e às FPGAs em particular, tendo sido tidos em conta no projeto e implementação do sistema de controlo apresentado.

Atendendo novamente à Fig. 3.17 da secção 3.3.2 verifica-se que o bloco denominado *Frequency Comparator* deverá receber à sua entrada os sinais *vsync* e *hsync* locais (Local FVAL e Local LVAL), a frequência/*frame rate* alvo (*Target Freq*) bem como um sinal de relógio. Caso o módulo esteja a operar em modo *Slave*, deverá em acréscimo receber os sinais *vsync* e *hsync* provenientes de um sensor configurado como *Master*. Na prática o módulo implementado segue a estrutura representada na Fig. D.1 do Anexo D. Este é constituído por dez sub-blocos representando os processos criados em VHDL sendo estes denominados por: *start_pulse_gen*, *enable_gen*, *line_counter*, *master_slave_diff*, *master_slave_eval*, *frame_diff_eval*, *master_slave_lval*, *target_clk_cnt_eval*, *comparator* e *dac_control*. Os esquemas elétricos destes sub-blocos podem ser consultados no Anexo D, bem como uma descrição geral sobre a função e o funcionamento de cada um destes.

3.3.6 Integração no *firmware* existente

A instanciação do módulo *Frequency Comparator* e a ligação dos sinais de controlo é feita recursivamente através de um ciclo *for*, sendo gerado apenas um módulo para cada um dos sensores acoplados à FPGA. Na geração dos sinais de controlo é definido um dos sensores como sendo o *Master*, definindo a sua *Slave Flag* como “0”, as restantes instanciações do módulo de controlo recebem a *Slave Flag* = “1”. A integração do *Frequency Comparator* com o restante *hardware* segue a estrutura genérica ilustrada na Fig. 3.36.

A arquitetura de controlo e descodificação de dados de um sensor é composta por: um *Frequency Comparator*, encarregue de modular a tensão a aplicar no sensor com base na frequência alvo recebida a partir da interface gráfica (modo *Master*), ou com base nos sinais de referência *Master* FVAL e LVAL (modo *Slave*); um DCO, composto por uma DAC controlada digitalmente e pelo sensor em si representado por um VCO; um *Deserializer* responsável por descodificar o *stream* de dados proveniente do sensor; e por um bloco denominado *Image Out*, que faz a adaptação dos dados de pixéis para transmissão na interface USB.

Os dados do Sensor A são descodificados no deserializador e os sinais *vsync* e *hsync* correspondentes são utilizados como as referências *Master* FVAL e *Master* LVAL nos restantes sensores a operar em modo *Slave*. Os sensores B até H ajustam o seu *frame rate* de modo a seguirem a frequência de operação imposta pelo sensor A. Este último ajusta o seu *frame rate* de acordo com a palavra *Target Freq* recebida a partir da interface gráfica do utilizador.

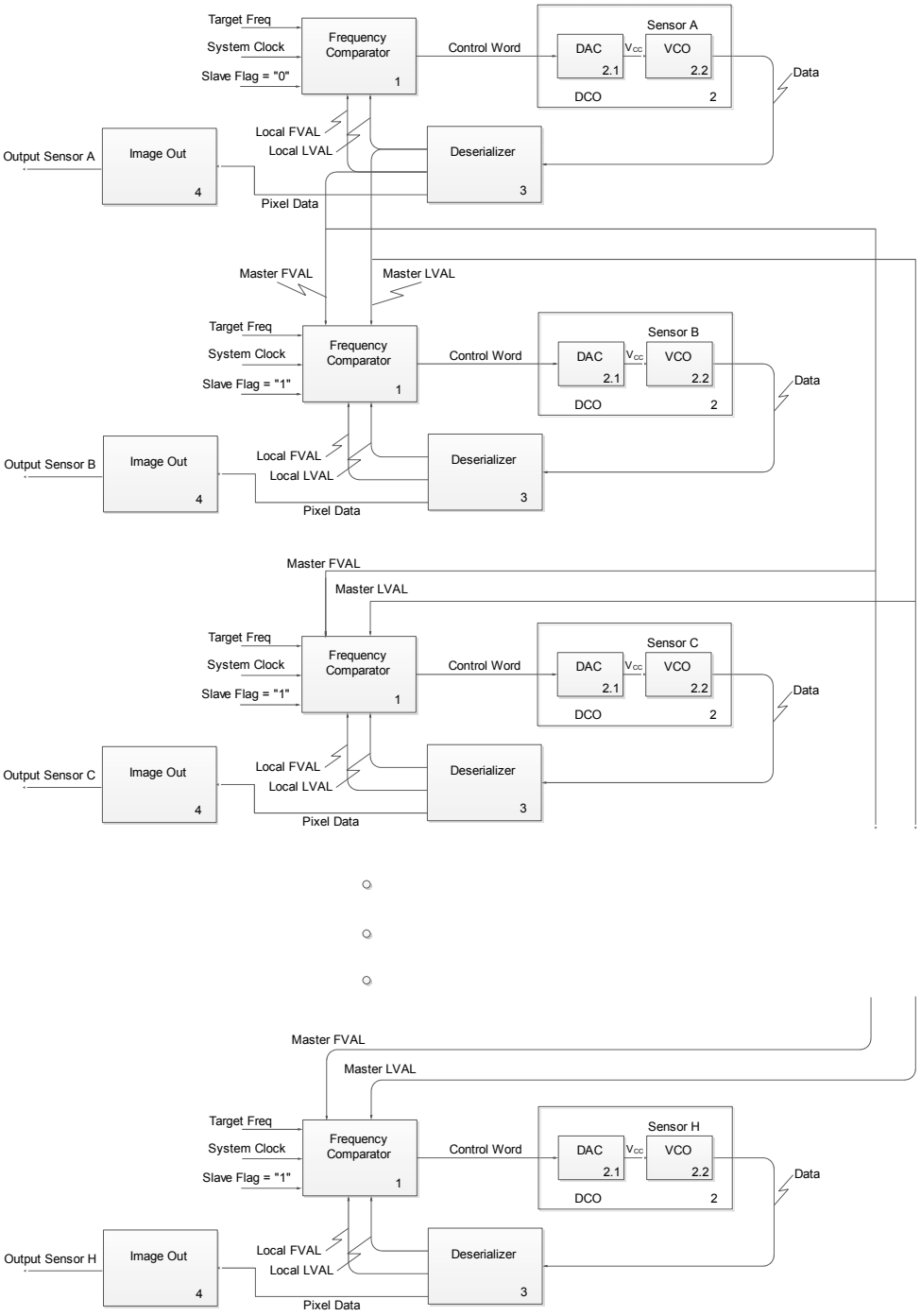


Figura 3.36 – Integração do módulo de controlo *Frequency Comparator* no *firmware* existente.

3.3.7 Resultados obtidos

3.3.7.1 Controlo individual de um sensor (Plataforma NanoUSB2)

Para verificar e comprovar a validade da implementação acima descrita, foi realizada, numa primeira fase, uma bateria de testes ao sistema da Fig. 3.37, constituído por uma plataforma FPGA Spartan 3E com interface USB2, dois sensores de imagem NanEye, na sua versão 2B e também 2C, e um computador pessoal.

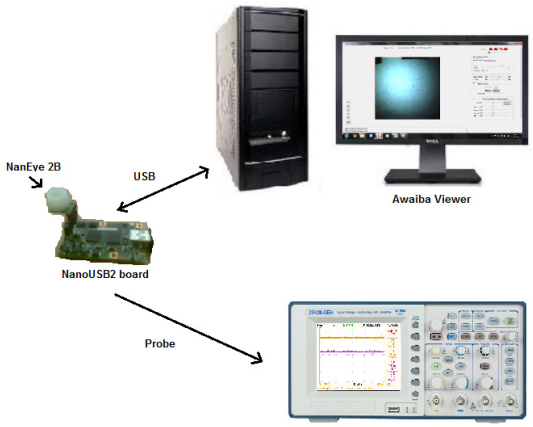


Figura 3.37 – Arquitetura do sistema de teste.

Utilizando um osciloscópio digital testou-se numa primeira fase o ajuste da frequência de operação do sensor para qualquer valor dentro da sua gama de operação. Testou-se o sistema de controlo em duas variantes do sensor de imagem NanEye, na versão 2B, visível na Fig. 3.38 com o involucro branco, e 2C visível na mesma figura com o involucro preto.

Figura 3.38 – Sensores de imagem NanEye, 2C à esquerda e 2B à direita.

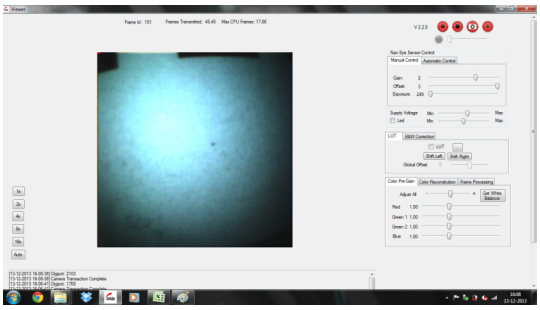
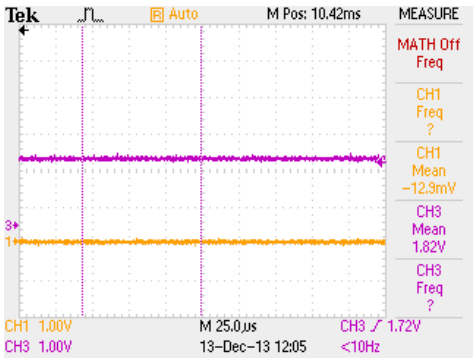
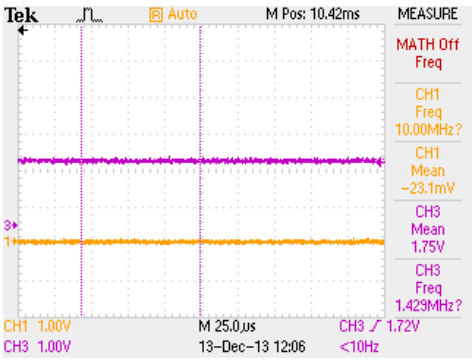


Figura 3.39 – Interface gráfica Awaiba Viewer.

Tomando como exemplo os testes realizados na versão 2C, testou-se o módulo de controlo à temperatura ambiente variando a referência enviada para a FPGA a partir da interface gráfica (Fig. 3.39) e verificando o comportamento do sensor. Nas Fig. 3.40 a) e b) pode-se observar a variação da alimentação imposta ao sensor através da DAC acoplada à plataforma FPGA.



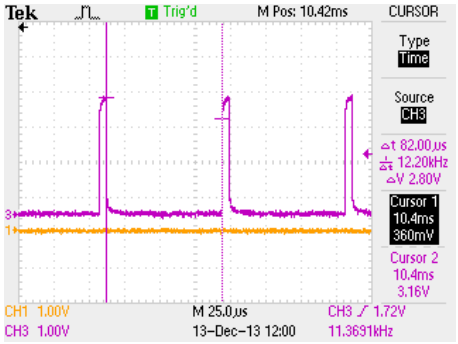
a) Nível de tensão superior (1,82 V) implica frequência mais elevada.



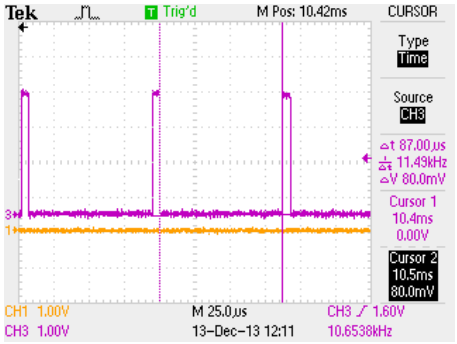
b) Nível de tensão inferior (1,75 V) implica frequência menos elevada.

Figura 3.40 – Tensão de alimentação imposta ao sensor.

A tensão imposta ao sensor implica por sua vez uma frequência de operação mais ou menos elevada visto que o sensor pode ser modelado por um oscilador em anel [15]. As Fig. 3.41 a) e b) demonstram precisamente este efeito sendo nestas representado o sinal *hsync* correspondente no contexto do trabalho ao *Line Valid*. Note-se que este último, na prática, é representado segundo a lógica *active low*[7] como demonstram as figuras.



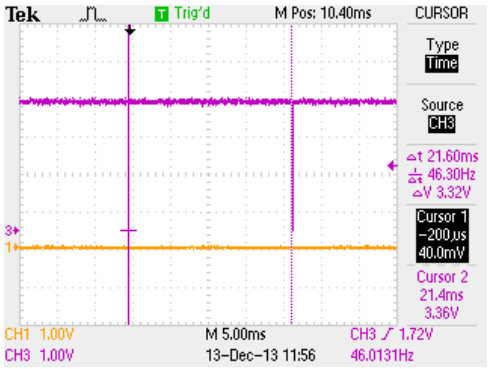
a) **Período *low* de linha menos elevado (82 µs).**



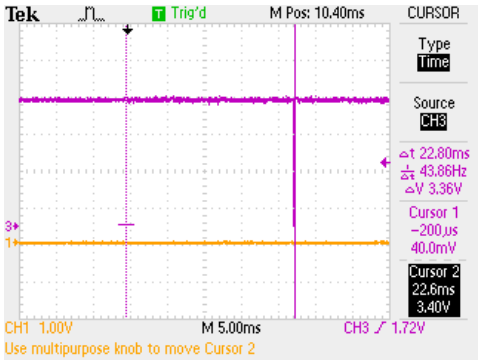
b) **Período *low* de linha mais elevado (87 µs).**

Figura 3.41 – Forma de onda do sinal *hsync* descodificada a partir do *stream* de dados do sensor.

Este mesmo comportamento foi verificado quando se observou o sinal *vsync* (*Frame Valid*), aqui representado pelas Fig. 3.42 a) e b). Ao contrário do caso anterior, o *vsync* é representado segundo a lógica *active high*[7]. No caso do exemplo ajustou-se o comando da interface gráfica para alcançar um *frame rate* de aproximadamente 46 fps e de 44 fps.



a) ***Frame rate* superior (~46 fps).**



b) ***Frame rate* inferior (~44 fps).**

Figura 3.42 – Forma de onda do sinal *vsync* descodificada a partir do *stream* de dados do sensor.

Da observação dos resultados obtidos nesta fase concluiu-se que o sistema de controlo era capaz de modular com sucesso a tensão aplicada ao

[7] *Active Low* - Período ativo é representado pelo nível lógico 0 (*low*) e o período inativo é representado pelo nível lógico 1 (*high*).
Active High - Período ativo é representado pelo nível lógico 1 (*high*) e o período inativo é representado pelo nível lógico 0 (*low*).

sensor de imagem de forma a ajustar a sua frequência de operação para qualquer valor dentro da gama dinâmica.

Contudo, como já foi anteriormente referido, existe uma forte dependência da frequência de oscilação do sensor com a temperatura devido aos efeitos referidos em [16] e [17] pelo que, deste modo, torna-se necessário submeter o sensor a um conjunto de testes sob diferentes condições de temperatura. Submeteu-se as duas versões do sensor a um gradiente de temperatura de cerca de 50 °C, recorrendo a um aquecedor e um *spray* congelante, testando o seu funcionamento em duas plataformas FPGA Spartan 3E distintas, designadas por *Board 1* e *Board 2*, observáveis na Fig. 3.43. No gráfico da Fig. 3.44, e nas Fig. 3.45 a) até f) pode ser observado o comportamento do sensor de imagem (em ambas as versões), para as condições de teste mencionadas. O objetivo deste teste em particular era observar a variação da tensão aplicada ao sensor de imagem (V_{CC}) em função da temperatura ambiente (T) de modo a que a frequência de operação do sensor se mantivesse constante em 48 fps[8].

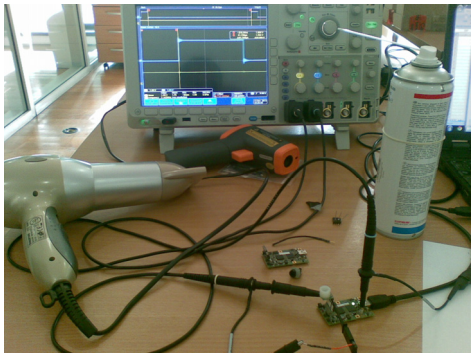


Figura 3.43 – Equipamento utilizado na realização dos testes de comportamento em função da temperatura (plataforma NanoUSB2).

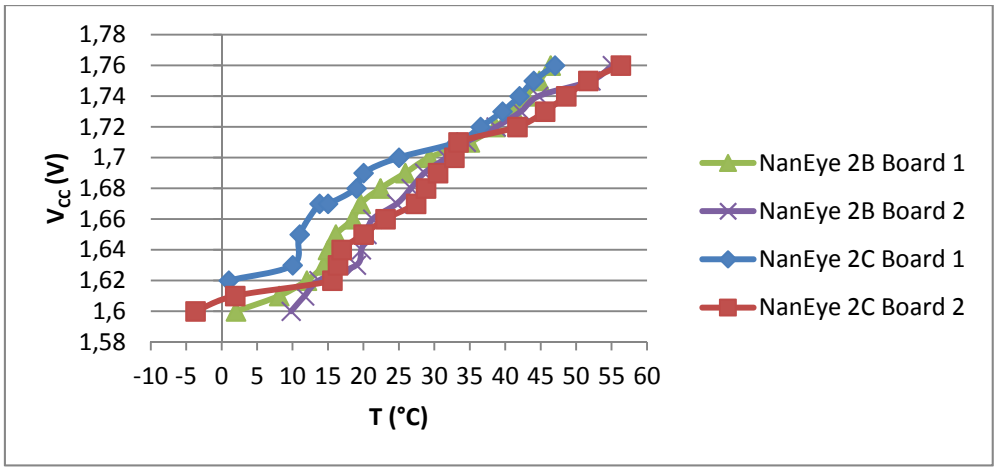
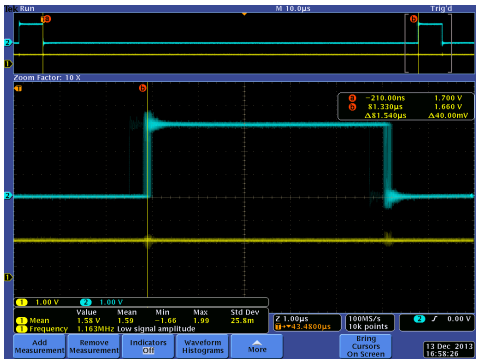
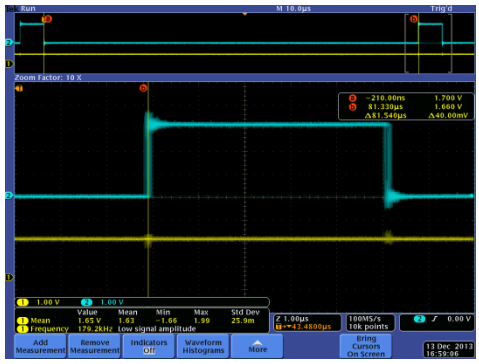


Figura 3.44 – Variação da tensão aplicada ao sensor (versão 2B e versão 2C) em função da temperatura ambiente para 48 fps.

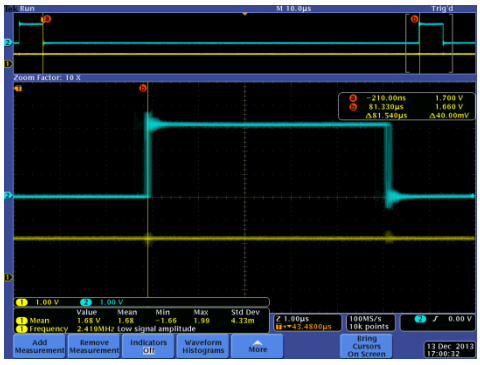
[8] Este *frame rate* foi escolhido para que se pudesse observar uma grande variação da tensão de controlo para uma gama de temperatura alargada, no caso de aproximadamente 0 °C a 55 °C.



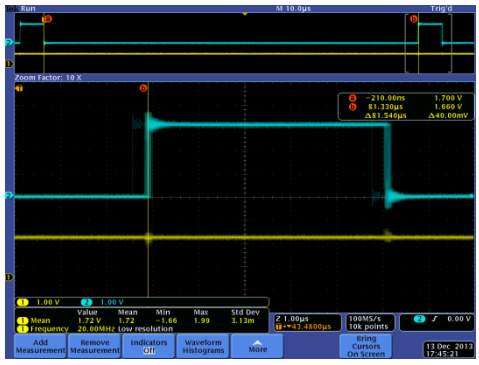
a) V_{CC} = 1,58 V, T = 2 °C



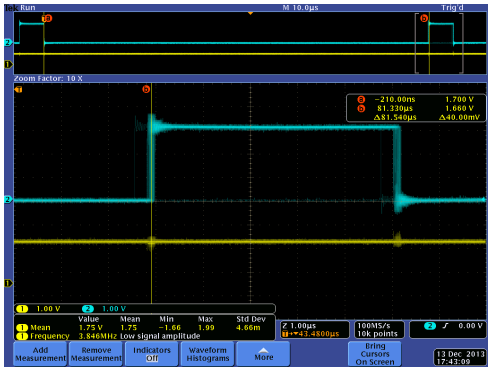
b) V_{CC} = 1,65 V, T = 16,1 °C



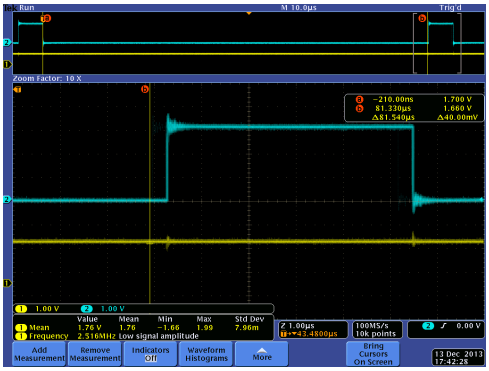
c) V_{CC} = 1,68 V, T = 22,4 °C



d) V_{CC} = 1,72 V, T = 38,6 °C



e) V_{CC} = 1,75 V, T = 44,8 °C



f) V_{CC} = 1,76 V, T = 46,4 °C

Figura 3.45 – Ajuste da tensão de alimentação do sensor de imagem em função da temperatura ambiente (plataforma NanoUSB2).

Pela observação das Fig. 3.45 a) até e) verifica-se que a tensão aplicada pela FPGA ao sensor ajusta-se dinamicamente à temperatura ambiente. A uma temperatura menor a tensão aplicada é também menor, no caso do exemplo, 1,58 V para uma temperatura de 2 °C. Com o aumento da temperatura o sistema ajusta-se automaticamente mantendo um *line rate* constante com um período *low* de 81,33 µs o que equivale à transmissão de dados à cadência de aproximadamente 48 fps. Na Fig. 3.45 f) verifica-se contudo que começa a existir um desfasamento entre o *line rate* pretendido e o medido, devendo-se este facto à limitação da tensão imposta ao sensor em cerca de 1,76 V. Como a tensão aplicada ao sensor não pôde aumentar, o sensor deixou de operar à frequência pretendida devido ao aumento da temperatura, recuperando novamente para a

frequência alvo assim que a temperatura baixou para uma gama de tensão abaixo dos 1,76 V. Este limite de tensão foi imposto para demonstrar precisamente este fenómeno e para comprovar a capacidade do sistema para voltar a recuperar o sincronismo com o *frame rate* alvo quando a temperatura voltasse a estar dentro dos limites de operação do sensor. No entanto isto não significa que o sensor pode apenas operar até temperaturas de 40-50 °C, antes pelo contrário visto que a tensão de alimentação do sensor pode ir até aos 2,4 V sem risco de danificar a câmara [15].

3.3.7.2 Controlo síncrono de dois sensores (Plataforma NanoUSB2)

Como anteriormente foi referido, nesta aplicação é pretendido que múltiplos sensores operem simultaneamente à mesma frequência de operação. Contudo, mesmo que se coloquem ambos os sensores a operar ao mesmo *frame rate*, isto não significa que ambos estejam síncronos, visto que poderá existir uma diferença de fase entre a *frame* extraída de um sensor em relação à *frame* descodificada de uma outra câmara. Experimentalmente observou-se pela Fig. 3.46 que ao colocar 4 câmaras em funcionamento a um *frame rate* idêntico (49,12 fps) se verifica um desfasamento considerável. No caso do exemplo o desfasamento máximo verificado foi de aproximadamente 9 ms (câmara 3, a roxo, em relação à câmara 1, a amarelo).

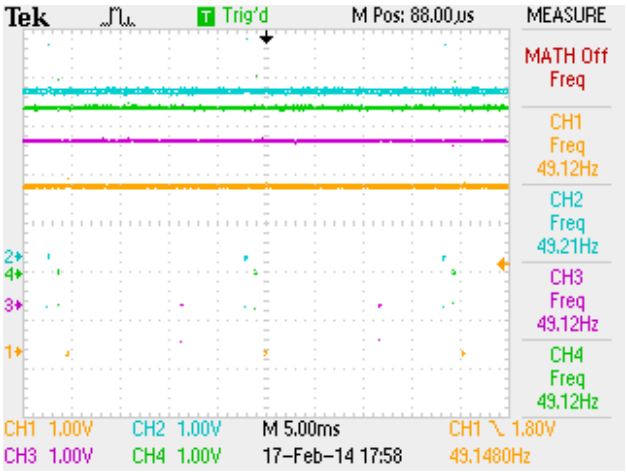


Figura 3.46 – Desfasamento entre as *frames* de diferentes câmaras.

Para ultrapassar este problema foi necessário implementar o algoritmo de sincronismo descrito na secção 3.3.3, o que implica a troca de sinais de controlo *Master*/*Slave* entre as duas placas NanoUSB2 através de portos de entrada/saída configurados como *buffers* bidirecionais[9], tal como ilustrado na Fig. 3.47.

[9] *Buffer* Bidirecional – Elemento de *hardware* utilizado para conectar lógica interna com um pino externo bidirecional [73]

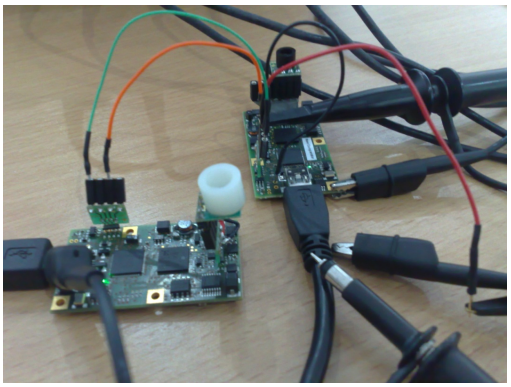
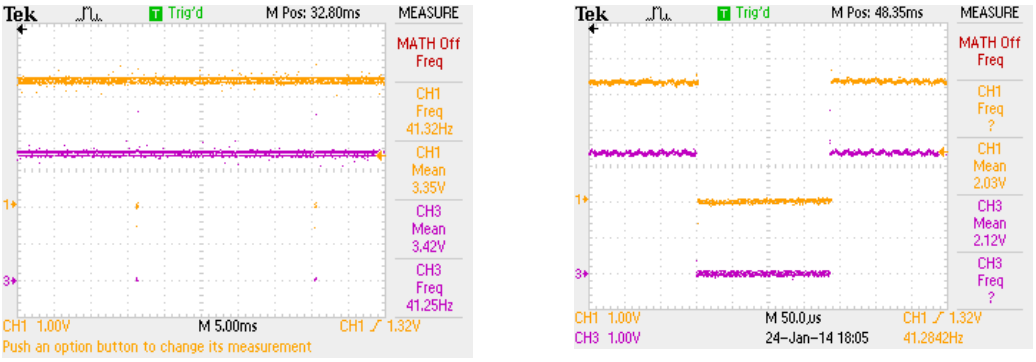


Figura 3.47 – Interconexão de duas plataformas NanoUSB2 para realização da interface *Master*/*Slave* entre os dois sensores de imagem.

Utilizando esta configuração é possível obter o sincronismo em frequência e em fase das *frames* descodificadas a partir dos dois sensores. Na Fig. 3.48 a) pode-se observar a sincronização realizada através desta configuração, e na Fig. 3.48 b) o erro de fase entre as *frames*.



a) b)

Figura 3.48 – Sincronismo em fase e em frequência das *frames* em duas placas NanoUSB2.

Da observação de ambas as figuras concluiu-se que o erro de sincronização é muito baixo. Experimentalmente através de sucessivas medições, mediu-se o erro de fase médio entre *frames*, na plataforma NanoUSB2, como sendo $Avg(\varphi_{Error})_{NanoUSB2} = 3{,}82$ µs. Na Tab. 3.4, inclui-se também o erro máximo e o erro mínimo.

Tabela 3.4 – Erros de fase entre *frames* na plataforma NanoUSB2 (comparativo com simulação).

Plataforma NanoUSB2		**MATLAB/SIMULINK**	
$Max(\varphi_{Error})_{NanoUSB2}$	5,80 µs	$Max(\varphi_{Error})_{Simulink}$	95,00 µs
$Min(\varphi_{Error})_{NanoUSB2}$	0,40 µs	$Min(\varphi_{Error})_{Simulink}$	16,67 µs
$Avg(\varphi_{Error})_{NanoUSB2}$	3,82 µs	$Avg(\varphi_{Error})_{Simulink}$	54,63 µs

Tendo em conta que uma *frame* pode ter um período na ordem das dezenas de milissegundos, este é um erro muito pequeno. Ao comparar-se com a simulação verifica-se que o erro no sistema prático é muito menor. Isto deve-

se ao modelo utilizado na simulação ser mais pessimista que a realidade nomeadamente nos atrasos introduzidos por componentes como a DAC foram assumidos as grandezas referentes ao pior caso.

3.3.7.3 Controlo síncrono de múltiplos sensores (Plataforma DisposcopeUSB3)

Comprovado o funcionamento do sistema de controlo e sincronização, através dos resultados anteriores, procedeu-se à integração e teste do sistema na plataforma DisposcopeUSB3 baseada numa FPGA Spartan 6 com uma interface de dados USB3. Na Fig. 3.49 encontram-se acoplados múltiplos NanEyes à plataforma através de uma *shield*. Sendo a *shield* composta por circuitos de filtragem de ruído e uma DAC para cada um dos sensores.

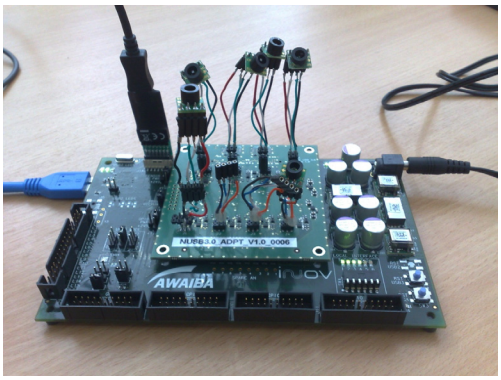


Figura 3.49 – Plataforma NanoUSB3 com múltiplos NanEyes acoplados.

Nesta plataforma o princípio de operação do sistema de controlo e as condições de teste mantêm-se, sendo a única exceção o número de sensores sob teste. Ao contrário da plataforma NanoUSB2 aqui a troca dos sinais de controlo *Master*/*Slave* é feita internamente na FPGA e é possível agrupar as imagens provenientes das várias câmaras acopladas à plataforma numa única interface gráfica, designada por Awaiba Quadviewer, como é o caso da Fig. 3.50.

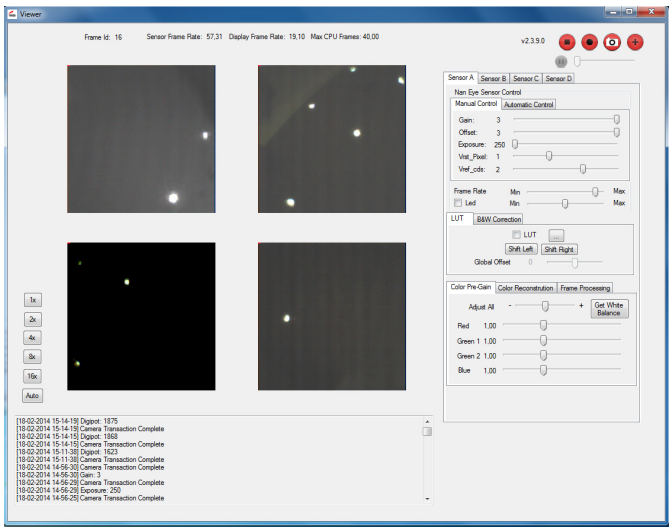


Figura 3.50 – Awaiba Quadviewer.

Conectaram-se as pontas de prova do osciloscópio digital a vários pinos configurados como saídas na placa DisposcopeUSB3 (ver Fig. 3.51). Controlando o *frame rate* diretamente através da interface gráfica pôde-se observar a sincronização das *frames* provenientes de quatro sensores diferentes tal como indicado na Fig. 3.52.

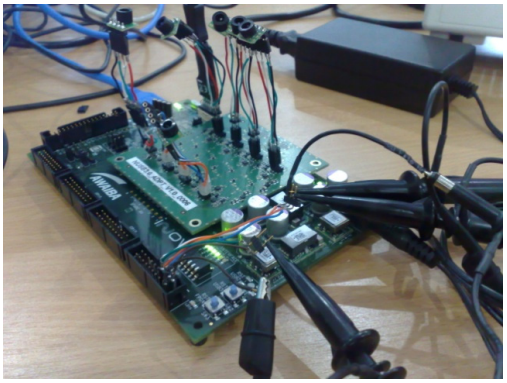


Figura 3.51 – Ligação das pontas de prova à plataforma DisposcopeUSB3.

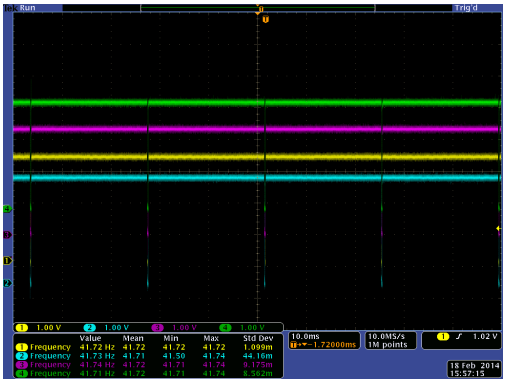
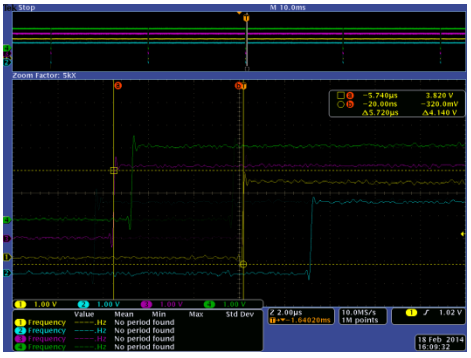
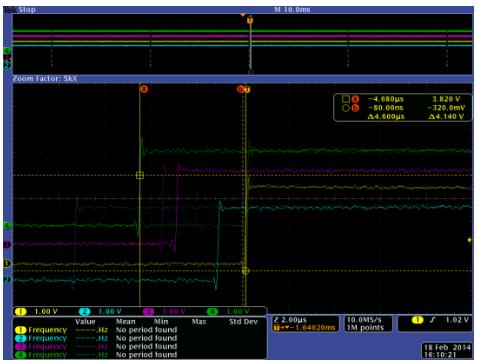


Figura 3.52 – Sincronismo em fase e em frequência de quatro *frames* numa placa DisposcopeUSB3.

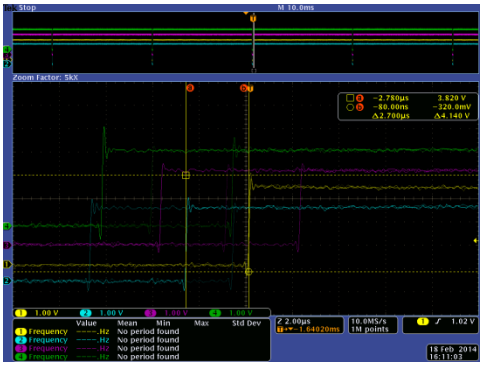
Em relação ao erro da sincronização, seguiu-se o mesmo procedimento realizado com a plataforma NanoUSB2, efetuando sucessivas medições do erro em relação a uma *frame* de referência, neste caso representada pela forma de onda a amarelo no osciloscópio tal como exemplificado nas Fig. 3.53 a) até d).



a) $\varphi_{\mathrm{Error}_{\mathrm{DisposcopeUSB3}}} = 5,72\mu s$



b) $\varphi_{\mathrm{Error}_{\mathrm{DisposcopeUSB3}}} = 4,60\mu s$



c) $\varphi_{\mathrm{Error}_{\mathrm{DisposcopeUSB3}}} = 2,70\mu s$

d) $\varphi_{\mathrm{Error}_{\mathrm{DisposcopeUSB3}}} = 7,52\mu s$

Figura 3.53 – Medição do erro de sincronização entre as *frames* na plataforma NanoUSB3.

Compilaram-se os resultados obtidos da plataforma DisposcopeUSB3 e a estes juntaram-se os resultados adquiridos anteriormente na placa NanoUSB2, formando a Tab. 3.5, contendo o erro de fase máximo, mínimo e médio entre *frames* em cada uma das *boards*.

Tabela 3.5 – Erros de fase entre *frames* em ambas as plataformas de teste.

Plataforma NanoUSB2		**Plataforma DisposcopeUSB3**	
$Max(\varphi_{Error})_{NanoUSB2}$	5,80 µs	$Max(\varphi_{Error})_{DisposcopeUSB3}$	8,76 µs
$Min(\varphi_{Error})_{NanoUSB2}$	0,40 µs	$Min(\varphi_{Error})_{DisposcopeUSB3}$	0,58 µs
$Avg(\varphi_{Error})_{NanoUSB2}$	3,82 µs	$Avg(\varphi_{Error})_{DisposcopeUSB3}$	3,77 µs

Num universo de 22 amostras, mediu-se o erro de fase médio entre *frames*, na plataforma DisposcopeUSB3, como sendo $Avg(\varphi_{Error})_{DisposcopeUSB3}$ = 3,77 µs e na plataforma NanoUSB2 como sendo $Avg(\varphi_{Error})_{NanoUSB2}$ = 3,82 µs. Como seria de esperar os resultados são semelhantes entre plataformas visto que a arquitetura de controlo continua a ser a mesma. Tendo em conta que o *frame rate* nominal de uma câmara NanEye é de 44 fps, isto significa que o erro de fase médio, na plataforma DisposcopeUSB3, é de aproximadamente 0,017%.

Em relação ao comportamento do sistema em função da temperatura, submeteu-se também os sensores de imagem, na plataforma DisposcopeUSB3, a um gradiente de temperatura de cerca de 50 °C, recorrendo a um aquecedor e um *spray* congelante, tal como pode ser observado na Fig. 3.54.

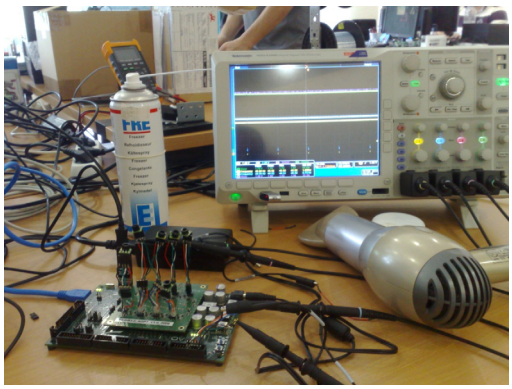


Figura 3.54 – Equipamento utilizado na realização dos testes de comportamento em função da temperatura (plataforma DisposcopeUSB3).

O propósito do teste é determinar se o sistema de controlo permite manter a sincronização das quatro câmaras sob condições de temperatura variáveis. Submeteu-se por exemplo uma das câmaras a temperaturas muito baixas provocando inclusive o seu congelamento (ver Fig. 3.55), enquanto um ou mais dos restantes sensores eram submetidos a temperaturas mais elevadas através de um aquecedor.

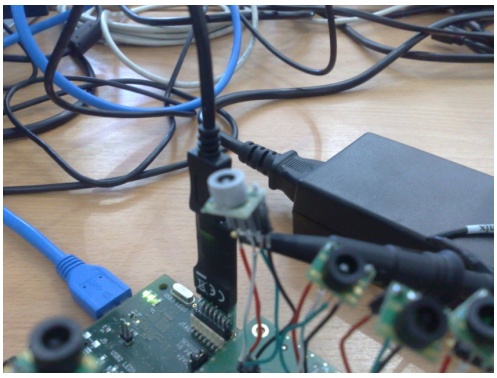


Figura 3.55 – NanEye submetido a uma temperatura inferior a 0 °C.

Na Fig. 3.56 a) a d) pode ser observado o comportamento de dois dos sensores de imagem, para as condições de teste mencionadas. Nas figuras, a violeta e a verde, estão representados os níveis de tensão aplicados aos sensores A e B, respetivamente. A azul está representado o sinal FVAL correspondente ao sensor A, e a amarelo o sinal FVAL correspondente ao sensor B.

a) $T_{Sensor\ A}$ **= 22,9 °C**

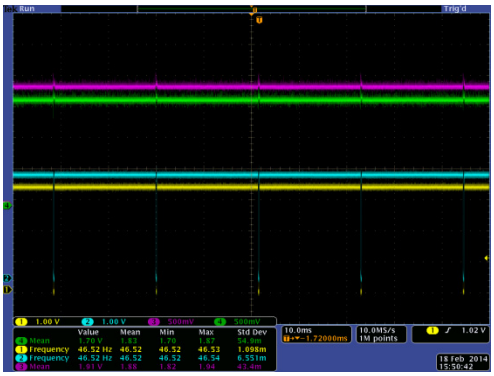
$T_{Sensor\ B}$ **= 22,9 °C**

b) $T_{Sensor\ A}$ **= -1,2 °C**

$T_{Sensor\ B}$ **= 23,1 °C**

c) $T_{Sensor\ A}$ **= 22,5 °C**

$T_{Sensor\ B}$ **= -1,7 °C**



d) $T_{Sensor\ A}$ **= 59,7 °C**

$T_{Sensor\ B}$ **= 0,9 °C**

Figura 3.56 – Ajuste da tensão de alimentação dos sensores de imagem em função da temperatura ambiente (plataforma DisposcopeUSB3).

Para uma melhor compreensão, condensaram-se os resultados obtidos em cada uma destas quatro situações na Tab. 3.6.

Tabela 3.6 – Observações retiradas durante o teste de variação de temperatura na plataforma DisposcopeUSB3.

	Sensor A			Sensor B			
	Temperatura (°C)	Tensão V_{CC} (V)	*Frame Rate* (fps)	Temperatura (°C)	Tensão V_{CC} (V)	*Frame Rate* (fps)	$\varphi_{Error_{DisposcopeUSB3}}$ (µs)[10]
Fig. 3.56 a)	22,9	1,81	46,54	22,9	1,83	46,52	25
Fig. 3.56 b)	-1,2	1,68	46,52	23,1	1,83	46,53	7
Fig. 3.56 c)	22,5	1,83	46,52	-1,7	1,69	46,52	8
Fig. 3.56 d)	59,7	1,91	46,52	0,9	1,70	46,52	4

Na Fig. 3.56 a), ambos os sensores encontram-se a uma temperatura semelhante pelo que o nível de tensão aplicado pelo sistema de controlo a cada um destes é também semelhante o que permite que ambas as câmaras se encontrem em sincronismo de fase e frequência a um *frame rate* constante de 46,5 fps. Na situação b) submeteu-se o sensor A a uma temperatura muito baixa através do *spray* congelante. Como consequência o sistema de controlo diminui a tensão V_{CC} aplicada para compensar, mantendo o sincronismo observado em a). Na situação c), os papeis invertem-se, submetendo-se o sensor B a temperaturas abaixo de 0 °C mantendo o sensor A à temperatura ambiente. Nestas condições continuou-se a verificar o sincronismo entre *frames* com baixo erro. Na última situação, ilustrada na Fig. 3.56 d), submeteu-se o sensor A a uma temperatura superior a 50 °C e o sensor B a uma temperatura próxima dos 0 °C, desta forma alcançando uma diferença de temperatura entre sensores de cerca de 60 °C. Nestas condições continuou-se a verificar o sincronismo de fase e frequência entre *frames* apesar da grande diferença de temperatura.

3.4. Conclusões

Neste capítulo abordou-se o problema e a implementação do módulo responsável pelo controlo de frequência ou *frame rate* dos sensores de imagem NanEye. Foi explicado o procedimento seguido para o estudo, simulação e implementação, de um sistema de controlo dinâmico de frequência baseado no período de linha e de *frame*. Foram apresentados os resultados obtidos para a simulação e implementação dos módulos criados, e justificadas as opções tomadas no desenvolvimento dos mesmos. Foi feito um estudo do sensor de imagem, da arquitetura de controlo, e das plataformas de desenvolvimento utilizadas na implementação do projeto. Concluiu-se que o sensor de imagem

[10] Note-se que este é o erro de fase entre *frames* instantâneo observado em cada uma das situações representadas nas figuras anteriores.

pode ser modelado por um oscilador controlado por tensão, o que implica um comportamento em frequência dependente da temperatura.

O método de controlo, denominado no contexto do projeto *Frequency Comparator*, demonstrou a capacidade de conseguir modular a tensão de alimentação imposta ao sensor de imagem, obtendo o *frame rate* desejado independentemente: da temperatura ambiente, da versão do sensor, e da plataforma FPGA. Revelou ser também capaz de realizar a sincronização de fase entre *frames* com um erro médio de aproximadamente 0,017% do período total de uma *frame*. Contudo o funcionamento do sistema encontra-se limitado pela gama de operação do sensor.

Uma das características desta arquitetura prende-se com a limitação na gama dinâmica de operação do sensor, que encontra-se entre os 1,6 V e os 2,4 V. Isto significa que para extremos de temperatura o *Frequency Comparator* não será capaz de realizar a compensação levando a um erro de frequência que poderá provocar a dessincronização entre os *streams* de vídeo. No entanto, isto não é uma limitação do sistema de controlo em si mas sim dos limites de operação a que o sensor está sujeito.

Foi realizada com sucesso a integração deste módulo de controlo com toda a arquitetura já existente nas plataformas baseadas em FPGA. Nisto inclui-se a seleção e tratamento dos sinais de interesse bem como a receção e interpretação de comandos de controlo recebidos a partir da interface gráfica *Awaiba Viewer*.

4. Algoritmo de controlo de iluminação sobre uma plataforma FPGA

No presente capítulo é discutida a implementação do módulo responsável pelo controlo de uma fonte de iluminação acoplada a um sensor de imagem NanEye. É examinado o procedimento utilizado durante o decorrer do estudo e implementação, de um núcleo de controlo baseado na análise do nível de saturação da imagem captada pelo sensor e respetivo ajuste da intensidade de iluminação emitida pela fonte luminosa. É feito um estudo aos requisitos e condicionantes da aplicação, e explicada a arquitetura de controlo proposta. São também apresentados e discutidos os resultados obtidos em laboratório.

4.1. Introdução

O propósito do sensor de imagem NanEye é o de fornecer ao mercado a mais pequena câmara digital para cirurgia minimamente invasiva e equipamentos de endoscopia de uso singular. Atualmente, a endoscopia é um procedimento médico relativamente indolor, e como tal é um equipamento amplamente utilizado para inspecionar as cavidades interiores do corpo humano, representando um papel importante na medicina moderna [5].

Hoje em dia, equipamentos de endoscopia de baixo custo e reduzidas dimensões incorporam sensores de imagem de pequeno tamanho, capazes de ajustar o ganho, tempo de exposição e intensidade de uma fonte de iluminação (normalmente composta por uma fibra ótica acoplada a um LED) [3]. A intensidade de iluminação do LED e o tempo de exposição têm que ser ajustados dinamicamente em períodos curtos de tempo para garantir uma experiência de visualização suave para o utilizador. Simultaneamente o endoscópio navega em ambientes de iluminação muito variáveis ao longo do tempo à medida que o profissional médico investiga órgãos internos como o esófago, estômago ou o aparelho respiratório.

Uma plataforma FPGA pode ser utilizada para implementar um algoritmo de controlo de iluminação em tempo real, atuando em *LED drivers* de resposta rápida. Realizando o controlo ao nível do *hardware* elimina a latência provocada pelo tempo de processamento, e pela troca de comandos de controlo com uma plataforma de controlo externa. De referir que a resolução e implementação deste projeto foi realizada apenas na plataforma NanoUSB2 introduzida na secção 3.2.2.1.

4.2. Análise do problema

4.2.1 Fonte de iluminação LED

O sensor de imagem NanEye pode ter acoplado na extremidade do cabo uma fonte de iluminação composta por um *array* LEDs (Fig. 4.1). Note-se no entanto, que não é possível controlar cada LED individualmente, mas sim como um conjunto. Alternativamente, a iluminação pode também ser fornecida por uma fonte de iluminação sob uma fibra POF com 1 mm de diâmetro (Fig. 4.2). À semelhança do que foi anteriormente descrito, o sensor de imagem conecta-se à estação base Awaiba (Awaiba *Base Station*) por meio de um cabo de fita plana até 2 m de comprimento. Inicialmente o controlo da intensidade da iluminação era realizado através de um controlo manual disponível na interface gráfica *Awaiba Viewer*. Posteriormente e com o desenvolvimento do módulo de controlo descrito no presente capítulo, a modulação da intensidade de iluminação passou a ser automática.

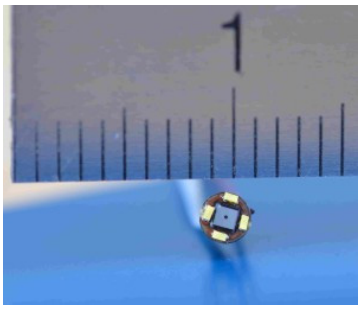


Figura 4.1 – Acessório de iluminação do sensor de imagem NanEye [6].

Figura 4.2 – Câmara digital NanEye com iluminação em fibra [6].

4.2.1.1 Arquitetura

Independentemente do meio de iluminação utilizado (LEDs acoplados diretamente ao sensor ou transmissão da luz por fibra), a arquitetura do sistema de modulação da iluminação é a mesma. Na Fig. 4.3 é possível observar que esta é composta por um PC com o display de vídeo, Awaiba Viewer, pela plataforma de desenvolvimento, NanoUSB2, e pelo sensor de imagem com o acessório de iluminação na forma de um conjunto de LEDs. A modulação da intensidade de iluminação dos LEDs é realizada através do controlo da corrente recorrendo a um *LED driver*. Isto permite linearizar a resposta do LED em função da tensão aplicada por uma DAC de 12 *bit*. Esta encontra-se diretamente acoplada à FPGA sendo comandada via protocolo SPI.

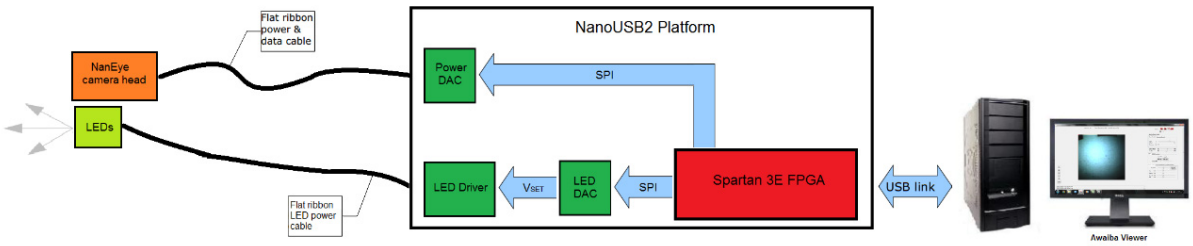


Figura 4.3 – Arquitetura do sistema de iluminação acoplado ao sensor de imagem NanEye.

A. LED Driver

Devido à necessidade de linearizar a resposta dos LEDs em função dos comandos enviados a partir da FPGA, foi necessário utilizar um *LED driver*. Uma forma de realizar isto é acoplar uma DAC a uma fonte de corrente controlada, tal como ilustrado na Fig. 4.4 [74]. A DAC recebe os comandos da FPGA através de uma interface série, e aplica uma tensão proporcional na entrada SET da fonte de corrente controlada, LT3092, da *Linear Technology*. Isto permite controlar a intensidade de corrente que percorre os LEDs entre +VCC LED e GND.

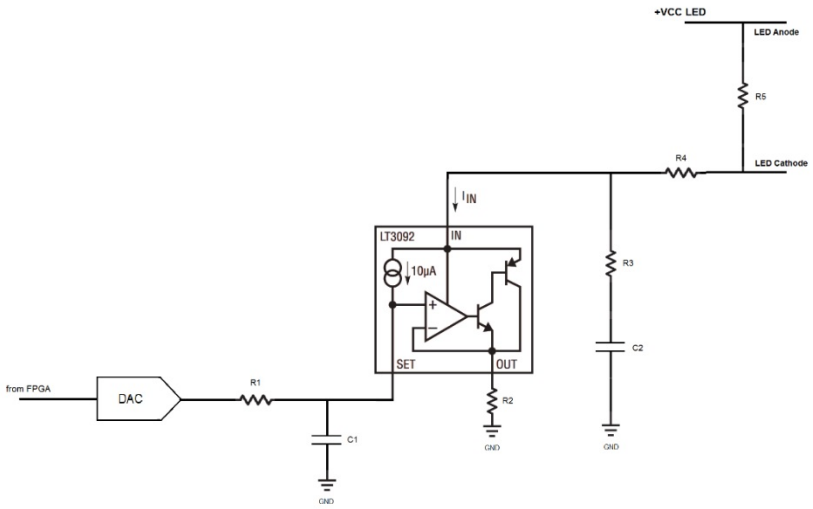


Figura 4.4 – Esquema elétrico do *LED driver*.

B. Relação I/V

A relação entre a corrente que percorre o díodo emissor de luz, I_{LED}, e a tensão aplicada pela DAC, $V_{LED\ DAC}$, foi medida experimentalmente. Para tal variou-se a tensão aplicada pela DAC, à fonte de corrente controlada, desde 0 V até 2,5 V, medindo consecutivamente a intensidade de corrente que percorria o LED. No gráfico da Fig. 4.5, pode-se observar o comportamento observado.

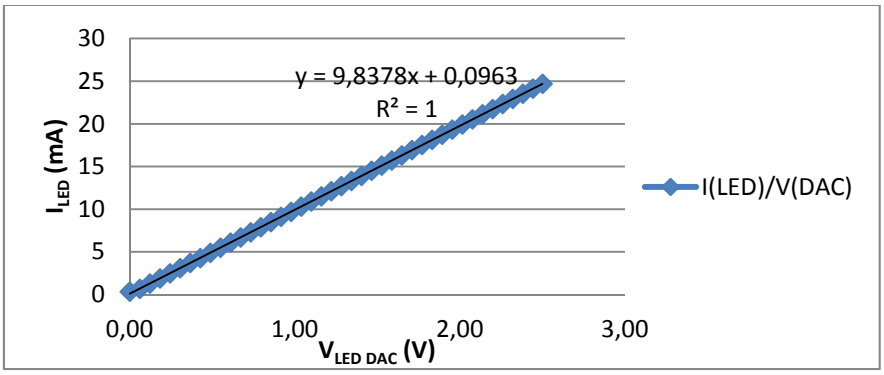


Figura 4.5 – Intensidade de corrente elétrica a percorrer os LEDs em função da tensão aplicada.

Como era de esperar, a corrente aumenta com a tensão aplicada na fonte controlada com uma relação perfeitamente linear ($R^2 = 1$). Desta forma é possível extrair a equação da curva de tendência como uma sensibilidade,

$$K_{LED\ Driver} \cong 9{,}8378\ \left(\frac{mA}{V}\right), \tag{4.1}$$

representando esta a intensidade de corrente que percorre o LED (e consequentemente intensidade luminosa) por unidade de tensão elétrica.

4.2.2 Condicionantes

4.2.2.1 *Target* (cena filmada)

Visto que o sistema de controlo apenas pode avaliar o nível de iluminação através da análise da imagem detetada, implica que o controlo se encontra dependente da cena filmada pela câmara (aqui designado por *Target*). Considerando que o *Target* é uma cena naturalmente escura, tal como o interior do corpo humano, implica que o nível de corrente a percorrer o LED terá que ser necessariamente mais elevado para que o utilizador possa visualizar uma cena bem iluminada. Por outro lado, se a cena for naturalmente clara, como um ambiente exterior, o nível de corrente a percorrer a fonte de luz não será necessariamente tão elevado para que o utilizador possa visualizar a imagem captada pela câmara com um nível de claridade adequado.

Esta realidade implica que a extração da função de transferência do sistema composto pela câmara e a cena filmada é complicada, pois não existe forma de prever qual será a cena a ser captada pelo sensor de imagem. Apesar disto, uma forma de analisar o nível de iluminação do *Target* é através de histogramas (Fig. 4.6) [31]. O histograma indica, da esquerda para a direita, o nível de tons escuros, médios e claros na imagem (nível de iluminação ou brilho), e do topo para baixo, a quantidade de pixéis da imagem distribuídos ao longo destas regiões.

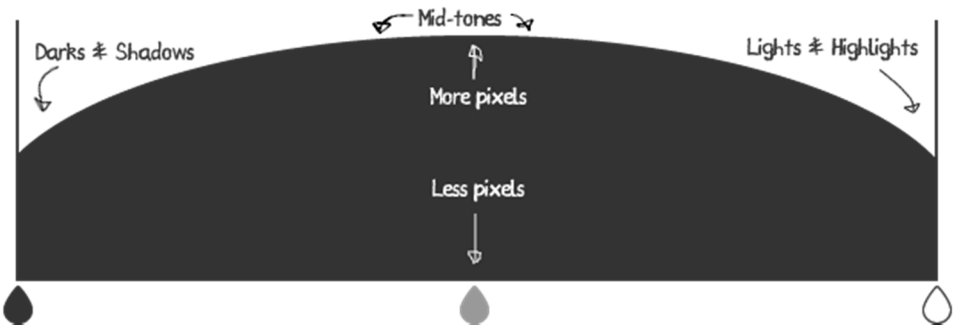


Figura 4.6 – Anatomia de um histograma de brilho [75].

Em rigor, existem histogramas para cada canal de cor, Vermelho (R), Verde (G) e Azul (B). Ao combinar-se os histogramas para cada canal é possível obter um histograma *Red Green Blue* (RGB) que reflete o nível de iluminação da imagem sem desconsiderar canais de cor [31]. O mesmo acontece com o sensor NanEye visto que os filtros de cor já são aplicados diretamente no sensor portanto pode-se tratar a imagem como um único conjunto de *digital numbers*, não tendo que lidar com cada cor individualmente. Na Fig. 4.7 é exemplificado o uso de histogramas na análise de imagens captadas através de uma câmara digital comercial.

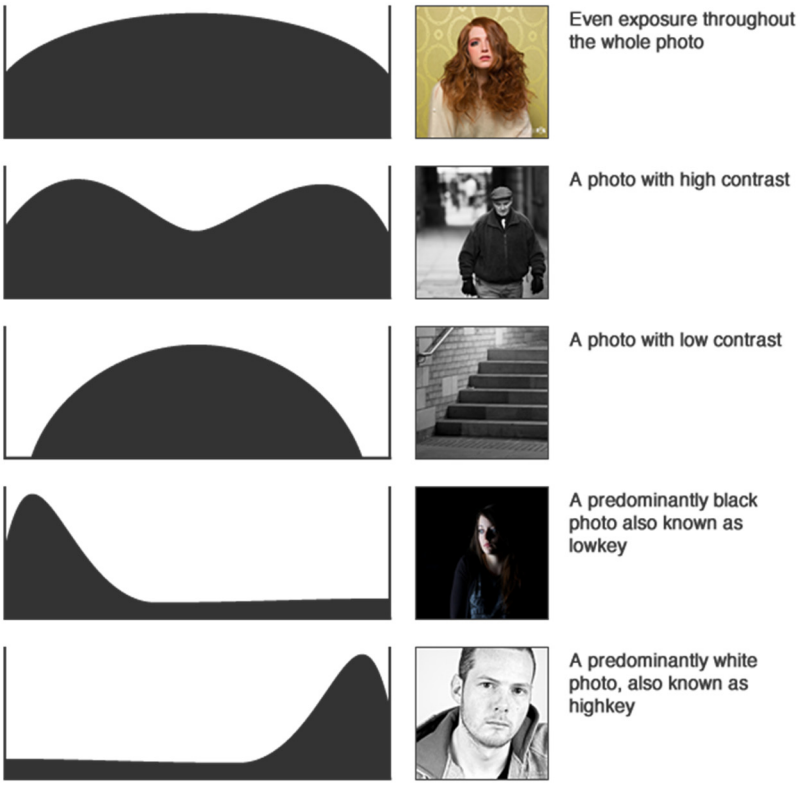


Figura 4.7 – Análise do nível de iluminação através de histogramas [75].

A figura compila cinco situações típicas: uma imagem exposta uniformemente, uma imagem com alto contraste, outra com baixo contraste, uma

imagem predominantemente escura, e outra predominantemente clara. O problema da utilização de histogramas reside no facto que estes apenas fornecem informação acerca da distribuição dos pixéis ao longo da escala dos cinzentos mas não a sua variação espacial. Assim, os histogramas apenas fornecem uma informação parcial acerca da imagem captada [37].

4.2.2.2 Efeito da lente

A lente colocada sobre o sensor produz um efeito indesejável de escurecimento das extremidades da imagem captada pela câmara. Este efeito, visível na Fig. 4.8, é designado por *vignetting* (vinheta). O efeito de atenuação do brilho na direção exterior do centro da imagem pode não ser objetável para o utilizador típico contudo pode influenciar significativamente algoritmos de visão computacional que dependem da precisam dos dados para analisar uma cena [76].

Figura 4.8 – Efeito visual do *vignetting* [76].

O fator proeminente é a queda do nível de iluminação fora do eixo de visão. Esta situação resulta do encurtamento da lente quando vista a partir de ângulos progressivamente superiores em relação ao eixo ótico. Outro fator que contribuí para o efeito do *vignetting* tem uma natureza geométrica. A luz que incide em ângulos oblíquos em relação ao eixo ótico pode ser parcialmente obstruída pelo aro da lente [76].

4.2.2.3 Região de interesse

Uma aplicação comum em sistemas de visão é a definição de regiões de interesse (ROI – *Region of Interest*) que permitem ao utilizador visualizar e/ou melhorar uma parte da imagem para melhor análise [77]. Este tipo de aplicação é especialmente útil em dispositivos médicos, nomeadamente mas não apenas, em endoscopia [5]. Muitas vezes é de interesse ao utilizador selecionar uma pequena parte da imagem captada com o objetivo de proceder a uma melhor visualização de alguma anomalia. Na Fig. 4.9 é possível observar o exemplo deste conceito, considerando a imagem captada por uma câmara digital NanEye com uma área efetiva de captação de luz de 250 x 250 píxeis.

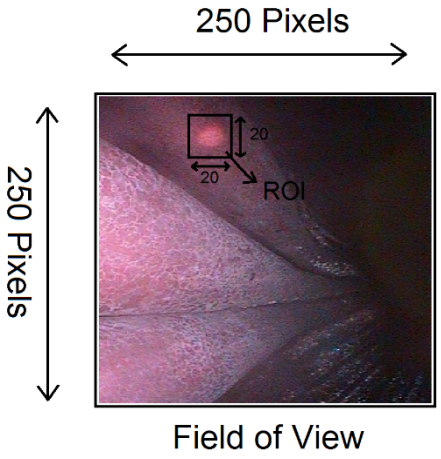


Figura 4.9 – Região de interesse no campo de visão de uma câmara digital.

No contexto de um sistema de controlo de iluminação o melhoramento da visualização passa por uma adaptação da intensidade luminosa apenas à ROI selecionada, que poderá ser, a título de exemplo, a metade superior esquerda da imagem captada que poderá ser afetada pelo efeito do *vignetting*. Isto implica que se o utilizador pretender selecionar uma ROI equivalente a esta extremidade da imagem, apenas a área definida pela ROI estará iluminada adequadamente, mesmo que a restante imagem esteja saturada (sobre iluminada) ou obscurecida (sub-iluminada).

4.2.2.4 Atrasos no sistema

A. <u>V2I LED</u>

Uma das componentes do atraso total no sistema é o tempo entre a aplicação da palavra digital na DAC e a real aplicação da corrente elétrica nos LEDs, τ_{LED}. Este atraso é composto por duas componentes,

$$\tau_{LED} = \tau_1 + \tau_{V2I}\,, \tag{4.2}$$

a 1ª é o atraso introduzido pela DAC, já determinado na secção 3.2.2.3 (note-se que esta DAC é igual à utilizada para controlar a alimentação dos sensores) como $\tau_1 = 4{,}4\ \mu s$. A 2ª é o atraso entre a aplicação da tensão na fonte de corrente controlada e o seu reflexo na intensidade de corrente que percorre os LEDs, τ_{V2I}.

Devido à incerteza associada às características dos componentes presentes no circuito de acionamento dos LEDs, foi necessário determinar o atraso total utilizando métodos experimentais. Assim, mediu-se τ_{LED} utilizando um osciloscópio digital, provocando uma estimulação na DAC e observando a resposta do *driver* em função do tempo. Seguindo esta metodologia determinou-se o tempo de estabilização da corrente como sendo,

$$\tau_{LED} = 19{,}8\ \mu s. \tag{4.3}$$

B. Integração e readout das frames

Consultando [15] é possível chegar à conclusão que o sensor de imagem NanEye realiza a integração e o *readout* das *frames* simultaneamente. Na Fig. 4.10 observa-se que à medida que este faz o *readout* de uma *frame* com um índice *N*, realiza simultaneamente a integração da *frame* seguinte (*N*+1). Isto implica que a *frame N*+1 já está exposta à iluminação anteriormente definida pela DAC quando o sistema ainda está a analisar a *frame* N. Consequentemente, o sistema de controlo apenas deverá atuar de duas em duas *frames* pois uma modificação na intensidade do LED só é refletida passada uma *frame*. Na prática, isto implica que o sensor de imagem introduz no sistema um atraso puro, τ_{sen}, equivalente ao período de uma *frame*.

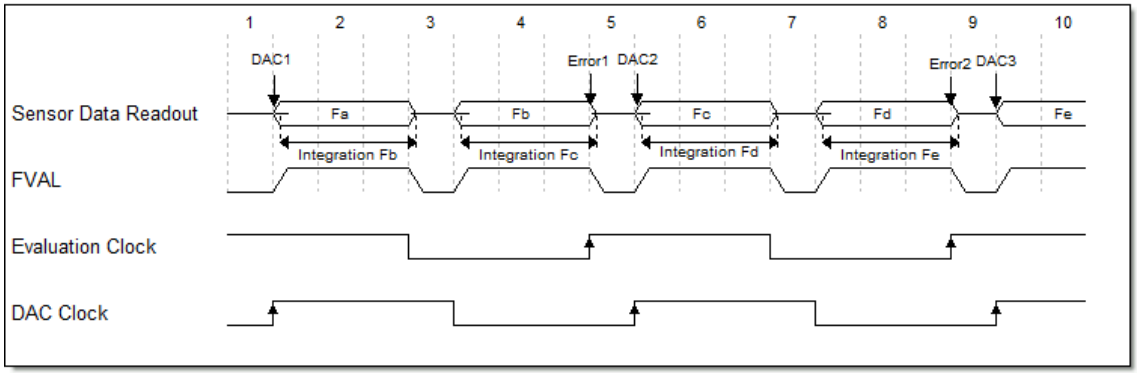


Figura 4.10 – Integração e *Readout* das *frames* na câmara digital NanEye [15].

Considerando um *frame rate* mínimo de 33 fps, definiu-se este atraso como,

$$\tau_{sen} = 30\ ms. \tag{4.4}$$

C. Linha de transmissão

O atraso da linha de transmissão foi já anteriormente calculado na secção 3.2.2.3, tendo sido definido como $\tau_2 = 3{,}53\ ns$.

D. Análise da frame

O último atraso a ter em conta é introduzido pelo próprio sistema de controlo. Visto que, para atuar corretamente na LED DAC é necessário, em primeiro lugar, analisar a *frame* recebida do sensor. Na prática, isto implica que o sistema só pode atuar após a análise de uma *frame* completa. Mesmo ignorando os atrasos introduzidos pelo processo de deserialização e descodificação do *stream* de dados, existe sempre um atraso, τ_{ctrl}, de pelo menos um período de uma *frame*. Considerando novamente o pior caso, definiu-se,

$$\tau_{ctrl} = 30\ ms. \quad (4.5)$$

Tendo também em conta o atraso introduzido pelo processo de integração e *readout* das *frames*, conclui-se que o sistema de controlo da iluminação só pode atuar de duas em duas *frames*, visto que o nível de iluminação definido no instante t só irá ser refletido num erro no instante $t + 2T_{Frame}$, com T_{Frame} o período de uma *frame*.

E. Atraso total

Somando todas as componentes anteriores determina-se o atraso total como sendo,

$$\tau_{total} = \tau_{LED} + \tau_{sen} + \tau_2 + \tau_{ctrl} \approx 60\ ms. \quad (4.6)$$

Os atrasos provocados pelo sensor e pelo módulo de controlo eclipsam as contribuições para o atraso do LED Driver e da linha de transmissão. O resultado é um atraso total bastante significativo o que à partida implica uma compensação muito lenta (forte componente integrativa) devido ao longo período de tempo entre a aplicação de um estímulo no sistema e a sua reflexão na saída.

4.3 Topologia

A arquitetura do sistema de controlo de intensidade de iluminação baseada em histogramas de imagem é motivada pela necessidade de criar um sistema de modulação dinâmica da intensidade de iluminação em linguagem de descrição de *hardware*, cujo espaço ocupado na plataforma FPGA seja minimizado. Assim, devido às restrições de espaço na plataforma de desenvolvimento, procurou-se desenvolver um módulo de controlo que minimizasse a quantidade de lógica utilizada evitando a execução de operações matemáticas computacionalmente intensivas como divisões e operações com vírgula flutuante.

A topologia desta proposta é apresentada na Fig. 4.11. Este sistema realimentado tem como função ajustar o nível de iluminação com base nos histogramas de imagem obtidos a partir da análise dos pixéis transmitidos no *stream* de vídeo da câmara (aqui designado por *Pixel Data*). A análise de cada *frame* é sincronizada pelos sinais de período de linha, *Line Valid* (LVAL), e período de *frame*, *Frame Valid* (FVAL). A topologia é constituída por: um bloco denominado *Illumination Control*; um LED controlado digitalmente, *LED Source*; a cena filmada pela câmara e exposta à fonte de iluminação, designada por *Target*; e a câmara digital em conjunto com o deserializador, agrupado nesta representação como um único bloco denominado *Sensor*.

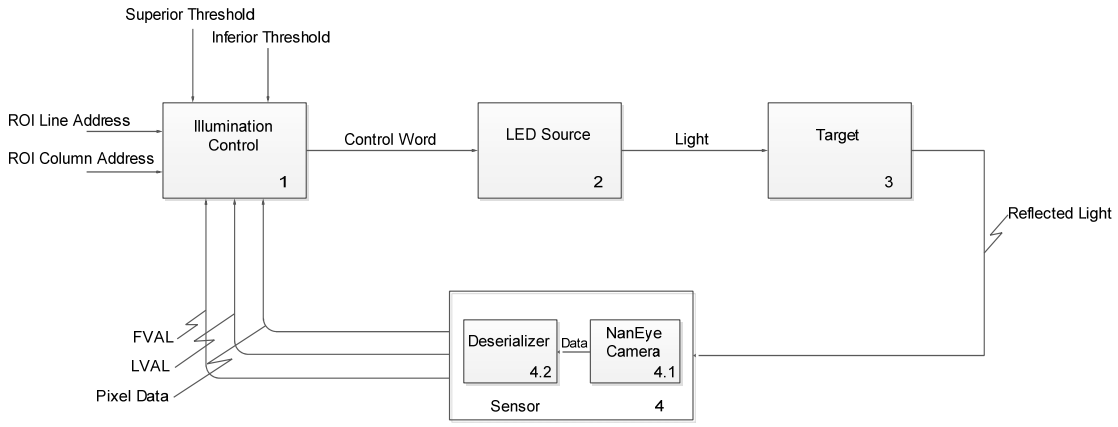


Figura 4.11 - Diagrama de blocos do sistema de controlo de iluminação.

O *stream* de dados proveniente do sensor de imagem é, numa primeira fase, descodificado e deserializado, obtendo à saída três sinais: o sinal de dados de vídeo, *Pixel Data*, o sinal de período de linha, LVAL, e sinal de período de *frame*, FVAL. No bloco *Illumination Control* é feita uma análise do histograma das *frames* captadas com os valores de referência *Inferior Threshold* e *Superior Threshold*, definidos pelo utilizador, de modo a definir um nível ótimo de iluminação. Note-se, no entanto, que apenas a ROI definida pelos sinais *Line Address* e *Column Address*, é analisada pelo *Illumination Control*.

A palavra de controlo à saída deste último bloco é utilizada para controlar a intensidade de iluminação emitida pelos LEDs. Nesta representação este foi denominado por *LED Source* no entanto, como já foi analisado anteriormente, na realidade, este é constituído por uma DAC, um LED *driver* e pelos LEDs em si. Esta fonte de luz ilumina uma cena/objeto/plano, aqui designado por *Target*. Este *Target* tem a particularidade de ser variável pois uma câmara digital pode ser utilizada na filmagem de uma multitude de cenas todas elas com diferentes características em termos de reflexão de luz e de iluminação natural. Isto reflete-se diretamente no nível de iluminação captado pela câmara digital e descodificado como *Pixel Data*.

4.3.1 Modelo teórico

A arbitrariedade da cena filmada pelo sensor de imagem implica a impossibilidade da extração da função de transferência (que neste caso seria a quantidade de intensidade luminosa refletida em direção ao sensor por unidade de fluxo luminoso emitido pelos LEDs em candela por lúmen) deste componente do sistema realimentado. Mesmo assumindo uma cena fixa com uma determinada FT existem ainda outros fatores a ter em conta na formulação de um modelo teórico. Nomeadamente é necessário determinar com precisão a intensidade de luz emitida pelos LEDs. Apesar de já ser conhecida a intensidade de corrente em função da tensão aplicada ao LED *Driver* não se conhece

contudo um fator importante na caracterização de um LED, o fluxo de luminância em função da potência injetada (eficiência do LED em lúmen por watt). Além disto há que ter em conta a função de transferência ótica da lente acoplada ao sensor (*Optical Transfer Function* - OTF) e a função de transferência do filtro de cor que tem uma forte dependência dos tons de cor refletidos pela superfície filmada. Da mesma forma a eficiência quântica do sensor CMOS em si é subordinada ao comprimento de onda da luz refletida pela superfície. Para mais o sistema deverá analisar as *frames* de vídeo tendo por base regiões de interesse contidas na matriz de pixéis que constitui uma *frame*. Regiões estas que são analisadas segundo um histograma formado pela análise da imagem.

Como muitos dos parâmetros envolvidos nos conceitos acima referidos não eram conhecidos, optou-se por representar o sistema segundo o modelo da Fig. 4.12. Neste, a arquitetura do módulo de controlo é representado como o algoritmo descrito na secção 4.4.2 e 4.4.3 e caracterizado pelos fluxogramas das Fig. 4.17 e 4.18, sendo as funções de transferência dos blocos: *LEDs*, *Target*, e *NanEye Camera*, incógnitas devido aos fatores referidos anteriormente.

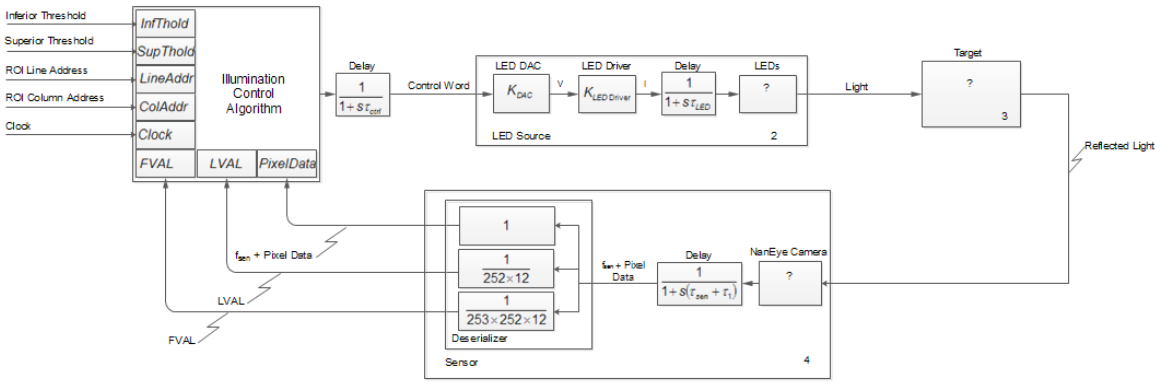


Figura 4.12 – Modelo teórico do sistema realimentado de controlo de iluminação.

4.4 Procedimento de design

Nesta secção será analisado em maior detalhe o procedimento e as opções tomadas no desenvolvimento do bloco de controlo *illumination control*.

4.4.1 Definição da ROI

Como foi referido anteriormente, um dos objetivos deste projeto era a possibilidade do utilizador definir ROIs na imagem captada pela câmara. Permitindo focar a adaptação do nível de iluminação apenas numa região restrita do plano filmado. Em termos práticos isto implica restringir o número de pixéis analisados pelo bloco *Illumination Control*, a uma região definida pelo utilizador na aplicação gráfica *Awaiba Viewer*. Para resolver este problema optou-se por recorrer a dois contadores, aqui designados por *Line Counter* e *Column Counter*.

Observe-se a Fig. 4.13, à FPGA chegam dois endereços para a linha (*Initial Line Address* e *End Line Address*) e mais dois endereços para a coluna (*Initial Column Address* e *End Column Address*). Os contadores ao percorrerem a *frame* de 0 a 250, ativam dois sinais de permissão, *Line Enable* e *Column Enable*, quando a sua contagem se encontra dentro dos limites definidos pelos endereços de inicio e fim. Combinando ambos os sinais de permissão formou-se o sinal *Pixel Evaluation Enable*. À medida que a *frame* é recebida este sinal apenas permite a avaliação dos pixéis contidos no período de ativação deste último.

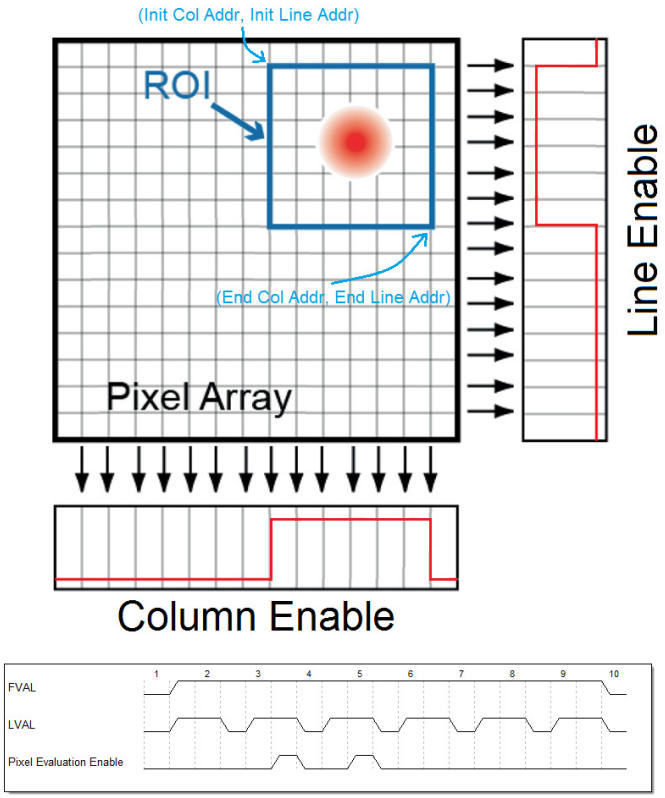


Figura 4.13 – Definição da ROI.

4.4.2 Algoritmo de controlo da iluminação (*DAC Word Control*)

Uma forma de controlar o nível de iluminação de uma cena filmada é retirando a média do valor dos pixéis numa *frame* e atuando no sistema de modo a manter um determinado nível médio de iluminação. O problema desta solução é que poderá haver situações em que uma grande área da imagem poderá estar totalmente saturada, e no entanto o nível médio de iluminação na *frame* mantém-se adequado. A extração de uma média implica também a utilização da operação divisão, algo que se pretendia evitar para minimizar a utilização de recursos.

Outra aproximação é definir níveis de referência no histograma da imagem, designados por *Thresholds*. Ao definir um nível de referência inferior, *Inferior Threshold*, e um nível de referência superior, *Superior Threshold*, é

possível controlar o nível de iluminação a que a câmara se encontra exposta. Tome-se como exemplo o diagrama da Fig. 4.14, onde se encontra ilustrado um histograma de imagem com dois níveis de referência, inferior e superior. Na arquitetura proposta neste projeto, são utilizados dois contadores de pixéis. Um referente ao *Inferior Threshold* e outro referente ao *Superior Threshold*. À medida que a *frame* é analisada o contador inferior acumula o número de pixéis abaixo do *Inferior Threshold* e o contador superior acumula o número de pixéis acima do *Superior Threshold*. Ao controlar a quantidade de pixéis acima do nível de referência superior, é possível manter um nível de iluminação adequado. Por exemplo, pode-se controlar a quantidade de pixéis relativamente ao número total de pixéis na ROI, formando uma percentagem.

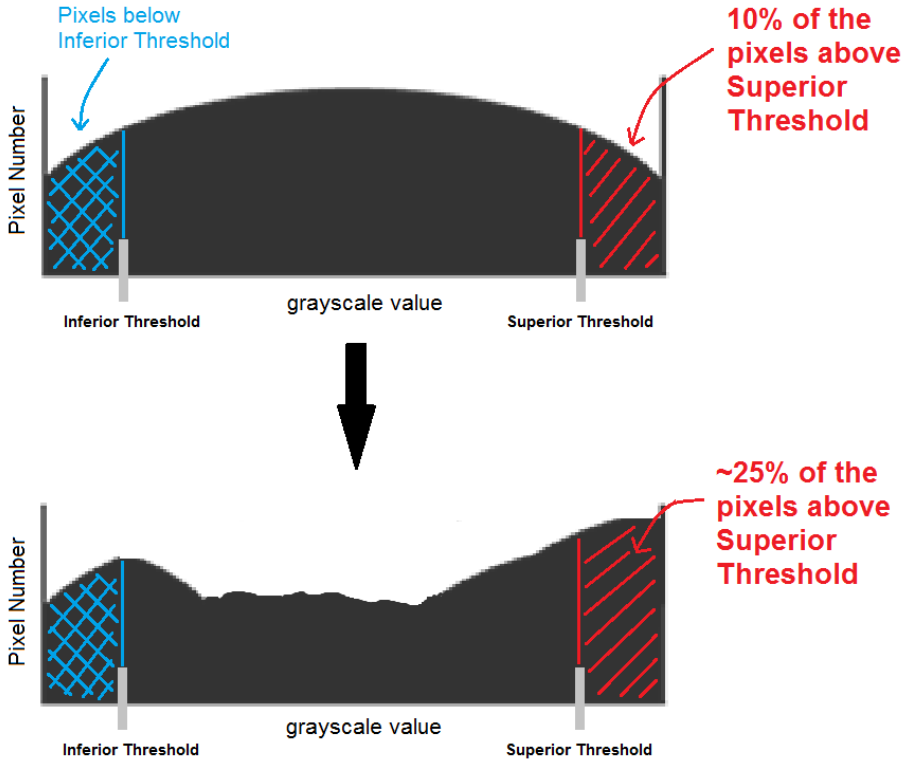


Figura 4.14 – Definição dos níveis de referência no Histograma de imagem.

Pensando em termos de percentagem, pode-se definir, a título de exemplo, que se pretende que 25% do número total de pixéis numa ROI se encontre acima do *Superior Threshold*. O sistema teria então que ajustar gradualmente o nível de tensão aplicada no *LED Driver* de modo a que o nível de iluminação correspondesse a esta percentagem, deslocando a forma do histograma para a direita. Uma vez que a ROI pode ter dimensões variáveis, um erro de 1000 pixéis em relação ao valor correspondente a 25% pode ser perfeitamente aceitável numa região de 50000 pixéis mas o mesmo poderá já não ser verdade numa região com 10000 pixéis. Então, este terá sempre que ser medido em percentagem do número total de pixéis numa ROI e não em valores

absolutos. Deste modo, obtém-se um valor alvo que se encontra sempre referenciado ao número total de pixéis na ROI.

À partida, um sistema realimentado como este necessita de um compensador discreto (PI ou PID) para que se possa acelerar a convergência do ajuste da iluminação sem perder a estabilidade. Contudo alguns dos parâmetros do compensador poderão ter valores não inteiros, implicando operações com vírgula flutuante. Os elementos de vírgula flutuante ocupam um espaço considerável na FPGA, e poderão necessitar vários ciclos de relógio para gerar resultados [52], [73], [78]. Para mais a presença de uma componente integral e derivativa implica uma divisão pelo tempo de amostragem. Uma forma de simplificar esta operação é manter o número de amostras por segundo como uma potência de 2, o que torna a divisão uma simples operação de deslocamento de bit [78]. Porém esta situação não se verificou nas condições deste projeto devido à dependência do tempo de amostragem com o *frame rate* da câmara.

Para contornar este problema desenvolveu-se um algoritmo de controlo para substituir um PID típico. Este algoritmo encontra-se ilustrado no fluxograma da Fig. 4.17. Conhecendo o número total de pixéis numa ROI, *Pixel Number*, é possível determinar a percentagem de pixéis acima do *Superior Threshold* ao utilizar um *Shift Register* para definir vários níveis de percentagem a partir do deslocamento de *bits* da palavra binária *Pixel Number*. Ao utilizar um deslocamento lógico para a direita (acrescentando um '0' como *bit* mais significativo), obtém-se a divisão por 2 para números positivos inteiros, o que no contexto desta aplicação, permite evitar a utilização de um divisor. Uma ilustração deste conceito é apresentada na Fig. 4.15.

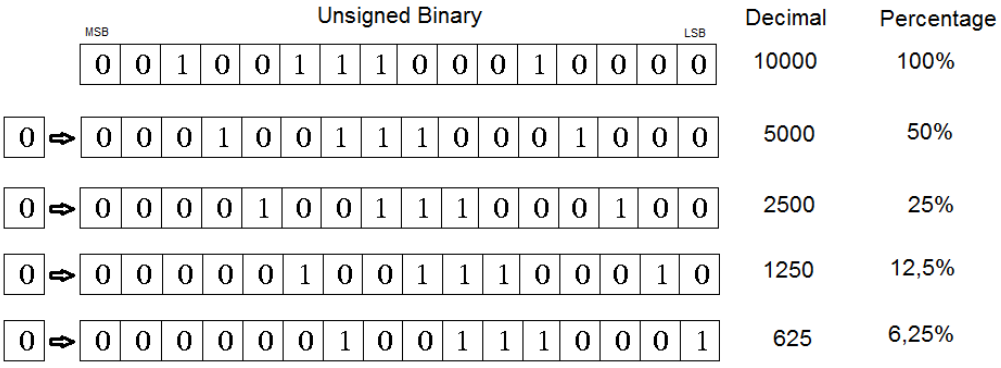


Figura 4.15 – Deslocamento lógico para a direita da palavra *Pixel Number*.

Ao somar-se estas percentagens em várias combinações (ex: 50%+25%) definiram-se os vários níveis, em percentagem, utilizados no algoritmo para determinar o erro relativo em percentagem, tal como no exemplo da Fig. 4.16. A combinação das percentagens permite formar muito mais níveis, no entanto verificou-se que a utilização dos níveis ilustrados na Fig. 4.16 permitem um

comportamento adequado sem acrescentar complexidade excessiva ao algoritmo.

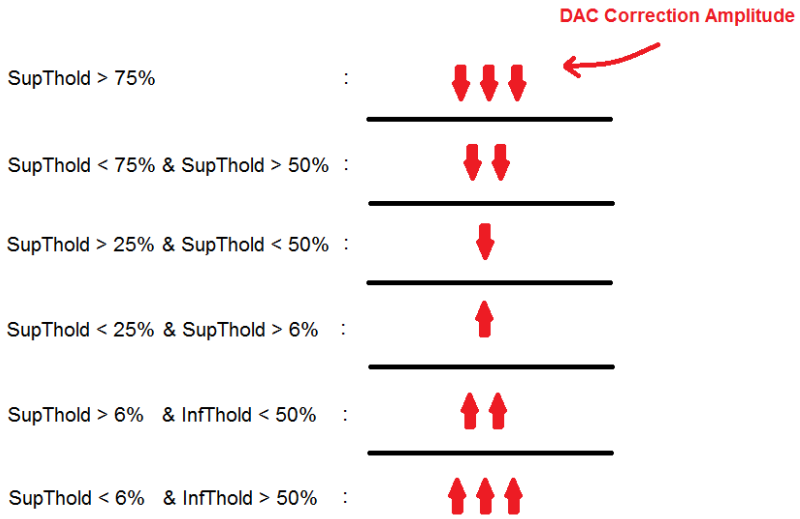


Figura 4.16 – Utilização dos níveis de percentagem na correção do erro.

São utilizados dois contadores para contar o número de pixéis acima do nível de referência superior, *Superior Threshold Count*, e o número de pixéis abaixo do nível de referência inferior, *Inferior Threshold Count*. O valor de cada um destes contadores é comparado com a percentagem relativa e o sistema atua com uma correção do valor da DAC. A correção é no sentido descendente se a quantidade de pixéis acima do *Superior Threshold* for superior a 25%, e ascendente se inferior. A amplitude da correção é dependente do erro em relação a estes 25%. Como tal, esta é mais elevada para grandes desvios, e menos elevada para pequenos desvios. Por exemplo, se a contagem *Superior Threshold Count* for maior que 75% do *Pixel Number*, a correção é no sentido descendente e com elevada amplitude. Por outro lado, se a mesma contagem for maior que 25% e menor que 50%, a correção é no sentido descendente e com baixa amplitude.

A utilização destes níveis de percentagem justifica-se pela necessidade de adicionar uma componente diferencial ao algoritmo sem ter que medir o erro em valores absolutos. Como já foi referido, o utilizador poderá definir ROIs de diferentes dimensões, isto implica que o erro entre o nível de iluminação ótimo e o atual pode tomar valores díspares para ROIs de diferentes dimensões. Por exemplo, um erro de 20 pixéis pode ser insignificante para uma região de 10000 pixéis mas o mesmo já não se verifica para uma região de 100 pixéis (o que deixa também antever uma maior sensibilidade para ROIs de menor dimensão). Atuando com base em valores relativos (percentagens) é possível ultrapassar este problema. Em concreto, o fator diferencial (designado no algoritmo por *Diff Factor*) toma um valor com base na diferença, em percentagem, entre o *Superior*

Threshold Count (N) e *Superior Threshold Count (N-1)*. Com *N* a contagem atual e *N-1* a contagem anterior. Isto significa que o sistema efetua uma correção provocando um estímulo. O estímulo resulta numa diferença que pode ser interpretada como,

$$\begin{cases} |Sup\ Thold\ Count\ (N) - Sup\ Thold\ Count\ (N-1)| < 1\% \Rightarrow Diff\ Factor = 26 \\ |Sup\ Thold\ Count\ (N) - Sup\ Thold\ Count\ (N-1)| < 3\% \Rightarrow Diff\ Factor = 18 \\ |Sup\ Thold\ Count\ (N) - Sup\ Thold\ Count\ (N-1)| < 6\% \Rightarrow Diff\ Factor = 14 \\ |Sup\ Thold\ Count\ (N) - Sup\ Thold\ Count\ (N-1)| < 12\% \Rightarrow Diff\ Factor = 10 \\ Outros\ casos \Rightarrow Diff\ Factor = 1 \end{cases} \quad (4.7)$$

Ao utilizar o *Diff Factor* como um fator multiplicativo na correção, é possível acelerar significativamente a velocidade de correção do erro.

Contudo, para garantir um comportamento adequado e uma estabilização mais rápida foi necessário adicionar também uma componente integradora à correção do erro. Em sistemas discretos isto pode ser realizado utilizando um simples somatório, evitando novamente a utilização de operações complexas como divisões e operações com vírgula flutuante. Na Fig. 4.17 encontra-se exemplificado este conceito. O somatório não foi implementado através de uma soma acumulativa mas sim através da soma de variáveis, isto para evitar o risco de *overflow*/*underflow* da soma.

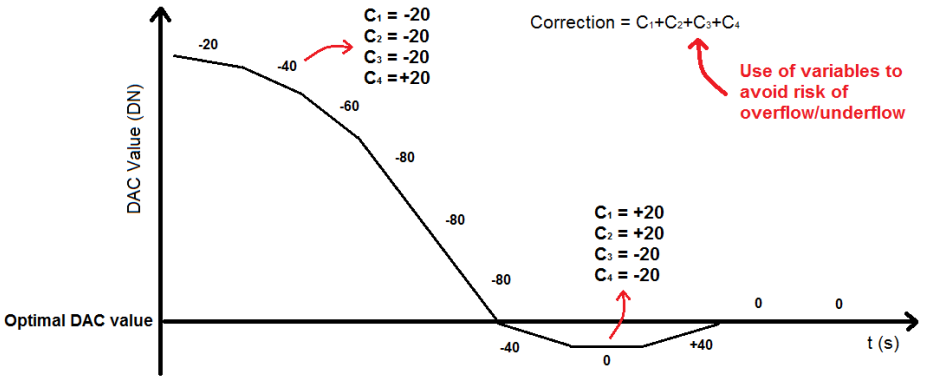


Figura 4.17 – Exemplo da componente integrativa da correção do erro.

Para melhorar a velocidade de estabilização incluiu-se no algoritmo um mecanismo que realiza o *reset* do somatório quando é cruzado o ponto ótimo de funcionamento (*Optimal DAC value*). São utilizados dois sinais de estado, *Above Flag* e *Below Flag*, estes basicamente sinalizam se a sequência de correções anteriores foi no sentido descendente ou ascendente e permitem ao sistema detetar oscilações de elevada amplitude no valor escrito na DAC. Isto diminui o tempo de estabilização evitando oscilações que se tornam visíveis ao utilizador na forma de um *flickering* da fonte de luz. Apesar disto, no fundo, este algoritmo

atua na dinâmica do sistema da mesma forma que um PID, implementando, por exemplo, fatores integrativos através de somatórios, e componentes diferenciais atuando na DAC com base em erros medidos em percentagens relativas. Este processo e todo o método de controlo descrito nesta secção encontra-se ilustrado no fluxograma da Fig. 4.18. Optou-se por realizar uma representação mais simplificada do algoritmo com o objetivo de proporcionar um entendimento claro sobre o essencial do seu princípio de funcionamento.

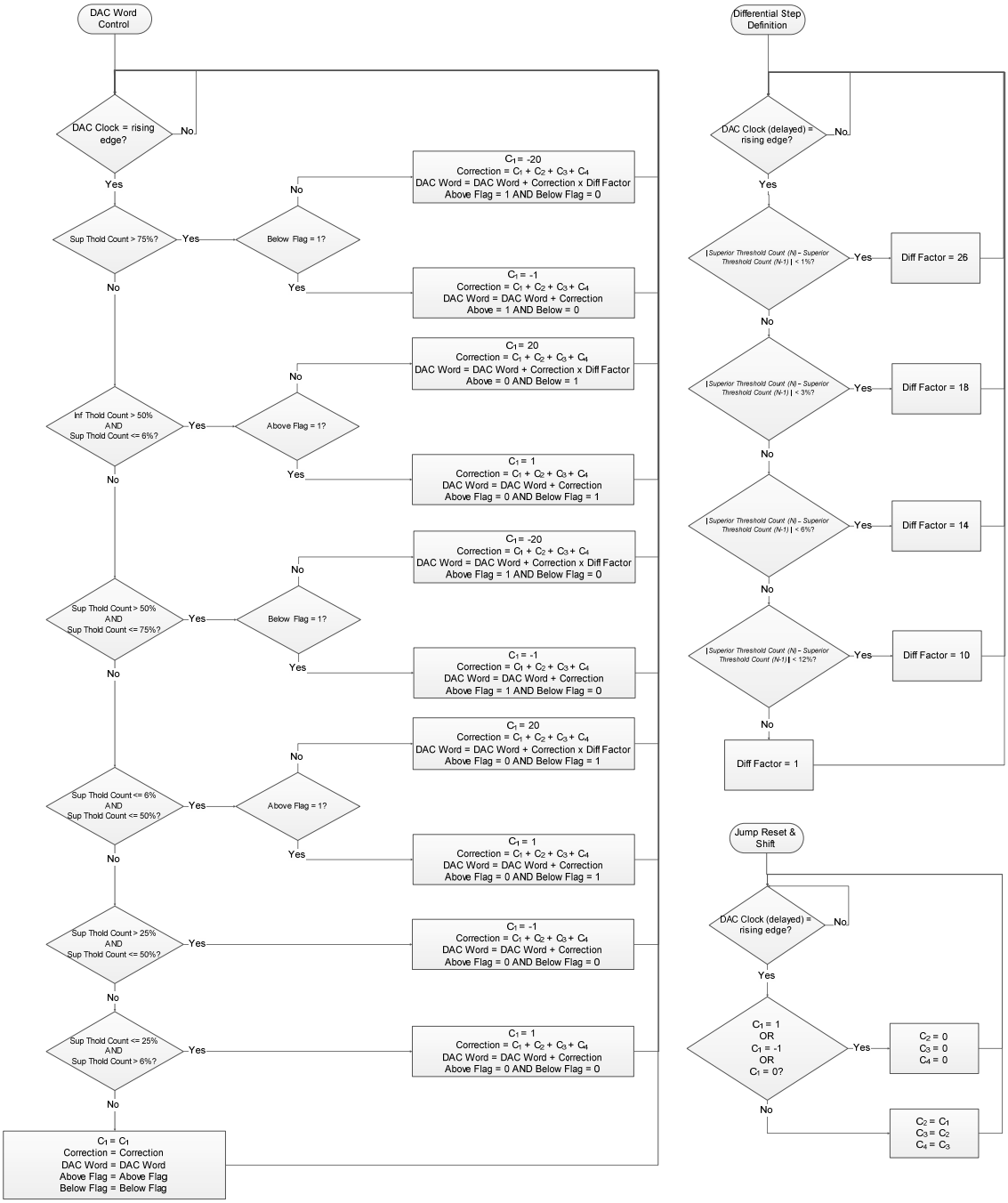


Figura 4.18 – Fluxograma do Algoritmo de controlo da iluminação (escrita na DAC).

4.4.3 Algoritmo completo do bloco Ilumination Control

Todo o restante bloco construído em VHDL encontra-se ilustrado no fluxograma da Fig. 4.19. Este contém todo o restante algoritmo pelo qual o bloco *illumination control* é baseado.

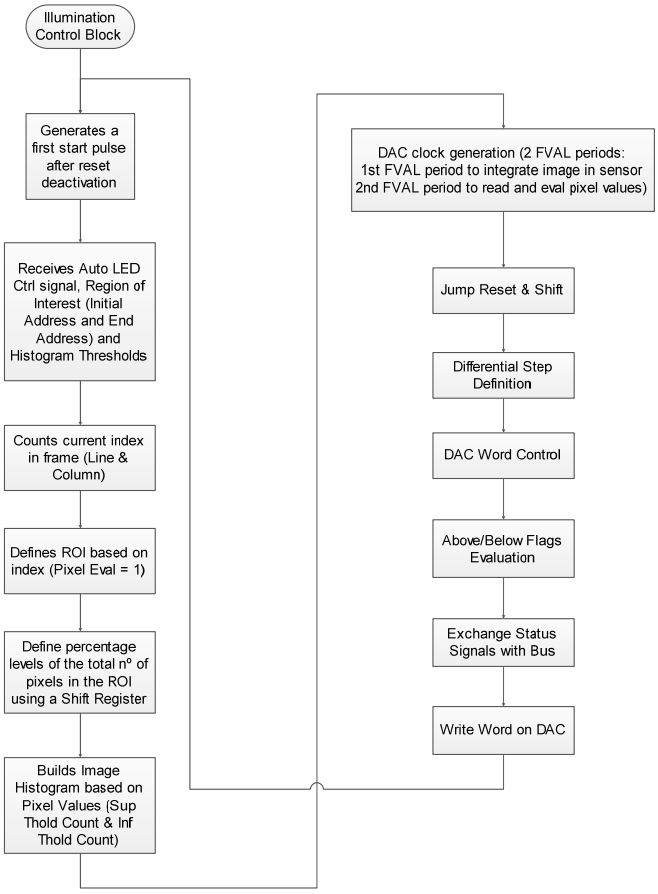


Figura 4.19 – Fluxograma do módulo de controlo *Illumination Control*.

Em primeiro lugar, é gerado um impulso de início, caso o bloco tenha sofrido um *reset*. De seguida o sistema lê do barramento os sinais de controlo automático da iluminação em dois registos. Um contém os endereços referentes à ROI e o outro os níveis de referência superior e inferior. É então definida a ROI com base na contagem dos índices de linha e coluna da *frame* e gerados os níveis de percentagem utilizando SRs (*Shift Registers*) para evitar o uso de divisores. Utilizando os níveis de referência é construído um histograma da imagem acumulando a quantidade de pixéis acima e abaixo do *Superior Threshold* e do *Inferior Threshold*, respetivamente. São gerados os sinais de relógio para a escrita na DAC utilizados no fluxograma descrito na Fig. 4.18. Após a troca de sinais de controlo com o barramento de dados, é escrita na DAC a palavra de controlo gerada pelo algoritmo descrito na secção anterior.

4.5 Implementação na FPGA

Na prática o módulo *Illumination Control*, implementado como *firmware* na FPGA, segue a estrutura representada na Fig. E.1 do Anexo E. Este é constituído por 18 sub-blocos representando os processos criados em VHDL sendo estes denominados por: *start_cnt_eval*, *auto_led_ctrl_regs*, *col_cnt_proc*, *line_cnt_proc*, *roi_eval*, *percent_sreg*, *sup_thold_eval*, *inf_thold_eval*, *dac_clk_gen*, *jump_sr*, *above_flag_eval*, *below_flag_eval*, *dac_word_ctrl*, *diff_factor_proc*, *dly_above_below_flags*, *wb_status_eval*, *dat_o_eval* e *dac_write_proc*. Os esquemas elétricos destes sub-blocos podem ser consultados no Anexo E, bem como uma descrição geral sobre a função e o funcionamento de cada um destes.

4.6 Resultados obtidos

Para verificar e comprovar a validade da implementação acima descrita, foram realizados testes ao sistema de controlo na configuração ilustrada na Fig. 4.20. Nesta encontra-se representada a plataforma FPGA Spartan 3E com interface USB2, um cabo de transmissão de dados e alimentação, um sensor de imagem NanEye com a fonte iluminação LED acoplada, e um computador pessoal com a interface gráfica.

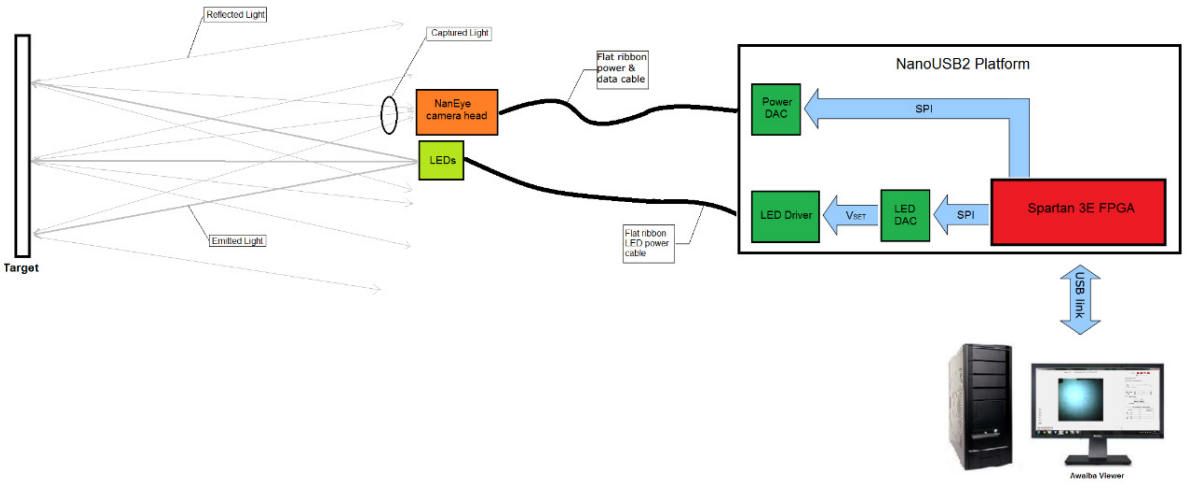


Figura 4.20 – Arquitetura do sistema de teste.

4.6.1 Região de interesse (ROI)

Utilizando um osciloscópio digital testou-se numa primeira fase a definição da região de interesse segundo os requisitos anteriormente descritos. Para tal, utilizou-se uma versão especial do *Awaiba Viewer* que permite ao utilizador acesso direto aos registos de controlo através da secção designada por *Expert Registers* (ver Fig. 4.21). Nesta é possível escrever no barramento de dados da

FPGA os endereços da ROI bem como definir os *thresholds* do histograma de imagem.

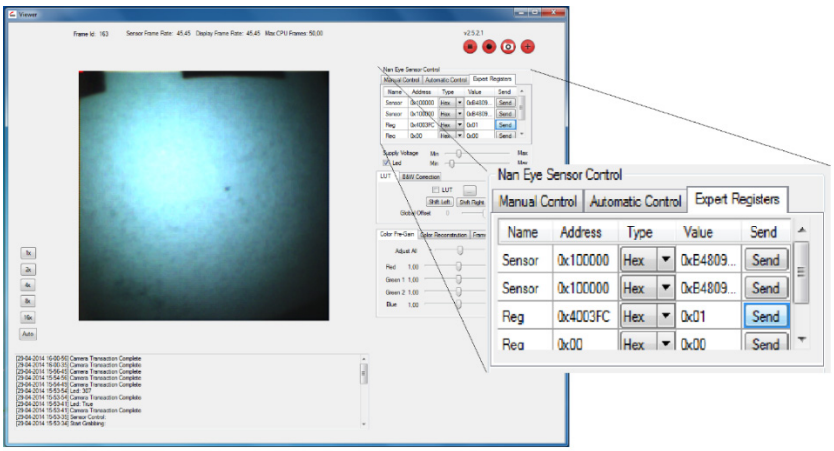
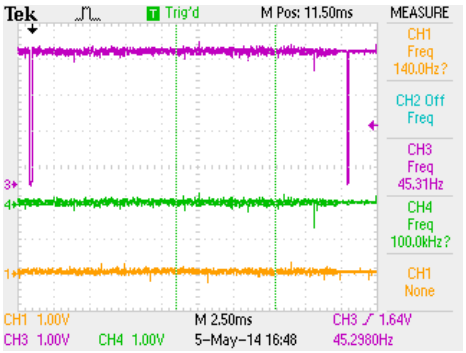
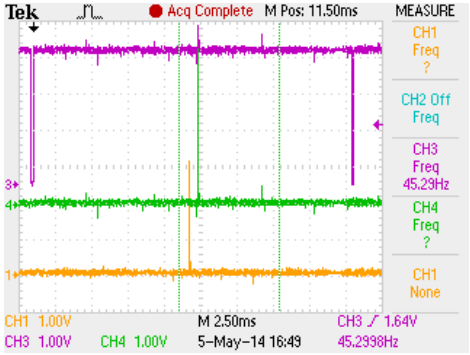


Figura 4.21 – Interface gráfica Awaiba Viewer com *Expert Registers*.

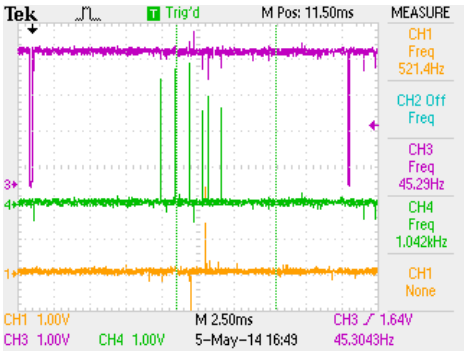
Deste modo definiram-se regiões de interesse de diferentes dimensões. Para verificar o funcionamento da definição da ROI colocaram-se nos pinos IO da plataforma de desenvolvimento os sinais *FVAL*, *Pixel_Eval_Ok* e *Pixel_Above_Supthold*. No osciloscópio estes são representados pelas formas de onda a roxo, verde e amarelo, respetivamente. Como foi referido, o sinal *FVAL*, representa o período de uma *frame*, e o período de tempo que o sinal *Pixel_Eval_Ok* se encontra ativo representa a área definida pelo utilizador como sendo a ROI. O sinal *Pixel_Above_Supthold* é ativado quando um píxel dentro da ROI tem um valor que ultrapassa o nível de referência definido pelo *Superior Threshold*. Este não tem qualquer aplicação prática no bloco e serve apenas para ilustrar o conceito. As Fig. 4.22 a) a f) demonstram o funcionamento do mecanismo de definição das ROIs com base nos endereços de linha e coluna. Os testes foram realizados iluminando diretamente o sensor com uma fonte de luz externa (neste caso uma lanterna) verificando a variação do sinal *Pixel_Above_Supthold*.



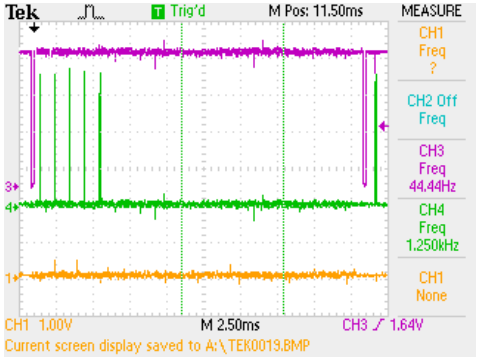
a) **ROI : Line = [0;0] Column = [0;0]**



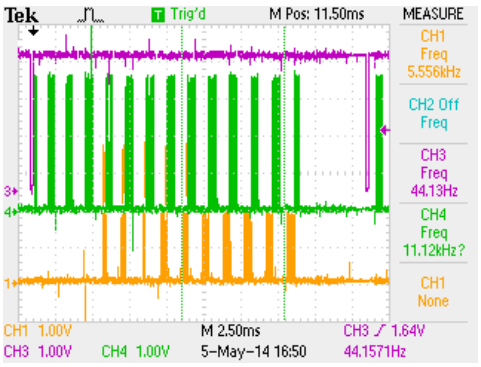
b) **ROI : Line = [120;130] Column = [120;130]**



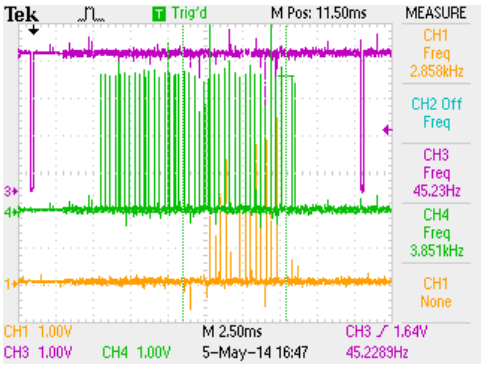
c) ROI : Line = [100;150] Column = [100;150]



d) ROI : Line = [0;50] Column = [0;50]



e) ROI : Line = [0;250] Column = [0;200]



f) ROI : Line = [50;200] Column = [50;200]

Figura 4.22 – Definição da ROI da imagem com base nos endereços de linha e coluna.

Na situação da Fig. 4.22 a) definiu-se propositadamente uma região nula. Tal como seria de esperar, mesmo iluminando diretamente o sensor com uma fonte de luz externa não se verificou qualquer reação. Na situação b) foi definida uma ROI com uma área muito restrita (10x10 pixéis). Esta encontrava-se situada no centro da imagem, nos endereços 120 a 130 para linha e coluna, resultando num pequeno período de avaliação representado pelo sinal *Pixel_Eval_Ok*. O mesmo se verificou para as restantes situações de teste, c) a f), embora com regiões de diferentes dimensões e localizações distintas. Por exemplo, na situação c) a ROI tem uma área de 50x50 pixéis localizada no centro da imagem. Por outro lado, em d) a mesma região com a mesma área já se encontra definida no canto superior esquerdo da *frame* (endereços 0 a 50). Em e) a região já não é simétrica, sendo excluídos as 50 últimas colunas de pixéis na metade direita da imagem. Por fim, em f) encontra-se ilustrada a ROI padrão que excluí 50 pixéis em toda a volta da imagem para minimizar o efeito introduzido pela lente.

4.6.2 Ajuste do nível de iluminação

Testou-se o sistema de controlo automático de iluminação em laboratório. Nas Fig. 4.23 a 4.24 são ilustrados dois destes testes. Estes foram realizados utilizando a ROI padrão e mantendo os *thresholds* fixos nos seguintes valores:

Superior Threshold: 600; *Inferior Threshold*: 300; *Initial Line Address*: 50; *End Line Address*: 200; *Initial Column Address*: 50; *End Column Address*: 200;

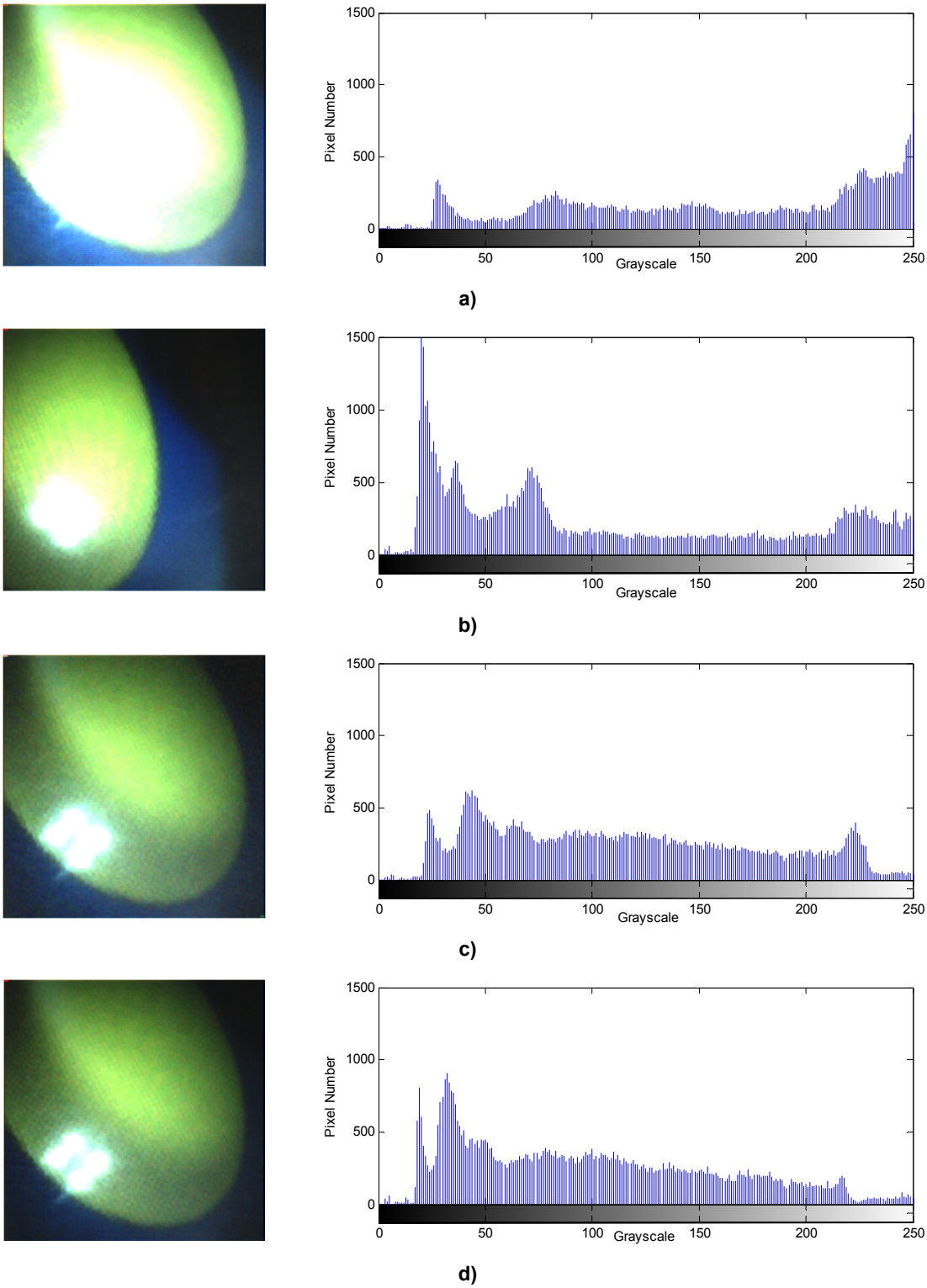


Figura 4.23 – Ajuste da iluminação para o nível ótimo (1).

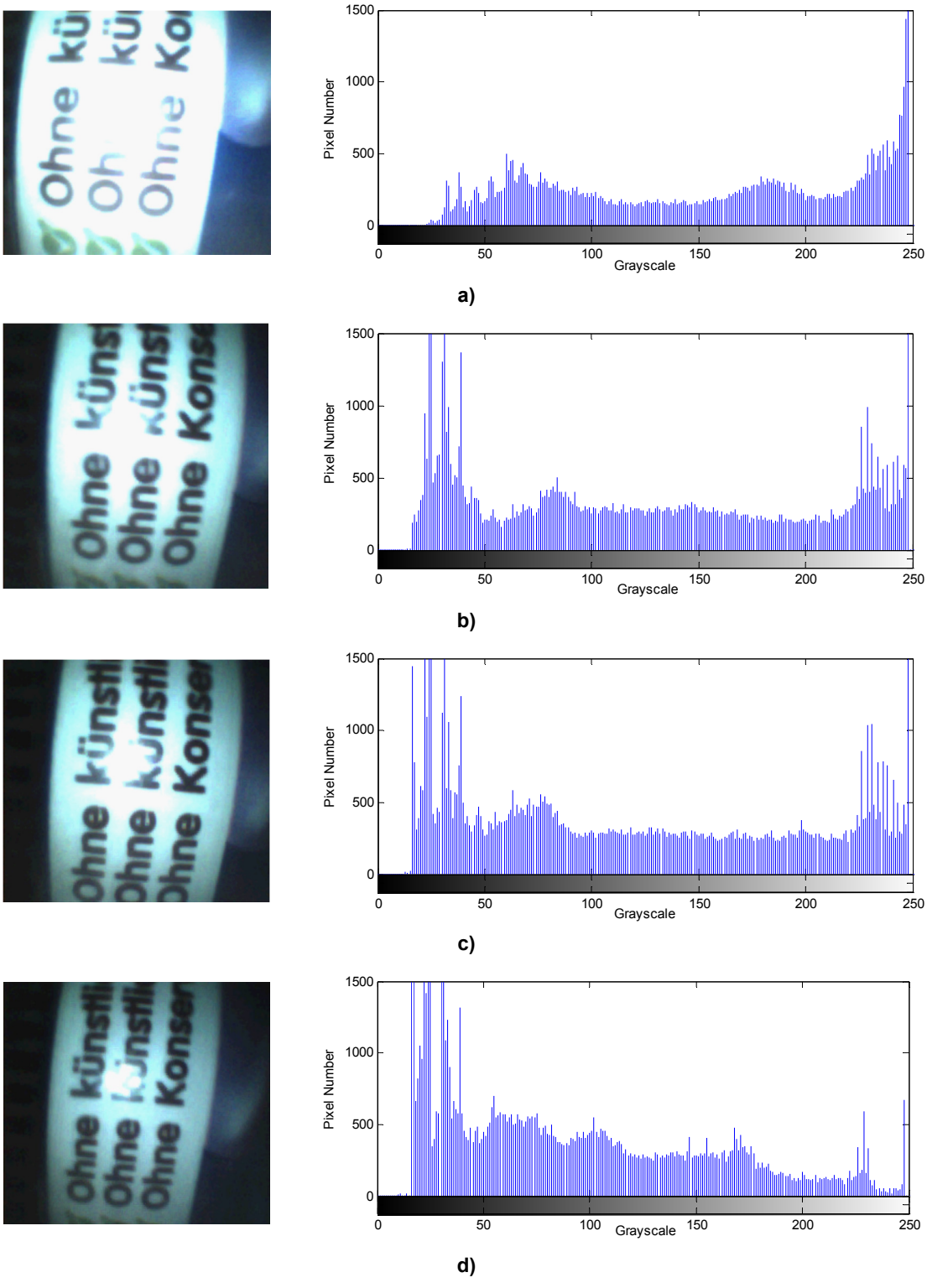


Figura 4.24 – Ajuste da iluminação para o nível ótimo (2).

Para cada *snapshot* gerou-se um histograma representando o número de pixéis em função da escala dos cinzentos. É possível observar que para imagens sobre iluminadas (como por exemplo nas Fig. 4.23 a) e 4.24 a)) existe uma grande concentração de pixéis na extremidade direita do histograma,

correspondendo aos níveis mais claros da *grayscale*. Nesta situação a percentagem de pixéis acima do *Superior Threshold* é superior a 25%, logo verifica-se um ajuste gradual da iluminação no sentido descendente. À medida que este ajuste é realizado, esta concentração diminui até que a percentagem de pixéis claros seja aproximadamente 25%, algo que se verifica nas Fig. 4.23 d) e 4.24 d). Naturalmente, a concentração de pixéis na extremidade esquerda do histograma aumenta à medida que a imagem se torna mais escura, contudo desde que o nível de pixéis claros se mantenha em torno dos 25% o algoritmo considera que a cena encontra-se corretamente iluminada.

Verificou-se que o sistema ajustou o nível de iluminação independentemente da multiplicidade dos tons de cor da cena filmada não sendo necessárias quaisquer alterações ao algoritmo para lidar com *targets* multicolor. O mesmo se verificou na presença de superfícies refletoras e não refletoras. Nesta última situação, apenas se verifica uma maior quantidade de luz refletida de volta para a câmara. Como consequência os LEDs necessitaram emitir menos intensidade luminosa para que se obtivesse o nível de iluminação ótimo.

Em termos de estabilidade e tempo de resposta observou-se um comportamento semelhante entre superfícies a preto e branco e multicolor e também entre superfícies refletoras e não refletoras. Isto explica-se pelo facto que os dados dos pixéis lidos pelo bloco *illumination control* apresentam valores, em DN, numa única escala de 0 a 1023, independentemente da cor do pixel (*Red*, *Green* ou *Blue*). Assim, da perspetiva do algoritmo, não existe distinção entre um vermelho, verde ou azul, apenas o nível de brilho que cada píxel individual captou é usado para construir o histograma sobre o qual o algoritmo atua. Adicionalmente foi conduzido mais um teste de ajuste de iluminação podendo este ser consultado no Anexo F.

4.6.2.1 Influência da região de interesse

Realizaram-se também testes tendo em conta diferentes dimensões da ROI de forma a verificar a influência desta no comportamento do módulo de controlo. Os testes foram realizados mantendo os *thresholds* fixos em: *Superior Threshold*: 600 e *Inferior Threshold*: 300.

Ao longo do desenvolvimento do algoritmo de controlo verificou-se que regiões mais pequenas (entre 100 pixéis e 1024 pixéis) implicavam uma maior sensibilidade devido ao menor número de pixéis sob análise. Isto significava que um pequeno ajuste no valor escrito na DAC, tinha um maior reflexo no erro. Contudo, como no algoritmo o erro é sempre referenciado ao tamanho da ROI, esta maior sensibilidade não se refletiu numa instabilidade percetível ao utilizador da aplicação.

Em regiões maiores (entre 30625 pixéis a 62500 pixéis), verificou-se que existiam gradientes elevados nos níveis de iluminação entre diferentes pontos da imagem. Algo natural tendo em conta que a ROI, nestes casos, cobre uma superfície elevada. No entanto não se verificaram diferenças no comportamento do ajuste dada esta situação. Por outro lado, se a ROI cobrisse os extremos da imagem do sensor, verificava-se um aumento do erro estático (numa versão preliminar do algoritmo este erro chegou aos 12%). Este era causado pela afetação da imagem pelo *vignetting* provocado pela lente. Em alguns casos isto provocava imagens com muito alto contraste o que resultava em histogramas com a forma ilustrada na Fig. 4.25.

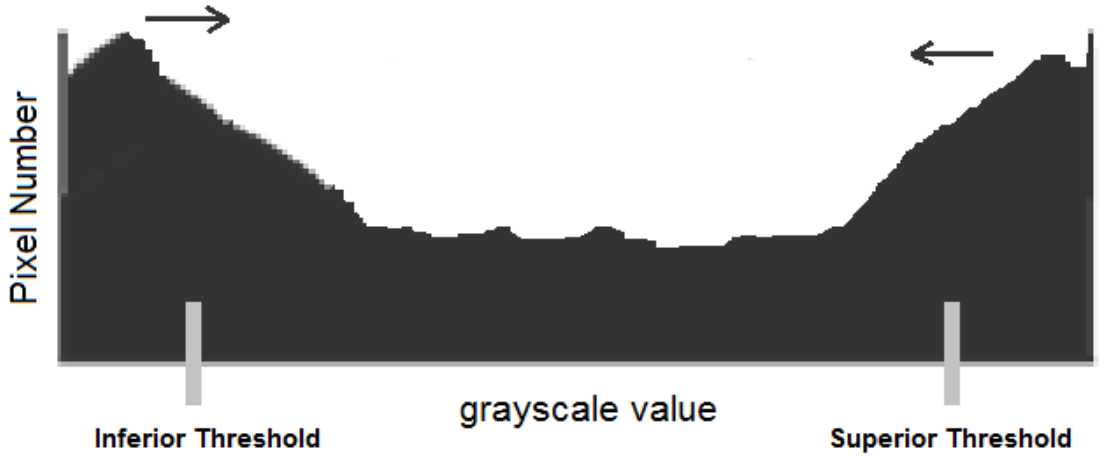


Figura 4.25 – Histograma de muito alto contraste.

Nestas condições, qualquer correção no sentido ascendente (aumento da iluminação), resultaria que o limite de pixéis acima do *Superior Threshold* seria ultrapassado logo o sistema de controlo faria uma correção no sentido descendente (diminuição da iluminação), o que resultaria que o limite de pixéis abaixo do *Inferior Threshold* também seria ultrapassado, levando o sistema a fazer nova correção no sentido ascendente. Como consequência gerava-se uma oscilação no nível de iluminação, levando a uma experiência de visualização desagradável para o utilizador. Para resolver esta situação foi necessário definir nas condições da estrutura do algoritmo que a contagem *Sup_Thold_Count* tem sempre prioridade sobre *Inf_Thold_Count* algo que já se encontra ilustrado no fluxograma da Fig. 4.18. Desta forma garante-se que o algoritmo de controlo apenas tem como referência o *Superior Threshold*, sendo a contagem inferior um fator auxiliar.

Noutras circunstâncias de teste não se verificaram diferenças assinaláveis no comportamento do controlo de iluminação, nem mesmo quando foi definida uma ROI de pequenas dimensões numa área da imagem afetada pelo *vignetting*.

4.6.2.2 Influência dos *thresholds*

Além dos testes acima referidos, foi também verificado o comportamento do sistema automático de controlo de iluminação para diferentes valores do *Superior* e *Inferior Threshold*, isto com o objetivo de demonstrar a influência dos *thresholds* no comportamento da iluminação. A ROI foi definida como: *Initial Line Address*: 50; *End Line Address*: 200; *Initial Column Address*: 50; e *End Column Address*: 200.

Verificou-se que é possível controlar o nível médio de iluminação aumentando e diminuindo o *Superior Threshold*. Ao aumentar este último de 600 para 900 e mantendo o *Inferior Threshold* fixo em 300, o nível médio de iluminação da imagem aumentava (traduzindo-se num aumento de corrente nos LEDs entre 5 mA a 10 mA dependendo da cena filmada e da ROI considerada). E ao diminuir, o nível médio de iluminação diminuía na mesma proporção.

Por outro lado, fixando o *Superior Threshold* e aumentando o *Inferior Threshold* verificou-se que a velocidade de ajuste da iluminação no sentido ascendente aumentava tal como pretendido. Da mesma forma, diminuindo o *Inferior Threshold* verificou-se que a velocidade de ajuste neste sentido diminuía. Este ganho de velocidade é necessário para que o algoritmo, na presença de erros elevados (> 50%), consiga corrigir com a mesma velocidade que no sentido descendente.

Colocando estes níveis de referência muito próximos um do outro, como por exemplo fixando o *Superior Threshold* em 550 e o *Inferior Threshold* em 450, observou-se que o algoritmo continua a funcionar adequadamente. Apesar de à partida estas condições gerarem uma situação semelhante ao problema ilustrado na Fig. 4.25, a solução apresentada permite ainda assim um funcionamento adequado.

4.6.3 Fases de desenvolvimento

Ao longo do desenvolvimento do algoritmo registou-se o comportamento do mesmo ao longo das várias fases de desenvolvimento. Como complemento aos resultados já apresentados, achou-se interessante descrever de uma forma resumida estes comportamentos referindo algumas vantagens e desvantagens observadas em algumas fases:

- **1ª Fase**: O controlo era simplesmente feito realizando ajustes em passos unitários no valor escrito na DAC com base no erro.
  - **Vantagens**: Erro de ajuste mínimo; Muito estável; Simplicidade.
  - **Desvantagens**: Resposta muito lenta;

- **2ª Fase**: Foram adicionados múltiplos níveis nos passos de correção. Quanto mais elevado o erro maior o passo de correção.
  - **Vantagens**: Ajuste mais rápido alcançando uma velocidade de ajuste razoável; Simplicidade de implementação.
  - **Desvantagens**: Erro de ajuste razoável mas tornou-se instável em torno do valor ótimo devido à correção puramente proporcional. Devido a este facto a velocidade de ajuste necessitou ser limitada (passos mais pequenos) para minimizar o erro. Nota: Nem mesmo aumentando o número de níveis de percentagem ou implementando uma histerese se conseguiu alcançar um comportamento aceitável.

- **3ª Fase**: Implementaram-se os sinais de estado *Above Flag* e *Below Flag* para tentar minimizar o erro estático em torno do valor ótimo. Estes indicam se os passos de correção anteriores forma feitos no sentido descendente ou ascendente acionando um mecanismo que reduz a amplitude do passo reduzindo o erro.
  - **Vantagens**: Velocidade de ajuste aumentou ligeiramente comparativamente à versão anterior.
  - **Desvantagens**: Erro estático continuou elevado traduzindo-se numa experiência visual não aceitável para o utilizador;

- **4ª Fase**: Incorporação de uma componente integrativa ao algoritmo utilizando um somatório de variáveis.
  - **Vantagens**: Erro estático mínimo; maior estabilidade (em relação à versão 2 e 3).
  - **Desvantagens**: Velocidade de ajuste continuou a não ser suficientemente elevada. Um dos requisitos deste sistema era um tempo máximo de ajuste até 1 s ou menos.

- **5ª Fase**: Com base no erro entre percentagens, adicionou-se uma componente diferencial ao algoritmo na forma de um fator multiplicado ao valor do somatório de variáveis.
  - **Vantagens**: Melhor compromisso encontrado entre velocidade de ajuste, erro estático e estabilidade. O tempo de resposta do sistema era sempre inferior a 1 s ou menos alcançando e mantendo o valor ótimo de iluminação sem oscilações percetíveis ao utilizador, proporcionando uma experiência de visualização agradável.
  - **Desvantagens**: Maior quantidade de lógica utilizada comparativamente às versões anteriores.

Para simplificar a comparação entre as diferentes fases de desenvolvimento do algoritmo, compilou-se na Tabela 4.1 os parâmetros mais significativos.

Tabela 4.1 – Comparação entre as diferentes fases de desenvolvimento do algoritmo de controlo de iluminação.

	1ª Fase	2ª Fase	3ª Fase	4ª Fase	5ª Fase
Erro	< 1%	< 12%	< 12%	< 3%	<3%
Velocidade de ajuste	> 10 s	2,5 s a 3,5 s	2 s a 3 s	4 s a 5 s	0,5 s a 1 s
Estabilidade	+++	--	-	++	+[11]
Lógica associada	--	-	+	++	+++

Note-se que os resultados apresentados na tabela anterior foram obtidos considerando uma ROI padrão: *Initial Line Address*: 50; *End Line Address*: 200; *Initial Column Address*: 50; e *End Column Address*: 200. Os *thresholds* foram fixos em: *Superior Threshold*: 300; e *Inferior Threshold*: 600.

4.6.4 Recursos utilizados

O módulo de controlo foi implementado numa Spartan 3 FPGA XC3S500E da Xilinx. Os dados de utilização do chip encontram-se apresentados na Tabela 4.2. Note-se que estes dados referem-se apenas ao módulo *Illumination Control* e foram obtidos instanciando este módulo isoladamente sem conexão ao restante *firmware* existente (do qual já fazia parte o bloco *Frequency Comparator* apresentado no Cap. 3). Note-se que devido a isto o número de portas de entrada e saída (*bonded IOBs*) excede o número máximo disponível para este dispositivo. Contudo há que ter em conta que, na realidade, este bloco não tem qualquer contacto direto com as portas IO da FPGA mas sim à lógica interna do restante *firmware*.

[11] Devido aos passos de correção elevados, causados pela componente diferencial do algoritmo, existe o maior risco de sobrelevações ocasionais em relação ao valor ótimo, como tal isto traduz-se numa ligeira redução da estabilidade.

Tabela 4.2 – Recursos utilizados.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	373	9,312	4%	
Number of 4 input LUTs	787	9,312	8%	
Number of occupied Slices	538	4,656	11%	
Number of Slices containing only related logic	538	538	100%	
Number of Slices containing unrelated logic	0	538	0%	
Total Number of 4 input LUTs	816	9,312	8%	
Number used as logic	787			
Number used as a route-thru	29			
Number of bonded IOBs	126	92	136%	
Number of BUFGMUXs	3	24	12%	
Number of MULT18X18SIOs	2	20	10%	
Average Fanout of Non-Clock Nets	3.20			

Seria de interesse comparar a utilização de recursos com outras arquiteturas de controlo. Contudo, apesar de existirem na literatura controladores baseados em FPGA, como por exemplo em [78]–[82], não foi possível retirar resultados conclusivos visto não ser inteiramente correto comparar a arquitetura apresentada neste trabalho com outras implementações cujos propósitos são distintos. Assim, não será possível retirar conclusões definitivas, neste aspeto, até realizar o mesmo sistema de controlo mas tendo por base uma arquitetura que implemente um controlador PID discreto.

4.7. Conclusões

Neste capítulo foi apresentada a topologia e explicado o funcionamento do sistema de controlo da intensidade de iluminação emitida por fontes LED. O controlo da iluminação foi realizado procedendo à análise dos pixéis restringidos por regiões de interesse em cada imagem captada pelo sensor. Desta análise resulta um histograma de imagem representando o nível de iluminação captado pela câmara. Definindo limites de referência, designados por *Superior* e *Inferior Thresholds*, é possível determinar se a *frame* captada pela câmara se encontra sobre ou sub exposta à iluminação contando o número de pixéis acima e abaixo do *Superior* ou *Inferior Threshold*, respetivamente. Comparando estas contagens com o número total de pixéis numa região de interesse, foi possível controlar o nível de iluminação ajustando a intensidade de corrente elétrica a percorrer os LEDs para que o nível de pixéis acima do *Superior Threshold* se mantivesse em torno dos 25%.

Este projeto demonstrou a capacidade do módulo de controlo para definir regiões de interesse na imagem captada pela câmara. Isto permite otimizar regiões restritas na imagem, algo especialmente útil em aplicações de endoscopia. O algoritmo de controlo demonstrou ser capaz de modular a intensidade de corrente a percorrer as fontes de iluminação de forma a

proporcionar uma experiência de visualização agradável para o utilizador. Foram alcançadas velocidades de resposta inferiores a 1 s com erros até 3%.

Verificou-se ser necessário integrar, no algoritmo de controlo, uma componente integrativa (na forma de um somatório) para reduzir o erro estático; e uma componente diferencial (através de níveis de percentagem) para acelerar a velocidade de convergência para o valor de iluminação ótimo.

O bloco *Illumination Control* foi integrado com sucesso com toda a arquitetura já existente na plataforma NanoUSB2. Nisto inclui-se a seleção e tratamento dos sinais de interesse bem como a receção e interpretação de comandos de controlo recebidos a partir da interface gráfica *Awaiba Viewer*.

5. Conclusão e trabalhos futuros

Nesta secção são apresentadas as conclusões gerais sobre o trabalho realizado no âmbito desta tese de mestrado. Em particular são tiradas ilações sobre os resultados obtidos no módulo de controlo de frequência e sincronização de câmaras bem como no núcleo de controlo e ajuste dinâmico da iluminação. Estes são relacionados com os objetivos inicialmente propostos e com possíveis trabalhos futuros.

5.1. Conclusões gerais

Nesta tese foi demonstrada uma nova técnica para sincronizar perfeitamente até 8 câmaras auto-temporizadas individuais com base na família de sensores de imagem NanEye fabricados em tecnologia CMOS pela Awaiba, e em plataformas FPGA com interface USB2/USB3. Os módulos de câmara de tamanho mínimo (1 mm x 1 mm ou mais pequenos) normalmente não permitem sincronização externa. Contudo, para visão estéreo ou reconstrução 3D com múltiplas câmaras, bem como para aplicações que exijam iluminação pulsada, é necessário realizar a sincronização de múltiplas câmaras. Neste trabalho, o desafio de sincronizar múltiplas câmaras auto-temporizadas com uma interface de apenas 4 fios foi resolvido ao regular adaptativamente a alimentação de cada câmara com o objetivo de sincronizar a frequência e a fase das suas *frames*. Para esse efeito, um módulo de controlo foi criado para monitorizar constantemente a frequência de operação de cada câmara ao medir o período de linha em cada *frame* com base num sinal de amostragem bem definido. A frequência é ajustada ao variar o nível de tensão aplicado no sensor com base no erro entre o período de linha medido e o período de linha desejado. Para assegurar a sincronização em fase entre *frames* de múltiplas câmaras, foi implementada uma interface *Master-Slave*. Uma única câmara foi definida como a entidade *Master*, com a sua frequência de operação a ser controlada diretamente através de uma interface gráfica baseada num PC. As restantes câmaras são configuradas em modo *Slave* e são conectadas diretamente com o módulo de controlo da câmara *Master*. Isto permite às restantes câmaras monitorizar o período de linha e *frame* deste último e ajustar o seu próprio para alcançar a sincronização em fase e em frequência.

O módulo de controlo, denominado *Frequency Comparator*, foi testado em ambiente laboratorial com até 4 câmaras. Os resultados indicam que este é capaz de manter uma sincronização com erro reduzido em temperaturas ambientes variáveis. Os testes demonstram que, mesmo na presença de diferenças de temperatura de 60 °C entre câmaras, a sincronização de fase e

frequência é ainda assim mantida. A gama de temperatura (testado para -2 ºC a 55 ºC) é apenas limitada pelos limites autoimpostos da tensão de alimentação da câmara. Para mais o erro de fase médio entre frames foi determinado como sendo de 3,77 µs. Tendo em conta que o frame rate nominal de uma câmara NanEye é de 44 fps, isto significa que o erro médio é de aproximadamente 0,017 %. Este módulo de controlo também demonstrou ser independente do comprimento do cabo conector bem como da versão da câmara NanEye.

Foi também demonstrado um novo algoritmo de controlo da intensidade de iluminação emitida por uma fonte de luz LED com base no sensor de imagem NanEye e numa plataforma FPGA com interface USB2. Os equipamentos de endoscopia são utilizados como meio para obter imagens do interior do corpo humano, facilitando o diagnóstico e o tratamento de uma série de doenças. Este tipo de equipamento incorpora sensores de imagem de pequeno tamanho aos quais são acopladas fontes de iluminação. Para que se possa obter uma boa experiência de visualização a intensidade da fonte de iluminação tem que ser ajustada dinamicamente em função das condições de iluminação ambiente. Neste trabalho, o problema de controlar a intensidade da fonte de iluminação sem recorrer a um luxímetro externo foi resolvido analisando as imagens captadas a partir da câmara e atuando dinamicamente no nível de corrente fornecido à fonte de luz. Para esse efeito, foi criado um módulo de controlo para selecionar uma região de interesse em cada *frame* captada e analisar o nível de cinzento de cada píxel contido na região. Isto permite gerar um histograma de cada imagem captada pela câmara que é analisado tendo em conta dois níveis de referência. São contabilizados o número de pixéis acima do nível de referência superior e o número de pixéis abaixo do nível de referência inferior. O algoritmo atua na alimentação da fonte de luz para que o número de pixéis acima do nível superior corresponda a 25% do número total de pixéis na região de interesse. Isto permite manter um nível de iluminação adequado independentemente do tamanho da região de interesse e do tipo de superfície iluminada.

O módulo de controlo, *Illumination Control*, foi também testado em ambiente laboratorial com um *array* de 4 LEDs acoplados a uma câmara. Os resultados mostraram que o algoritmo de controlo é capaz de manter um nível de iluminação adequado para diferentes condições de teste, incluindo diferentes tipos de *targets*, e iluminação ambiente distinta. Foi demonstrado que o módulo de controlo permite a otimização da iluminação em regiões de interesse cuja dimensão pode totalizar a área total de captação de luz do sensor (250 x 250 pixéis) ou em pequenas regiões como por exemplo 10 x 10 pixéis. A versão final

do algoritmo de controlo demonstrou ser capaz de alcançar velocidades de ajuste inferiores a 1 s com erros menores que 3%.

5.2. Trabalhos futuros

O resultado deste trabalho permitirá a realização de sistemas de visão estéreo 3D com dimensões menores que 3 mm bem como equipamento de visão no contexto da endoscopia médica, tais como em robôs cirúrgicos ou cirurgia micro invasiva. Nomeadamente o desenvolvimento em *hardware* de um sistema de processamento e visualização 3D com base numa plataforma FPGA. Será também possível a implementação de sistemas automáticos de controlo de iluminação em equipamentos de visão aplicados à indústria médica tais como endoscópios, braços robóticos, sondas, entre outros.

Alternativamente, seria de interesse transferir parte do processamento de imagem (aplicação de filtros, correção de imagem, etc), realizado neste momento na aplicação gráfica, *Awaiba Viewer*, para a FPGA. Traduzindo as componentes computacionalmente mais intensivas para *hardware* dedicado permitiria reduzir o peso da aplicação no PC e ao mesmo tempo diminuir a latência na visualização do vídeo.

Referências

- [1] A. El Gamal and H. Eltoukhy, “CMOS Image Sensors,” *IEEE Circuits Devices Mag.*, no. June, 2005.
- [2] D. Wedge, D. Huynh, and P. Kovesi, “Video Sequence Synchronization,” University of Western Australia, 2007.
- [3] ECRI Institute, “Video Endoscopy Systems,” *Healthc. Prod. Comp. Syst.*, vol. 1, pp. 1–7, 2007.
- [4] A. Y. K. Chan, “Endoscopic Video Systems,” in *Biomedical Device Technology: Principles and Design*, pp. 536–546, 2008.
- [5] M. Liedlgruber and A. Uhl, “Computer-aided decision support systems for endoscopy in the gastrointestinal tract: a review.,” *IEEE Rev. Biomed. Eng.*, vol. 4, pp. 73–88, Jan. 2011.
- [6] Awaiba, “Medical image sensors – Awaiba – Your key to CMOS image sensors,” 2011. [Online]. Available: http://www.awaiba.com/v2/en/products/medical-image-sensors/.
- [7] B. Luo, Y. Fuxing, and L. Yan, “Key technologies and research development of CMOS image sensors,” *2010 Second IITA Int. Conf. Geosci. Remote Sens.*, pp. 322–325, Aug. 2010.
- [8] A. Palakodety, “CMOS Active Pixel Sensor for Digital Cameras: Current State-of-the-Art,” University of North Texas, 2007.
- [9] Matrox, “Camera Interface Guide,” Dorval, Quebec, 2003.
- [10] N. Flores-Guzmán, J. H. Sossa-Azuela, and R. Bizuet-García, “A Decision and Communication Management Methodology for embedded Multi- smart Camera systems, applied to real-time inspection in lamps production,” in *Second ACM/IEEE International Conference on Distributed Smart Cameras*, pp. 1–10, 2008.
- [11] NXP Semiconductors, “UM10204 I2C-bus specification and user manual,” 2014.
- [12] Micron, “MT9P031 Image Sensor,” 2006.
- [13] ST Microelectronics, “VGA CMOS COLOR IMAGE SENSOR,” 2003.
- [14] K. Cho, A. Krymski, and E. R. Fossum, “A 3-Pin 1.5 V 550 μW 176 x 144 Self-Clocked CMOS Active Pixel Image Sensor,” *ISLPED’01*, pp. 316–321, 2001.
- [15] M. Wäny and E. Reis, “NanEye_2D ASIC SPECIFICATION,” *Awaiba*, vol. 41, no. 2.00.6, pp. 1–24, 2014.
- [16] R. Kumar and V. Kursun, “Impact of temperature fluctuations on circuit characteristics in 180 nm and 65 nm CMOS technologies,” *2006 IEEE Int. Symp. Circuits Syst.*, no. V, p. 4, 2006.
- [17] S. Suman and B. P. Singh, “Ring Oscillator Based CMOS Temperature Sensor Design,” *Int. J. Sci. Technol. Res.*, vol. 1, no. 4, pp. 76–81, 2012.
- [18] J. Herrmann, “3D in Machine Vision – both Hype and Reality,” 2011.
- [19] W. Kaczurba, “FPGA-Based System Combines Two Video Streams to Provide 3D Video,” *Analog Dialogue*, no. 47–12, pp. 1–5, 2013.
- [20] RobotShop, “mesa-imaging-tof-camera-sr4500-69x55-fov-1-large,” 2014. [Online]. Available: http://www.robotshop.com/media/files/images/mesa-imaging-tof-camera-sr4500-69x55-fov-1-large.jpg. [Accessed: 28-Jul-2014].
- [21] H. P. Moravec, “Robot Spatial Perception by Stereoscopic Vision and 3D Evidence Grids,” Carnegie Mellon University, 1996.

- [22] G. Litos, X. Zabulis, and G. Triantafyllidis, “Synchronous Image Acquisition based on Network Synchronization,” in *Conference on Computer Vision and Pattern Recognition Workshop, 2006. CVPRW ’06.*, 2006.
- [23] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd Editio. Cambridge University Press, 2004.
- [24] D. Bradley, B. Atcheson, I. Ihrke, and W. Heidrich, “Synchronization and rolling shutter compensation for consumer video camera arrays,” in *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–8, 2009.
- [25] W. Nunziati, J. Alon, S. Sclaroff, and A. Del Bimbo, “View Registration Using Interesting Segments of Planar Trajectories,” in *Proceedings. IEEE Conference on Advanced Video and Signal Based Surveillance*, pp. 75–80, 2005.
- [26] L. Hou, S. Kagami, and K. Hashimoto, “Frame synchronization of high-speed vision sensors with respect to temporally encoded illumination in highly dynamic environments.,” *MDPI Sensors 2013*, vol. 13, no. 4, pp. 4102–4121, Jan. 2013.
- [27] P. Greisen, S. Heinzle, M. Gross, and A. P. Burg, “An FPGA-based processing pipeline for high-definition stereo video,” *EURASIP J. Image Video Process.*, vol. 2011, no. 1, p. 18, 2011.
- [28] T. Kanade, P. J. Narayanan, and P. W. Rander, “Virtualized Reality: Being Mobile in a Visual Scene,” in *International Conference on Artificial Reality and Tele-Existence/Conference on Virtual Reality Software and Technology*, pp. 273–285, 1995.
- [29] Silicon Labs, “AN377 - Timing and Synchronization in Broadcast Video,” 2009.
- [30] J. Xia, X. Zhao, and W. Guo, “Research on the method of multi-channel video acquisition and display based on FPGA,” *I.J. Comput. Netw. Inf. Secur.*, vol. 1, no. November, pp. 17–23, 2010.
- [31] M. Muehlebach, C. Huerzeler, M. Tresanchez, and J. Nikolic, “Camera Auto Exposure Control for VSLAM Applications,” Swiss Federal Institute of Technology Zurich, 2010.
- [32] C. Seger, “Colloquium on ‘Illumination for High Speed imaging,’” 2006.
- [33] C. Grana, D. Borghesani, P. Santinelli, and R. Cucchiara, “Veiling Luminance estimation on FPGA-based embedded smart camera,” in *2012 IEEE Intelligent Vehicles Symposium*, pp. 334–339, 2012.
- [34] C. Carrillo, E. Diaz-Dorado, J. Cidrás, A. Bouza-Pregal, P. Falcón, A. Fernández, and A. Álvarez-Sánchez, “Lighting control system based on digital camera for energy saving in shop windows,” *Elsevier Energy Build.*, vol. 59, pp. 143–151, Apr. 2012.
- [35] I. Andorko and S. M. Ieee, “An FPGA-based Stereoscopic Camera - Electronic Design Tools and Techniques,” pp. 43–48, 2010.
- [36] Q. K. Vuong, S. Yun, and S. Kim, “A New Auto Exposure System to Detect High Dynamic Range Conditions Using CMOS Technology,” in *Convergence and Hybrid Information Technologies*, no. March, M. Crisan, Ed. InTech, pp. 227–238, 2010.
- [37] D. Ilstrup and R. Manduchi, “One-Shot Optimal Exposure Control,” in *Proceedings of the 11th European conference on Computer vision: Part I*, pp. 200–213, 2010.
- [38] S. Schulz, M. Grimm, and R.-R. Grigat, “Using Brightness Histogram to perform Optimum Auto Exposure,” *WSEAS Trans. Syst. Control*, vol. 2, no. 2, pp. 93–100, 2007.
- [39] C.-T. Lin, K.-W. Fan, and W.-C. Cheng, “An Illumination Estimation Scheme for Color Constancy Based on Chromaticity Histogram and Neural Network,” in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2488–2494, 2005.

- [40] G. P. Ngoc, S.-H. Lee, Y. M. Yu, and K.-R. Kwon, “An Intelligent LED Illumination Control System Using Camera,” *J. Converg. Inf. Technol.*, vol. 8, no. 12, pp. 208–218, Jul. 2013.
- [41] L. Svilainis, “LED brightness control for video display application,” *Elsevier Displays*, vol. 29, no. 5, pp. 506–511, Dec. 2008.
- [42] A. Abu-aisheh, S. Khader, and O. Hasan, “Sustainable PV-powered FPGA-controlled high brightness LED illumination systems,” in *Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON)*, pp. 1–5, 2012.
- [43] F. Xiao, J. M. DiCarlo, P. B. Catrysse, and B. A. Wandell, “Image analysis using modulated light sources,” in *Proc. SPIE 4306, Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications II*, 2001.
- [44] W. Wang and W. Li, “Design of Reconfigurable LED Illumination Control System Based on FPGA,” in *2009 2nd International Congress on Image and Signal Processing*, pp. 1–4, 2009.
- [45] H. Cho and O.-K. Kwon, “A Backlight Dimming Algorithm for Low Power and High Image Quality LCD Applications,” *IEEE Trans. Consum. Electron.*, vol. 55, no. 2, pp. 839–844, 2009.
- [46] C. Liao, F. Lin, L. Liao, and Y. Huang, “Inverse of Mapping Function (IMF) Method for Image Quality Enhancement of Dual-Panel Displays,” in *SID Int’l Symposium Dig. Tech. Papers*, vol. 2, pp. 431–434, 2007.
- [47] F. Roth, “Using low cost FPGAs for realtime video processing,” Masaryk University, 2011.
- [48] F. N. S. Oliveira and P. L. R. Ferreira, “Câmara IP sem fios para Vídeo - Vigilância,” Universidade do Porto, 2006.
- [49] Xilinx, “Spartan-3E FPGA Family Data Sheet,” *Prod. Specif.*, no. DS312 v4.1, 2013.
- [50] Xilinx, “Spartan-6 Family Overview Summary of Spartan-6 FPGA Features,” *Prod. Specif.*, vol. 160, no. DS160 v2.0, pp. 1–11, 2011.
- [51] Xilinx, “ISE In-Depth Tutorial,” no. UG695 v13.1, pp. 1–148, 2011.
- [52] G. Stitt, “Are field-programmable gate arrays ready for the mainstream?,” *IEEE Micro*, vol. 31, no. 6, pp. 58–63, 2011.
- [53] Y. Zhao and I. E. G. Richardson, “Complexity management for video encoders,” *Proc. tenth ACM Int. Conf. Multimed. - Multimed. ’02*, p. 647, 2002.
- [54] L. Zhang and W. Dong, “Spatial-Temporal Color Video Reconstruction from Noisy CFA Sequence,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 6, pp. 1–18, 2010.
- [55] C. S. Hannangara, I. E. Richardson, and A. J. Miller, “Computational Complexity Management of a Real-Time H.264/AVC Encoder,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 9, pp. 1–11, 2007.
- [56] Awaiba, “NanEye Camera System,” no. 1.0.14 Web, pp. 1–10, 2012.
- [57] S. Voltz, “NanEye Worlds smallest digital camera NanEye Miniaturized Cameramodule,” *Awaiba*, pp. 0–11, 2012.
- [58] G. Jovanovi, M. Stojcev, and Z. Stamenkovic, “A CMOS Voltage Controlled Ring Oscillator with Improved Frequency Stability,” *Sci. Publ. State Univ. Novi Pazar*, vol. 2, no. ser. A: Apll. Math. Inform. and Mech., pp. 1–9, 2010.
- [59] P. M. Farahabadi, H. Miar-Naimi, and A. Ebrahimzadeh, “A New Solution to Analysis of CMOS Ring Oscillators,” *Iran. J. Electr. Electron. Eng.*, vol. 5, no. 1, pp. 32–41, 2009.
- [60] B. Razavi, *Design of Analog CMOS Integrated Circuits*. New York, New York, USA: McGraw Hill, pp. 482–532, 2001.

- [61] F. Kallel, M. Fakhfakh, M. Loulou, S. A. A. Oumansour, and M. H. Sbaa, “Modelling the frequency Sensitivity Kvco of a Ring Oscillator National Engineering School of Sfax,” in *ICECS 2008. 15th IEEE International Conference*, pp. 794–797, 2008.
- [62] R. Jacob Baker, *CMOS: Circuit Design, Layout, and Simulation*, 3rd ed. Wiley-IEEE, pp. 551–613, 2010.
- [63] A. Holme, “PLL Theory Q&A,” 2004. [Online]. Available: http://www.aholme.co.uk/Ethernet/PLLQA.htm.
- [64] S. P. Mohanty and E. Kougianos, “Incorporating Manufacturing Process Variation Awareness in Fast Design Optimization of Nanoscale CMOS VCOs,” *IEEE Trans. Semicond. Manuf.*, vol. 27, no. 1, pp. 22–31, 2014.
- [65] Awaiba, “NanoUSB2.1 Product Specification,” no. 1.0.0, pp. 1–8, 2013.
- [66] R. S. Krishna, “Designing With EZ-USB ® FX2LP ™ Slave FIFO Interface,” *Cypress*, no. 001, 2014.
- [67] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips, “Universal Serial Bus Specification - Revision 2.0,” no. 2.0, 2000.
- [68] Hewlett-Packard, Intel, Microsoft, Renesas, ST-Ericsson, and Texas Instruments, “Universal Serial Bus 3 . 1 Specification,” no. 1.0, 2013.
- [69] V. Bexiga and F. Piedade, “Evaluation Board Interfaces,” *Awaiba*, no. 0.0.1, pp. 1–36, 2013.
- [70] R. S. Krishna, “Designing with the EZ-USB ® FX3 ™ Slave FIFO Interface,” *Cypress*, no. 001, pp. 1–66, 2013.
- [71] Linear Technology Corporation, “LTC2630 - Single 12-/10-/8-Bit Rail-to-Rail DACs with Integrated Reference in SC70,” pp. 1–22, 2012.
- [72] J. A. R. Azevedo, “Linhas de Transmissão,” *Apontamentos Radiação e Propagação, Univ. da Madeira*, 2012.
- [73] Xilinx, “Spartan-3E Libraries Guide for HDL Designs,” vol. 617, pp. 1–489, 2011.
- [74] Linear Technology Corporation, “LT3092 - 200mA 2-Terminal Programmable Current Source,” pp. 1–22, 2009.
- [75] R. Nunn, “Rick Nunn - Histograms,” 2010. [Online]. Available: http://ricknunn.com/thoughts/histograms.
- [76] Y. Zheng, S. Lin, C. Kambhamettu, J. Yu, and S. B. Kang, “Single-Image Vignetting Correction,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 12, pp. 2243–2256, Dec. 2009.
- [77] I. T. Young, J. J. Gerbrands, and L. J. van Vliet, “Fundamentals of Image Processing,” Delft University of Technology, 1998.
- [78] E. Normann Naess and S. Hendseth, “Implementing Controller Strategies in FPGA,” Norwegian University of Science and Technology, 2009.
- [79] B. V Sreenivasappa and R. Y. Udaykumar, “Analysis and Implementation of Discrete Time PID Controllers using FPGA,” vol. 2, no. 1, pp. 71–82, 2010.
- [80] R. Nema, R. Thakur, and R. Gupta, “Design & Implementation of FPGA Based On PID Controller,” no. 2, pp. 14–16, 2013.
- [81] S. Sonoli, “Implementation of FPGA based PID Controller for DC Motor Speed Control System,” vol. II, 2010.
- [82] A. Trimeche, A. Sakly, A. Mitbaa, and M. Benrejeb, “PID Controller Using FPGA Technology,” in *Advances in PID Control*, V. D. Yurkevich, Ed. InTech, pp. 259–273, 2011.
- [83] E. Zianbetov, M. Javidan, F. Anceau, D. Galayko, E. Colinet, and J. Juillard, “Design and VHDL modeling of all-digital PLLs,” in *Proceedings of the 8th IEEE International NEWCAS Conference, NEWCAS2010*, pp. 293–296, 2010.

- [84] J. A. Tierno, A. V. Rylyakov, and D. J. Friedman, “A Wide Power Supply Range, Wide Tuning Range, All Static CMOS All Digital PLL in 65 nm SOI,” *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 42–51, 2008.
- [85] D. Li, P. Chuang, and M. Sachdev, “Comparative analysis and study of metastability on high-performance flip-flops,” in *Proceedings of the 11th International Symposium on Quality Electronic Design, ISQED 2010*, pp. 853–860, 2010.
- [86] V. Kratyuk, K. Mayaram, and U.-K. Moon, “Digital Phase-Locked Loops for multi-GHz Clock Generation,” Oregon State University, 2006.
- [87] A. Nemirovski, “Optimization II Numerical Methods for Nonlinear Continuous Optimization,” Technion – Israel Institute of Technology, 1999.
- [88] V. Chitturi and C. Hutchens, “Major Concerns in Clock Recovery of Manchester Encoded Data using a Phase Lock Loop,” Oklahoma State University, 2005.
- [89] Xilinx, “Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs,” *Appl. Note Spartan-3 Spartan-3L FPGA Fam.*, vol. 462, pp. 1–68, 2006.
- [90] M. Happe, H. Hangmann, A. Agne, and C. Plessl, “Eight ways to put your FPGA on fire — A systematic study of heat generators,” *2012 Int. Conf. Reconfigurable Comput. FPGAs*, pp. 1–6, Dec. 2012.

Anexo A - Controlo dinâmico de frequência baseado numa ADPLL

A.1 Introdução

Existem múltiplas formas de implementar uma PLL. Esta pode ser construída utilizando apenas circuitos analógicos, apenas circuitos digitais, ou ambos. A implementação do módulo de controlo de frequência numa plataforma FPGA exige que a aplicação seja totalmente digital sendo dispensável a utilização de componentes analógicos externos como condensadores e resistências. Desta forma, focou-se no desenvolvimento de uma PLL inteiramente digital (ADPLL).

Comparativamente às PLLs analógicas, as ADPPLs são menos sensíveis ao ruído uma vez que não existe um controlo analógico do VCO. Além disto podem ser implementadas exclusivamente com blocos digitais *standard*. Assim, o controlo de um oscilador baseado numa ADPPL apresenta-se como uma solução interessante [83].

A.2 Topologia

A topologia da ADPLL proposta para realizar a sincronização das múltiplas câmaras segue o trabalho desenvolvido em [83]. Optou-se por esta arquitetura visto que combina as vantagens de um detetor de fase-frequência *Bang-Bang* permitindo simultaneamente medir a amplitude do erro de fase através de uma porta XOR [62].

A topologia da ADPLL é apresentada na Fig. A.1. É um sistema realimentado cuja função é alinhar a fase e a frequência do sinal proveniente do sensor de imagem com a frequência de referência indicada pela entrada ref. É constituída por um detetor de fase-frequência (PFD), um filtro digital (LF) e um oscilador controlado digitalmente (DCO). A escolha desta topologia justifica-se pela sua arquitetura totalmente digital, excluindo o uso de fontes de corrente em configuração *Charge Pump* ou componentes analógicos como resistências e condensadores no filtro do *loop* [62]. Deste modo, permite-se a implementação numa plataforma FPGA de um sistema de controlo escalável, testável e previsível.

O PFD é utilizado para comparar fase/frequência do sinal de referência (que deverá ser variável) com o sinal de relógio proveniente do sensor de imagem, extraído a partir do *stream* de dados. Desta comparação resulta um código binário representativo do sinal e do módulo da diferença entre

frequências. A saída do PFD é filtrada pelo filtro digital para eliminar harmónicos de ordem superior e diminuir o nível de ruído. O sinal binário filtrado é utilizado para controlar o DCO, alterando o comportamento da câmara modulando a tensão aplicada diretamente através de uma DAC.

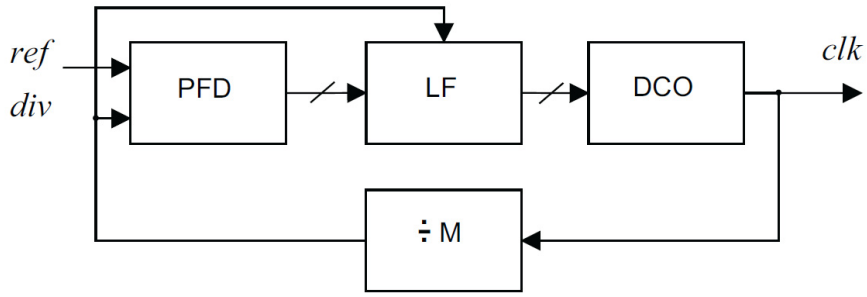


Figura A.1 – Diagrama de blocos da ADPLL [83].

A.2.1 PFD

A estrutura do PFD pode ser observada na Fig. A.2. Este é constituído por um detetor *bang-bang* de fase-frequência (BB-PFD), um conversor temporal para digital (TDC) e um bloco de aritmética. Este segue a arquitetura proposta em [83] sendo esta uma evolução do trabalho desenvolvido em [84].

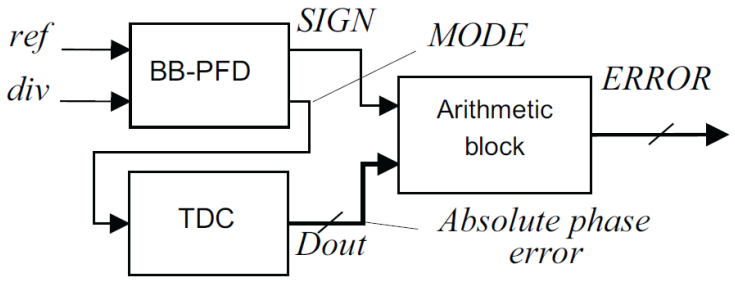


Figura A.2 – Estrutura do detetor de fase-frequência (PFD) [83].

A.2.1.1 Bang Bang – Phase frequency detector

O BB-PFD, cuja estrutura pode ser observada na Fig. A.3, é composto por dois *flip-flops* utilizados para fazer a deteção das transições nos dois sinais de relógio colocados à sua entrada. O *arbiter* deteta qual das transições ocorreu em primeiro lugar, produzindo um “1” ou “0” à sua saída. A *latch* de saída (X10) armazena o valor do sinal (*SIGN*), enquanto simultaneamente o elemento C (X8) gera um sinal de *reset* quando a saída dos dois *flip-flops* está bem definida.

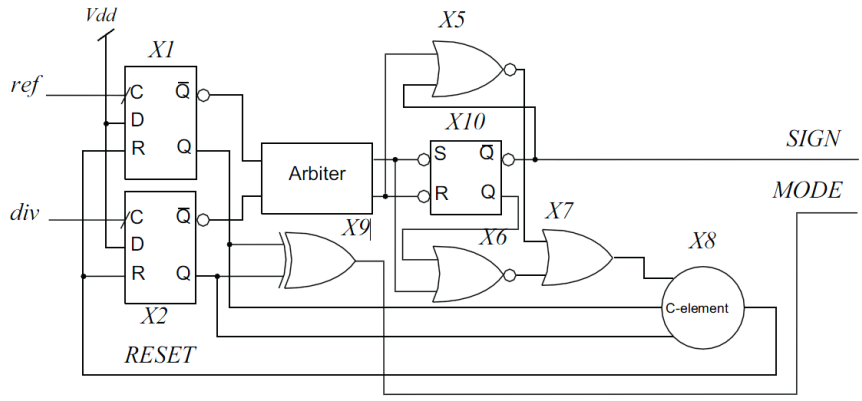


Figura A.3 – Estrutura do detetor *bang-bang* de fase-frequência [83].

O BB-PFD deteta o sinal do erro de fase (*SIGN*) e a amplitude do erro de fase (*MODE*) utilizando: dois *flip-flops* para deteção das transições ascendentes em ref e $div;$ um filtro de estados metastáveis[12] designado *arbiter*; um registo para armazenamento temporário; e um sistema lógico de *reset* [83]. Este tem uma gama dinâmica infinita em termos de diferença de fase entre as suas entradas. Ao contrário de um PFD linear, a saída do detetor mantém-se no nível lógico *high* ou *low* durante a totalidade do ciclo de relógio de referência, e não por um tempo proporcional à diferença de fase entre o sinal de relógio de referência, ref, e o sinal realimentado, div. Ao detetar uma transição em ref, este espera por uma transição em div, ignorando outras transições que poderão entretanto acontecer no primeiro sinal [84]. Como resultado o detetor de fase-frequência apresenta o comportamento ilustrado na Fig. A.4.

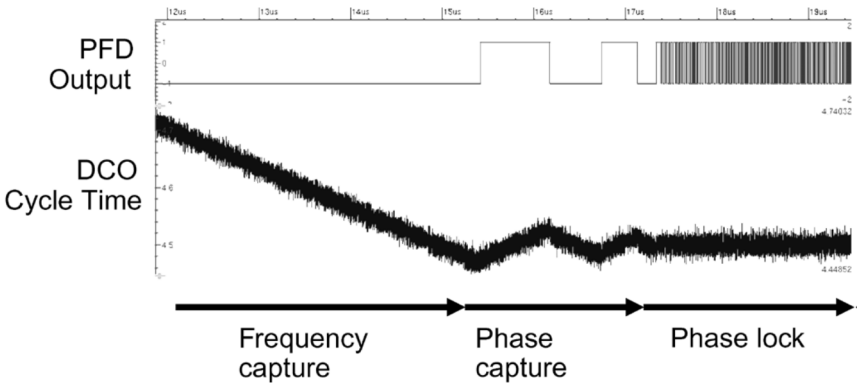


Figura A.4 – Ilustração da captura de frequência e fase na PLL descrita em [84].

Para uma explicação mais detalhada sobre o funcionamento da arquitetura dever-se-á consultar [83] e [84].

[12] Metaestabilidade – Fenómeno onde um elemento biestável entra num terceiro estado indesejável no qual a sua saída é um nível intermédio entre o nível lógico “0” e “1” [85].

A.2.1.2 Time-to-digital converter

É possível implementar um PFD com uma saída digital de múltiplos *bits* ao digitalizar a amplitude do erro de fase ao utilizar um TDC (*Time-to-Digital Converter*). Isto permite reduzir o tempo de sincronização do *loop* de controlo e melhorar a performance em relação ao ruído [86]. A amplitude do erro é convertida numa palavra binária sem sinal pelo TDC, traduzindo a duração do erro de fase numa representação interpretável digitalmente.

Existem arquiteturas que tornam possível realizar TDCs com grande resolução baseados numa arquitetura com elementos de atraso em paralelo ou em linhas de atraso de Vernier, permitindo medir impulsos menores que o atraso de um único *buffer* [86]. Embora útil em aplicações na gama dos GHz, no caso do projeto em causa não se justificava a utilização de uma arquitetura de TDC com tão elevada resolução. Por este motivo optou-se por uma arquitetura mais simples, composta por uma linha de atrasos formada por *buffers* com um tempo de atraso T_{dly}, representada na Fig. A.5. O sinal *MODE* é propagado por uma linha de *L* atrasos. Quando existe uma transição no flanco descendente, as saídas de todos os *buffers* são lidas para um registo e codificadas. A saída digital D_{out} é um código que representa a diferença temporal entre o sinal ref e div (e traduzida no sinal *MODE*) com uma resolução T_{dly} [86].

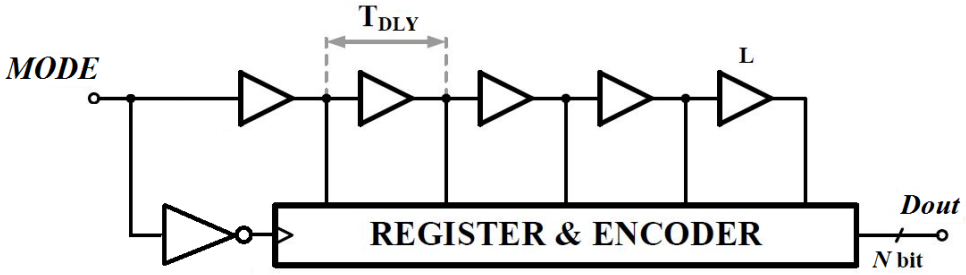


Figura A.5 – Arquitetura do TDC [86].

Embora a resolução do TDC dependa de variações de processo e condições de ambiente tal como a temperatura, esta implementação permite obter uma boa resolução tendo em conta que a frequência dos sinais ref e div não ultrapassarão as dezenas de MHz.

Tendo isto em conta dimensionou-se o TDC com uma linha de atraso composta por 12 *buffers*, obtendo a função de transferência ilustrada na Fig. A.6. O eixo dos xx indica o erro de fase $\Delta\varphi$ detetado pelo BB-PFD (*MODE*), e o eixo dos yy é a palavra digital $Dout$ do TDC.

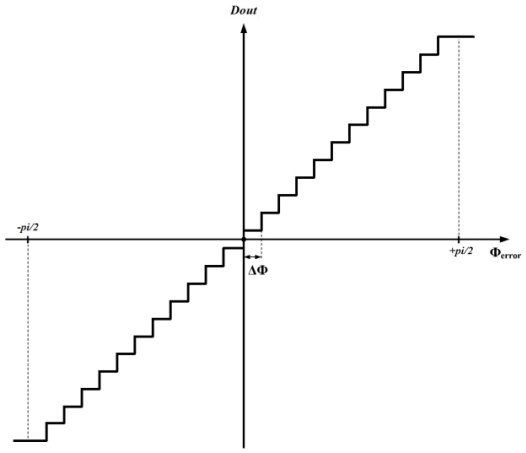


Figura A.6 – Função de transferência do TDC [86].

A.2.1.3 Bloco de aritmética

Esta última é combinada com *SIGN*, no bloco de aritmética (*Arithmetic block*), para formar uma palavra binária com sinal dada por,

$$ERROR = SIGN * (Dout + 1), \tag{A.1}$$

onde *SIGN* pode tomar os valores +1 ou -1. Para que se possa obter informação acerca do sinal para pequenos erros (nos quais o TDC gera 0), o bloco de aritmética adiciona 1 a $Dout$ [83].

A.2.2 Filtro digital

O filtro digital segue a estrutura representada na Fig. A.7. Este consiste num percurso proporcional com ganho K_P e num percurso integral com um ganho K_I, e um atraso proporcionado por um *flip-flop* [86]. No domínio discreto a função de transferência do filtro digital é escrita como,

$$H(z) = K_P + \frac{K_I}{1 - z^{-1}} T_{CK}. \tag{A.2}$$

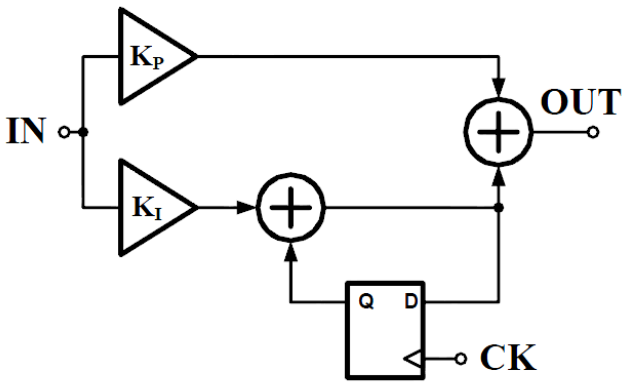


Figura A.7 – Filtro PI Digital [86].

O ganho proporcional e o ganho integral foram determinados como, $K_P = 0.134075$ e $K_I = 0.0005$, por meio da ferramenta MATLAB/SIMULINK através do bloco *Step Response Specifications*[13] da *Simulink Response Optimization Toolbox*. O uso desta ferramenta justifica-se pela difícil análise teórica deste sistema de controlo, devido ao seu comportamento não linear (*reset* do erro de fase-frequência).

A.3 Simulação em MATLAB/SIMULINK

A.3.1 Implementação em simulink

O modelo utilizado na simulação em *software* MATLAB/SIMULINK segue a estrutura ilustrada na Fig. A.8. Modelou-se o sistema com blocos idênticos aos que foram sintetizados pela FPGA na implementação prática. Os sub-blocos, visíveis na Fig. A.8, encontram-se representados em maior detalhe no Anexo B.

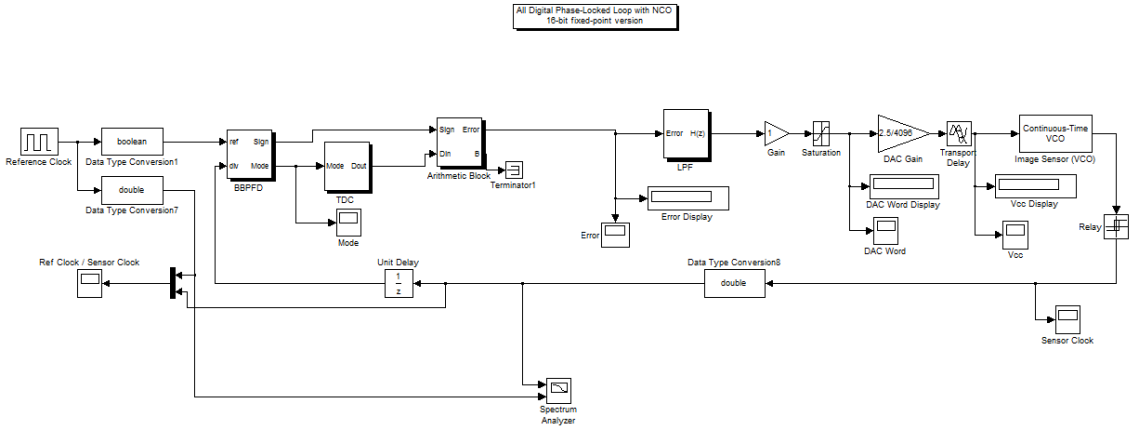


Figura A.8 – Estrutura do módulo ADPLL.

A.3.2 Resultados Simulink

Verificou-se que o sistema leva aproximadamente 7,6 ms a alcançar o sincronismo (*Phase Lock*) entre o sinal de referência e o relógio do sensor. Na Fig. A.9 é possível observar a evolução do valor da palavra de comando da DAC à saída do filtro digital.

[13] Esta ferramenta determina os valores ótimos dos ganhos utilizando métodos de otimização numérica como Gradiente Descendente [87].

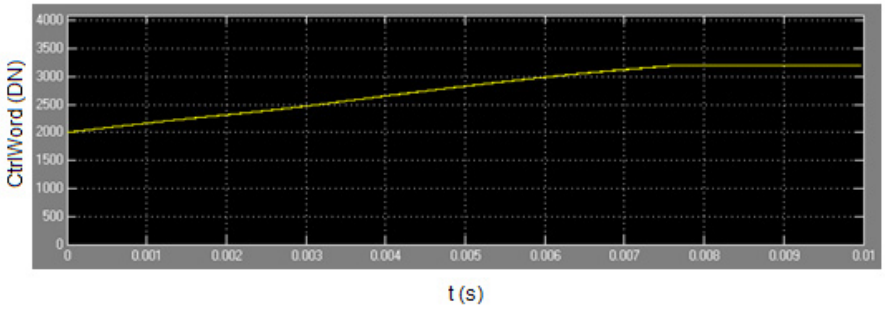


Figura A.9 – Saída do filtro digital.

O erro absoluto de fase à saída do bloco de aritmética apresenta o comportamento ilustrado na Fig. A.10. Quando o sistema alcança o sincronismo em frequência e em fase o seu comportamento é caracterizado por oscilações em torno de 0, tal como ilustrado na amplificação do gráfico.

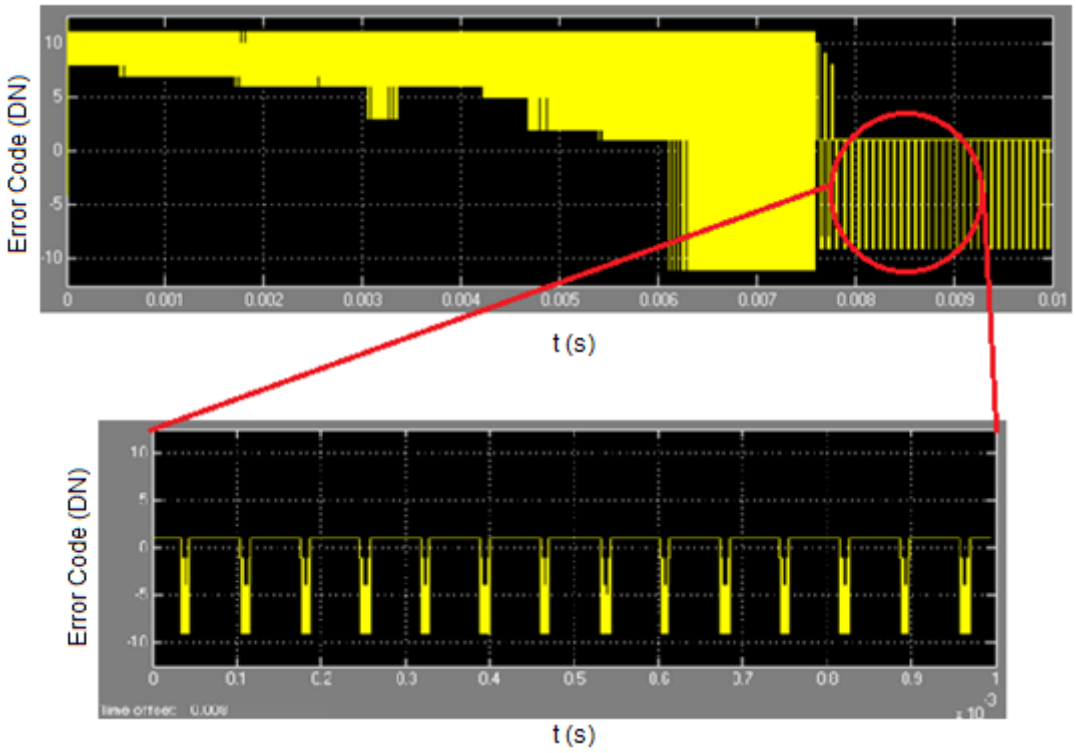


Figura A.10 – Erro absoluto de fase à saída do bloco de aritmética (*Phase Lock*).

Nas Fig. A.11 até A.13 é possível observar o erro de fase entre o sinal de relógio de referência (a amarelo) e o sinal de relógio do sensor de imagem (a roxo). Nas condições indicadas na Tab B.1 do Anexo B, extraíram-se os erros de fase entre os dois *clocks*, medindo nos gráficos a diferença temporal entre as duas transições. Na Tab. A.1 foram compilados os erros de fase, máximo, mínimo e médio, obtidos através deste método.

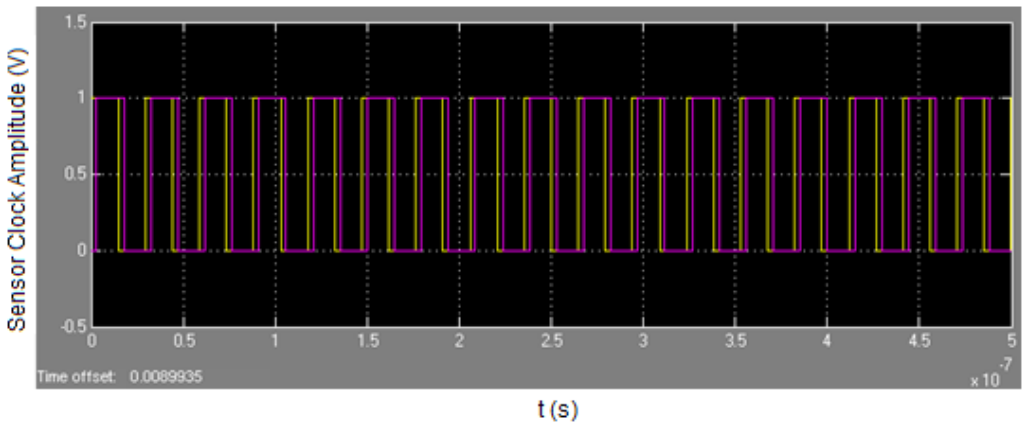


Figura A.11 – Erro de sincronismo entre o relógio de referência (amarelo) e o relógio do sensor (roxo) (1).

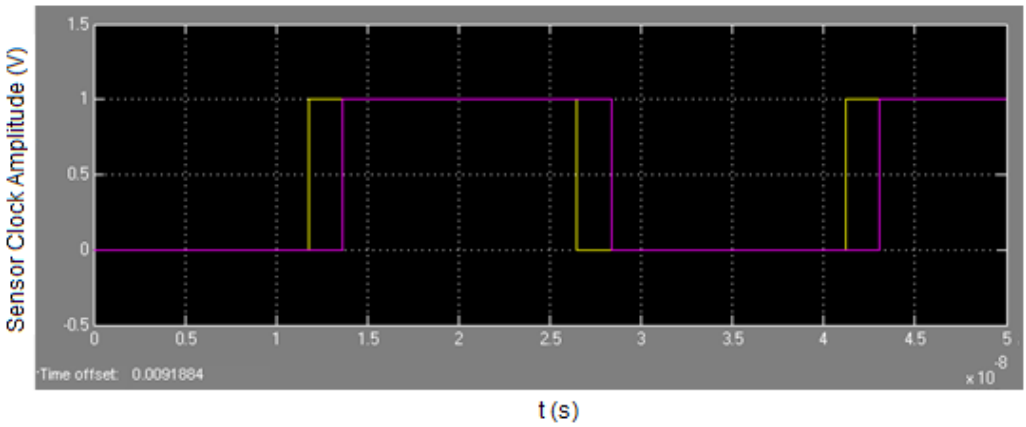


Figura A.12 – Erro de sincronismo entre o relógio de referência (amarelo) e o relógio do sensor (roxo) (2).

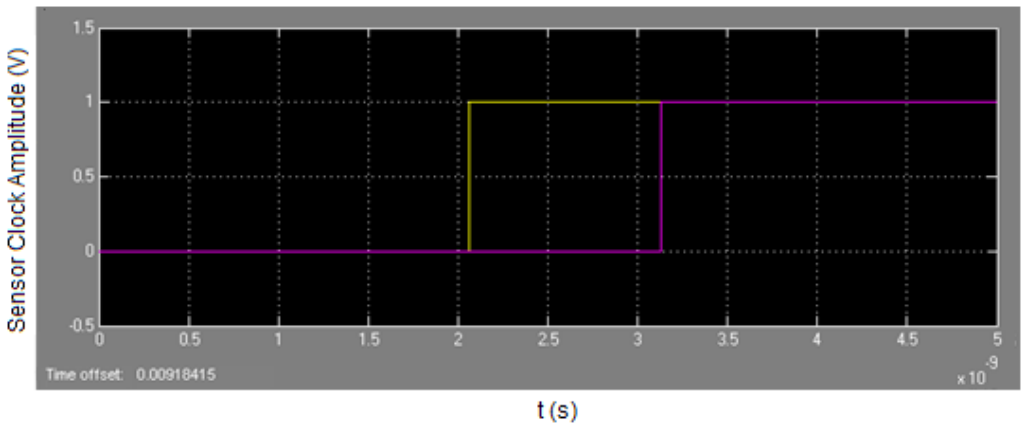


Figura A.13 – Erro de sincronismo entre o relógio de referência (amarelo) e o relógio do sensor (roxo) (3).

Tabela A.1 – Erros de fase entre *frames* obtidos via simulação (ADPLL).

MATLAB/SIMULINK	
$Max(\varphi_{Error})_{ADPLL}$	3,51 ns
$Min(\varphi_{Error})_{ADPLL}$	2,01 ns
$Avg(\varphi_{Error})_{ADPLL}$	2,78 ns

O erro de fase medido entre os dois sinais de relógio representa, em média, 9,6 % do período total do relógio de referência. Contudo não existe uma base de comparação com [83] visto que neste não é especificado o erro de sincronização entre dois sinais de relógio. Para mais o modelo implementado nesta simulação tem em conta os atrasos do sistema de controlo, implicando um compromisso entre velocidade de ajuste e erro de fase. Algo que não é indicado em [83].

A.4 Implementação na FPGA

A implementação da ADPLL na prática seguiu a estrutura descrita na secção A.2, resultando na arquitetura de *hardware* ilustrada na Fig. A.14. Foi incluído um bloco denominado *ref_clock_generator*, constituído por um DCM[14] (*Digital Clock Manager*) gerado a partir de um *IP Core* da Xilinx. Este tem a função de gerar o sinal de relógio com a frequência de referência que a ADPLL deverá seguir.

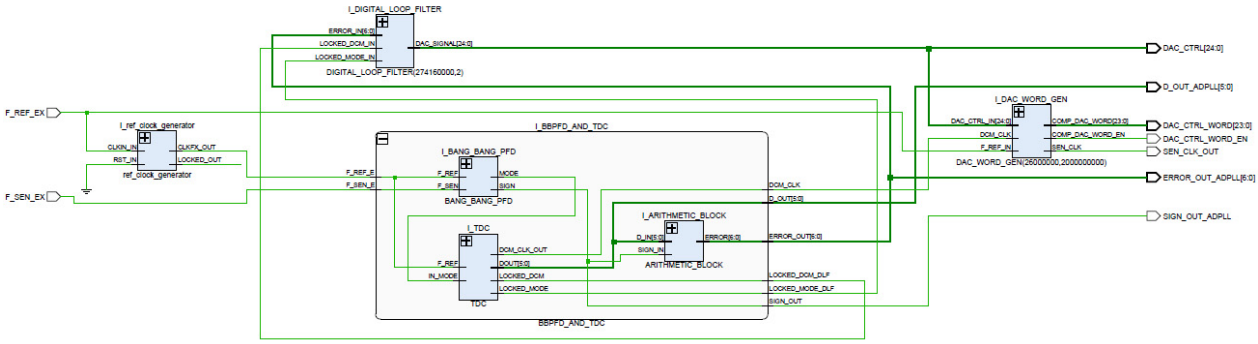


Figura A.14 – Estrutura do módulo ADPLL.

Além deste foi também incluído o sub-módulo *dac_word_gen* cuja função é realizar o condicionamento da palavra digital à saída do filtro para escrita na DAC acoplada ao sensor de imagem, segundo o protocolo SPI.

O sinal de relógio do sensor é extraído do *stream* de dados Manchester utilizando uma técnica desenvolvida no decorrer da realização deste trabalho. Tipicamente a extração do sinal de relógio a partir de uma codificação Manchester exige a utilização de um *clock* de amostragem com uma frequência muito superior ao *stream* de dados acoplado a uma PLL [88]. Contudo para obter uma boa resolução e baixo erro na sintetização do sinal de relógio é necessário

[14] DCM – Gestor de sinal de relógio digital que permitem realizar a multiplicação e/ou divisão da frequência de um *clock* de entrada com o objetivo de sintetizar um outro sinal de relógio com uma frequência distinta [89].

que a frequência do *clock* de amostragem seja várias ordens de grandeza superior à frequência do *stream* de dados. Como na plataforma NanoUSB2 apenas é possível gerar um sinal de relógio com frequência máxima de 270 MHz [89], chegou-se à conclusão que esta frequência de amostragem não era suficiente para extrair o sinal de relógio do sensor sem distorção (com um *duty cycle* de exatamente 50%). Fez-se uso de uma máquina de estados já implementada no *firmware* existente, cuja função é gerar um sinal com impulsos *full-bit* (correspondentes a um período inteiro do sinal de relógio do sensor) e impulsos *half-bit* (correspondentes a meio período do sinal de relógio do sensor), monitorizando as transições de sinal no *stream* de dados, tal como ilustrado na Fig. A.15. Ao eliminar seletivamente alguns dos impulsos, é possível sintetizar um sinal de relógio cujo período é igual a 2 vezes o período do relógio do sensor. No fundo este sinal resultante é o relógio do sensor cuja frequência foi dividida por 2, portanto ao adaptar os parâmetros do resto da ADPLL para esta condição, como por exemplo diminuindo a frequência do sinal F_REF para metade, é possível realizar o controlo do *frame rate* do sensor para o valor desejado.

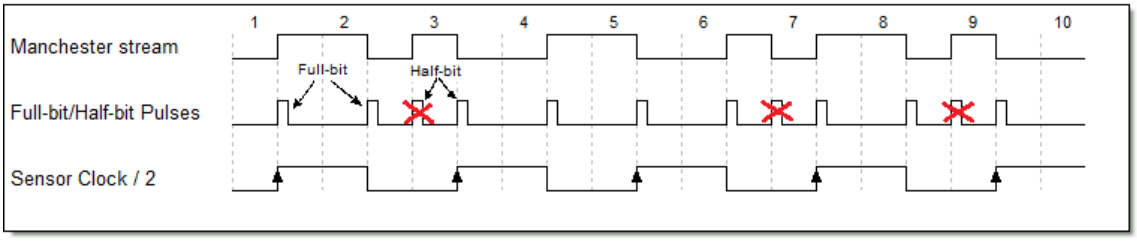


Figura A.15 – Extração do sinal de relógio do sensor de imagem.

Contudo, apesar de todo o desenvolvimento anteriormente citado, chegou-se à conclusão que o sistema de controlo de frequência com base numa ADPLL era impraticável devido à impossibilidade de gerar um sinal de referência, F_REF, com uma frequência arbitrária. Na prática este sinal de referência seria gerado por intermédio de um DCM. Este teria por base o sinal de relógio de sistema (48 MHz), elevaria o mesmo para a frequência máxima de 270 MHz e a partir deste último seriam gerados os vários passos de frequência, através de divisores de frequência. O problema desta configuração é que o DCM apenas é capaz de gerar múltiplos inteiros da frequência original. Como tal, passos de frequência como 30 MHz, 31 MHz, 32 MHz, etc, são impossíveis de gerar, o que impossibilitou a utilização desta arquitetura.

A.5 Conclusões

A arquitetura de controlo de frequência baseada numa ADPLL foi estudada, analisada e implementada em simulação. O *software* MATLAB/SIMULINK foi utilizado como meio para realização de testes ao

comportamento dinâmico do sistema e determinação dos fatores proporcional e integral do filtro digital. Foi também implementado um modelo em VHDL e testado com a ferramenta de simulação ModelSim, contudo como esta solução foi abandonada em detrimento do *Frequency Comparator*, não se extraíram resultados práticos úteis deste procedimento. Para mais a frequência do sinal de referência, gerado internamente na FPGA, teria uma dependência da temperatura pois os próprios DCMs contêm na sua arquitetura DLLs (*Delay-Locked Loops*) [89]. Note-se que a própria FPGA, quando em funcionamento, pode aquecer em casos extremos a temperaturas em torno dos 100 °C [90]. Sendo o aumento da temperatura dependente, entre outros fatores, da quantidade de lógica utilizada e da frequência dos sinais de relógio internos [90].

Anexo B - Blocos e Parâmetros utilizados na simulação em MATLAB/SIMULINK do módulo de controlo ADPLL

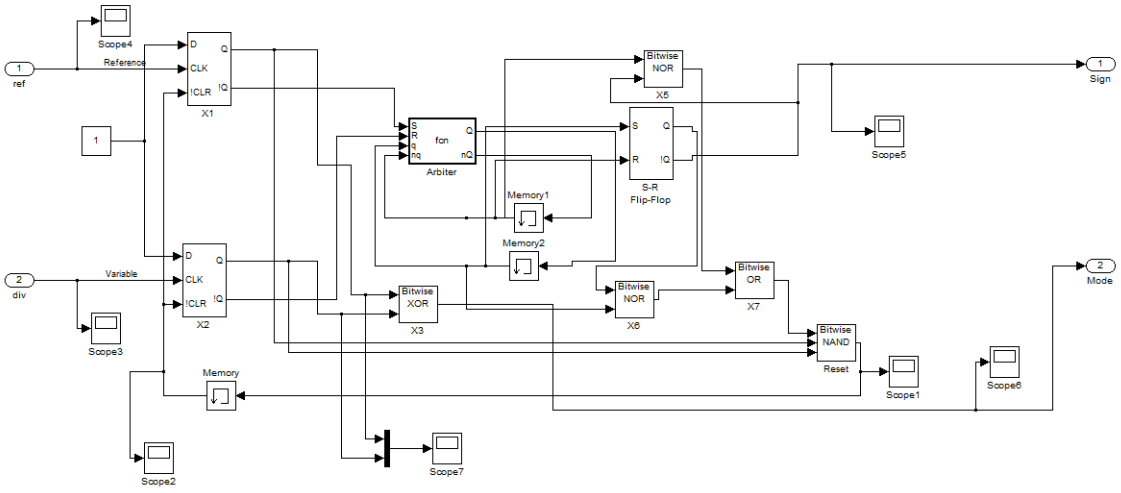


Figura B.1 – Estrutura interna do bloco *Bang Bang Phase-Frequency Detector*.

```
function [Q,nQ] = fcn(S,R,q,nq)
% This block supports the Embedded MATLAB subset.
% See the help menu for details.
Q=boolean(q);
nQ=boolean(nq);
if (S==0 && R==0 && q==nq)
    Q=boolean(rand(1));
    %Q=boolean(1);
    nQ=not(Q);
elseif (S==0 && R==0)
    Q=q;
    nQ=nq;
elseif (S==1 && R==1)
    Q=boolean(0);
    nQ=boolean(0);
elseif (S==1)
    Q=boolean(1);
    nQ=boolean(0);
elseif (R==1)
    Q=boolean(0);
    nQ=boolean(1);
end;
```

Listagem B.1 – Código do bloco *Arbiter*.

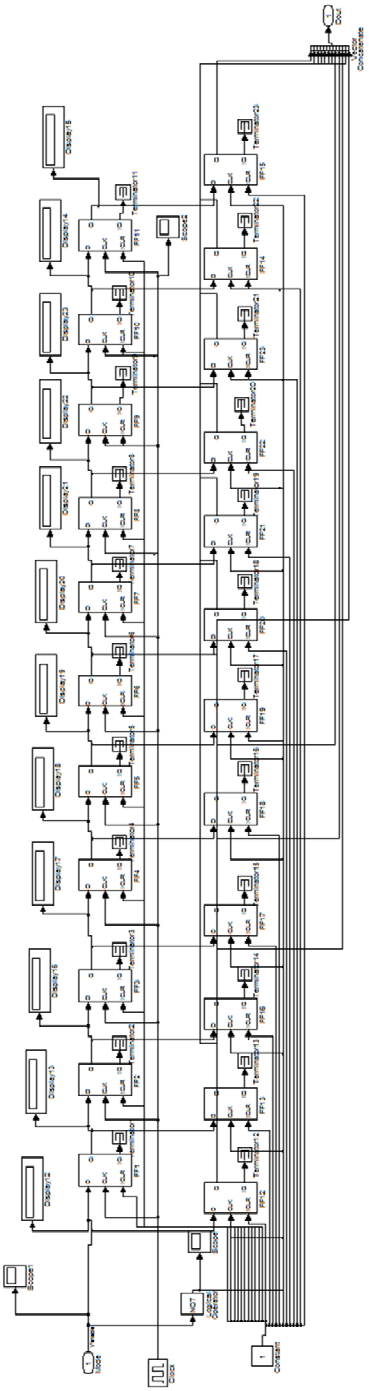


Figura B.2 – Estrutura interna do bloco *Time-to-Digital Converter*.

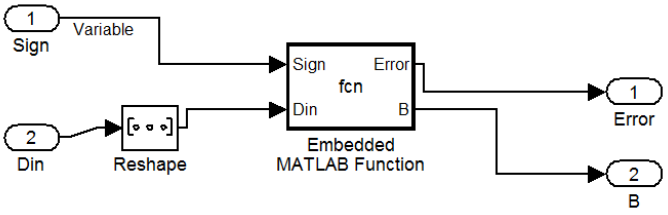


Figura B.3 – Estrutura interna do bloco *Arithmetic Block*.

```
function [Error,B] = fcn(Sign,Din)

if (Sign == 1)
    Sinal = 1;
else
    Sinal = -1;
end;

B = sum(Din.*2.^(numel(Din)-1:-1:0));

if (B == -1)
    Cod=0;
elseif (B>=1 && B<3)
    Cod=1;
elseif (B>=3 && B<7)
    Cod=2;
elseif (B>=7 && B<15)
    Cod=3;
elseif (B>=15 && B<31)
    Cod=4;
elseif (B>=31 && B<63)
    Cod=5;
elseif (B>=63 && B<127)
    Cod=6;
elseif (B>=127 && B<255)
    Cod=7;
elseif (B>=255 && B<511)
    Cod=8;
elseif (B>=511 && B<1023)
    Cod=9;
elseif (B>=1023 && B<2047)
    Cod=10;
elseif (B>=2047 && B<4095)
    Cod=11;
elseif (B==4095)
    Cod=12;
else
    Cod=0;
end;

Error=Sinal*(Cod);
```

Listagem B.2 – Código do *Arithmetic Block*.

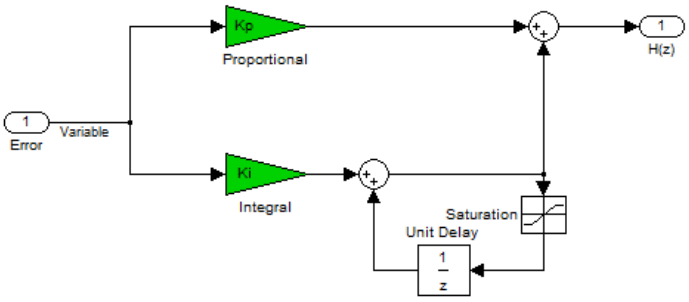


Figura B.4 – Estrutura interna do bloco *Loop Filter*.

Tabela B.1 – Parâmetros utilizados na simulação em MATLAB/SIMULINK da ADPLL.

Bloco	Parâmetro	Valor	Descrição
Reference Clock	Amplitude	$1\ V$	Amplitude do sinal de relógio de referência
	Period	$\frac{1}{34 \times 10^6}\ s$	Período do sinal de relógio de referência
	Pulse Width	50%	*Duty cycle* do sinal de relógio de referência
	Phase delay	$0\ s$	Atraso de fase do sinal de relógio de referência
LPF	K_p	$0{,}134075$	Ganho proporcional do compensador da ADPLL
	K_i	$0{,}0005$	Ganho integral do compensador da ADPLL
	Upper Limit	4095	Limite superior do saturador do filtro
	Lower Limit	1	Limite inferior do saturador do filtro
DAC Gain	Gain	$\frac{2{,}5\ V}{4096}$	Ganho da DAC
Transport Delay	Time delay	$4{,}4 \times 10^{-6}$	Atraso provocado pela DAC

Tabela B.1 – Parâmetros utilizados na simulação em MATLAB/SIMULINK da ADPLL (continuação).

Image Sensor (VCO)	Output amplitude	$1\ V$	Amplitude do sinal de relógio da câmara
	Quiescent frequency	$10\ Hz$	A frequência de saída do oscilador quando o sinal de entrada é zero
	Input sensitivity	$17{,}432 \times 10^6\ Hz/V$	Sensibilidade do VCO da câmara
	Initial phase	$0\ rad$	Fase inicial do oscilador da câmara

Anexo C - Blocos e Parâmetros utilizados na simulação em MATLAB/SIMULINK do módulo de controlo *Frequency Comparator*

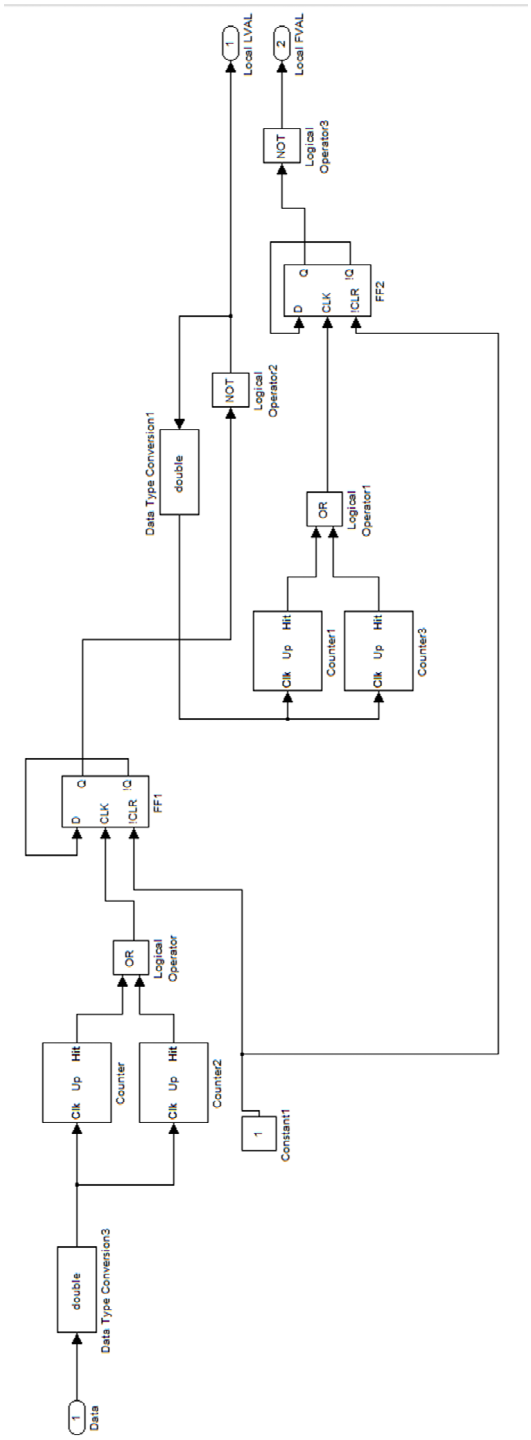


Figura C.1 – Estrutura interna do bloco *Deserializer*.

Tabela C.1 – Parâmetros utilizados na simulação em MATLAB/SIMULINK do bloco *Deserializer*.

Bloco	Parâmetro	Valor	Descrição
Counter	Max count	3024	Este bloco conta o número de transições do sinal de relógio do sensor e envia um impulso à saída quando a contagem chega ao *Hit value*. Ao utilizar a operação OR e um Flip-Flop é possível gerar os sinais FVAL e LVAL. Os valores das contagens foram definidos de forma a corresponderem aos sinais de linha e de *frame* do NanEye: • 12 bit x 252 PP = 3024 ciclos • 12 bit x 249 PP = 2988 ciclos • 250 linhas de dados + 3 linhas a '0' = 1 frame period
	Initial count	0	
	Output	hit	
	Hit value	2988	
Counter 2	Max count	3024	
	Initial count	0	
	Output	hit	
	Hit value	3024	
Counter 1	Max count	253	
	Initial count	0	
	Output	hit	
	Hit value	250	
Counter 3	Max count	253	
	Initial count	0	
	Output	hit	
	Hit value	253	

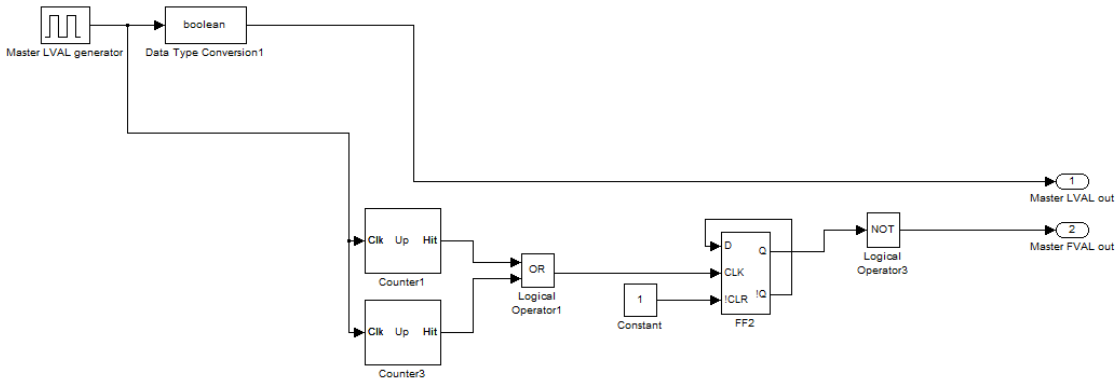


Figura C.2 – Estrutura interna do bloco *Master Camera*.

Tabela C.2 – Parâmetros utilizados na simulação em MATLAB/SIMULINK do bloco *Master Camera*.

Bloco	Parâmetro	Valor	Descrição
Master LVAL gen	Amplitude	$1\ V$	Amplitude do sinal LVAL da câmara *Master*
	Period	$\frac{1}{12650}\ s$	Período do sinal LVAL da câmara *Master*
	Pulse Width	$92{,}39\ \%$	*Duty cycle* do sinal LVAL da câmara *Master*
	Phase delay	$4{,}3 \times 10^{-5}\ s$	Atraso de fase do sinal LVAL da câmara *Master*
Counter 1	Max count	253	Este bloco conta o número de transições do sinal de relógio do sensor e envia um impulso à saída quando a contagem chega ao *Hit value*. Ao utilizar a operação OR e um Flip-Flop é possível gerar os sinais FVAL e LVAL. Os valores das contagens foram definidos de forma a corresponderem aos sinais de linha e de *frame* do NanEye: • 250 linhas de dados + 3 linhas a '0' = 1 frame period
	Initial count	0	
	Output	hit	
	Hit value	250	
Counter 3	Max count	253	
	Initial count	0	
	Output	hit	
	Hit value	253	

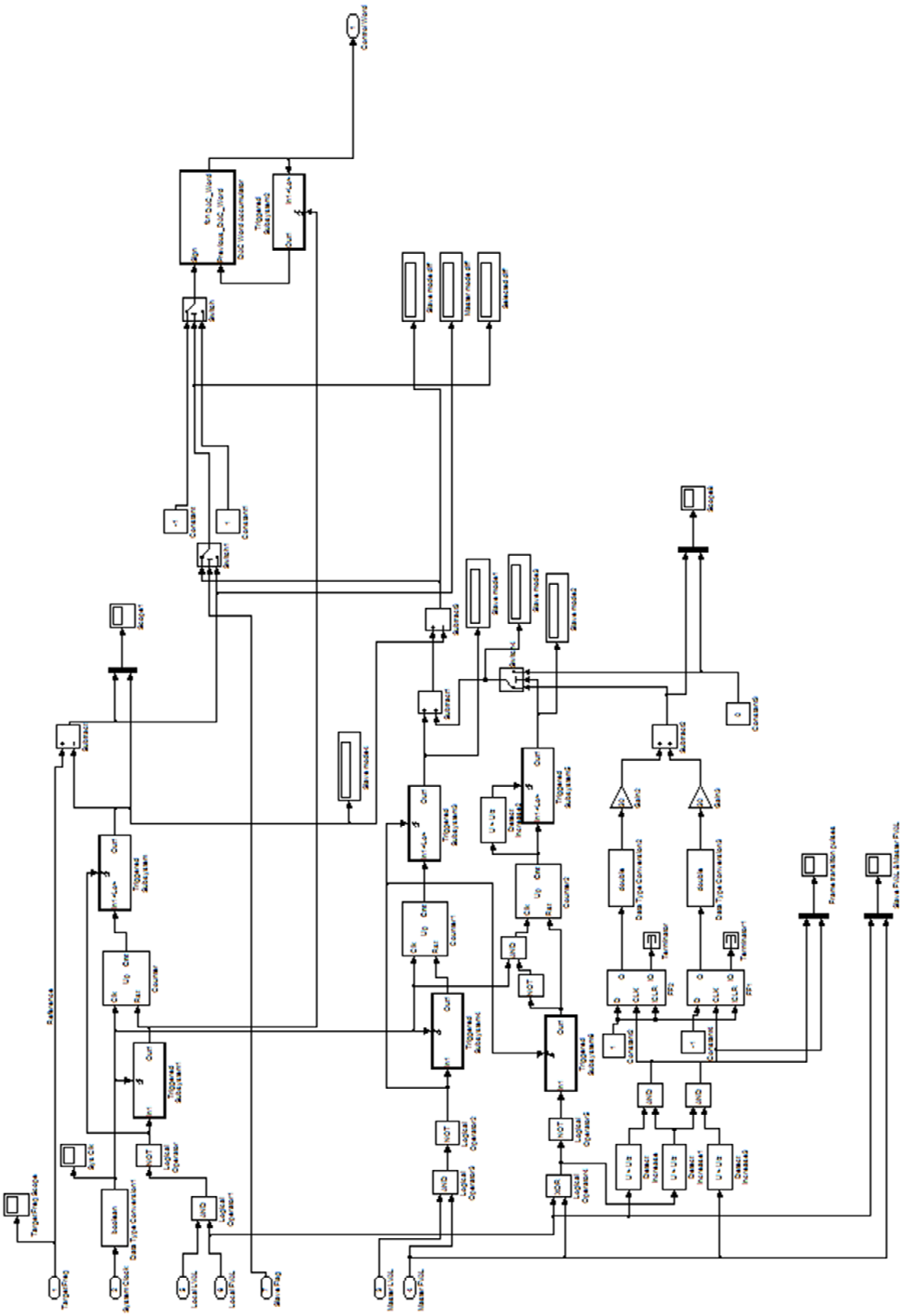


Figura C.3 – Estrutura interna do bloco *Frequency Comparator*.

Tabela C.3 – Parâmetros utilizados na simulação em MATLAB/SIMULINK do bloco *Frequency Comparator*.

Bloco	Parâmetro	Valor	Descrição
Constant 2	Constant value	1	Passo de correção lento (sentido positivo) do desfasamento entre *frames*.
Constant 4	Constant value	-1	Passo de correção lento (sentido negativo) do desfasamento entre *frames*.
Gain 2	Gain	30	Passo de correção rápido (sentido positivo) do desfasamento entre *frames*.
Gain 3	Gain	-30	Passo de correção rápido (sentido negativo) do desfasamento entre *frames*.
Counter	Max count	65535	Estes blocos contam o número de transições do sinal de relógio de sistema dentro das linhas contidas num período de *frame*. A contagem é analisada pelos restantes blocos de forma a que a que o sistema possa atuar.
	Initial count	0	
	Output	$count$	
	Hit value	$não\ utilizado$	
Counter 1	Max count	65535	
	Initial count	0	
	Output	$count$	
	Hit value	$não\ utilizado$	
Counter 2	Max count	65535	
	Initial count	0	
	Output	$count$	
	Hit value	$não\ utilizado$	

```
function DAC_Word  = fcn(Sign,Previous_DAC_Word)
      DAC_Word = Previous_DAC_Word + Sign;
      if (DAC_Word < 1000)
          DAC_Word=4000;
      elseif (DAC_Word > 4095)
          DAC_Word=4095;
      else
          DAC_Word=DAC_Word;
      end;
```

Listagem C.1 – Código do sub-bloco *DAC Word Accumulator*.

Tabela C.4 – Parâmetros utilizados na simulação em MATLAB/SIMULINK dos restantes blocos.

Bloco	Parâmetro	Valor	Descrição
DAC Gain	Gain	$\frac{2{,}5\ V}{4096}$	Ganho da DAC
Target Freq	Constant value	4363	Número de ciclos de relógio alvo
Sys Clk	Amplitude	$1\ V$	Amplitude do sinal de relógio de sistema
	Period	$\frac{1}{48 \times 10^6}\ s$	Período do sinal de relógio de sistema
	Pulse Width	50%	*Duty cycle* do sinal de relógio de sistema
	Phase delay	$0\ s$	Atraso de fase do sinal de relógio de sistema
Image Sensor (VCO)	Output amplitude	$1\ V$	Amplitude do sinal de relógio da câmara
	Quiescent frequency	$10\ Hz$	A frequência de saída do oscilador quando o sinal de entrada é zero
	Input sensitivity	$17{,}432 \times 10^6\ Hz/V$	Sensibilidade do VCO da câmara
	Initial phase	$0\ rad$	Fase inicial do oscilador da câmara

Anexo D - Implementação em *hardware* dos sub-blocos do *Frequency Comparator*

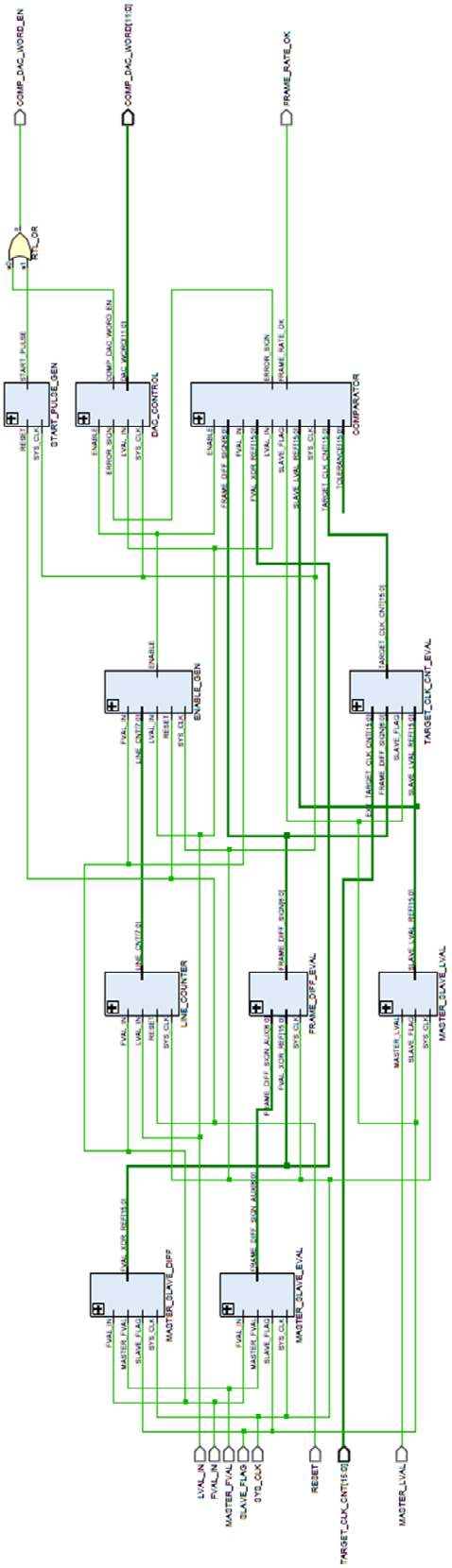


Figura D.1 – Estrutura do módulo *Frequency Comparator*.

D.1 Master slave evaluation

No modo *Slave*, o processo *master_slave_eval* (Fig. D.2) permite determinar se o sinal FVAL local se encontra em avanço ou em atraso em relação ao sinal *Master* FVAL. Para tal é detetado o flanco descendente de cada um dos sinais e comparado o instante da sua ocorrência. À saída temos um sinal de correção que pode tomar um valor positivo ou negativo consoante o estado detetado. Se o sensor estiver a operar em modo independente ou *master* (*Slave_Flag* = "0"), a saída do bloco é zero.

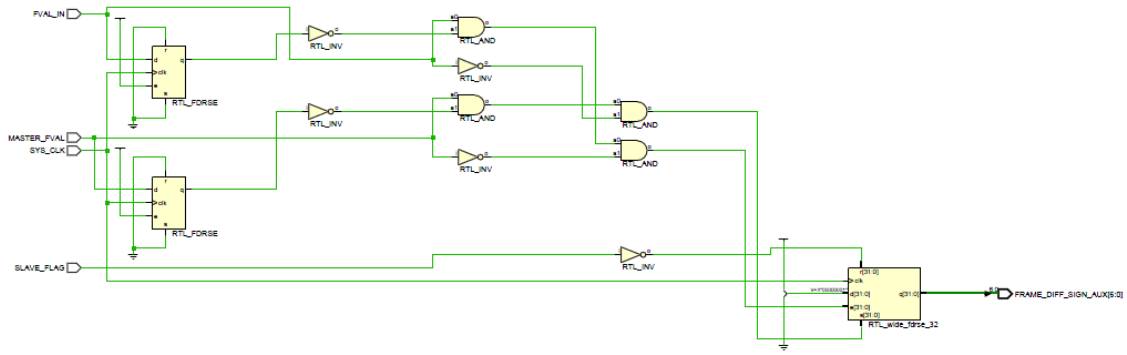


Figura D.2 – Implementação em *hardware* do bloco MASTER_SLAVE_EVAL.

D.2 Master slave difference

Considerando que o sistema está a operar em modo *Slave*, o bloco *master_slave_diff* (Fig. D.3) é responsável por determinar a diferença em número de transições do sinal de relógio entre o FVAL local e o FVAL proveniente do *Master*. Para tal aplica a operação OU Exclusivo sobre estes dois sinais e faz a contagem do número de *clocks* dentro do período *High* do sinal resultante. À saída do sistema temos um valor binário proporcional à amplitude do erro de fase medido.

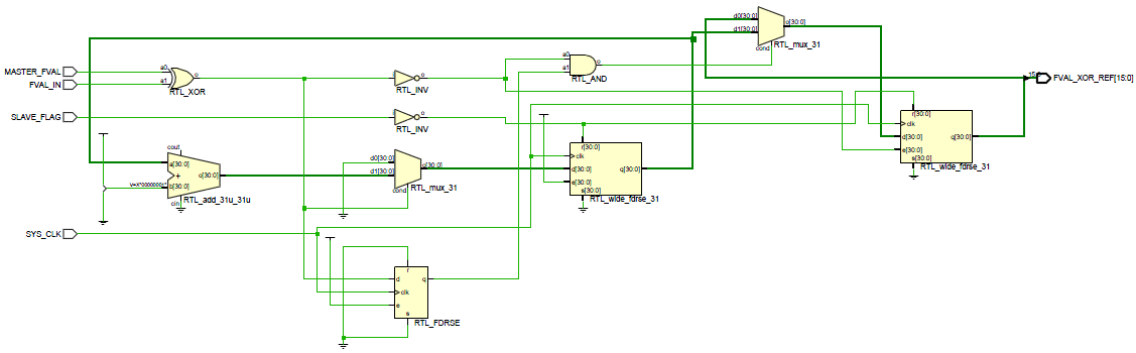


Figura D.3 – Implementação em *hardware* do bloco MASTER_SLAVE_DIFF.

D.3 Frame difference evaluation

Consoante a amplitude do erro o bloco *frame_diff_eval* (Fig. D.4) amplifica ou mantém a correção efetuada pelo processo *master_slave_eval*. Assumindo, por exemplo, que o erro de fase é positivo e superior a 3000 ciclos do sinal de relógio, então este processo terá que ajustar a correção de 1 para 30 DN (*Digital Number*) por cada *frame*.

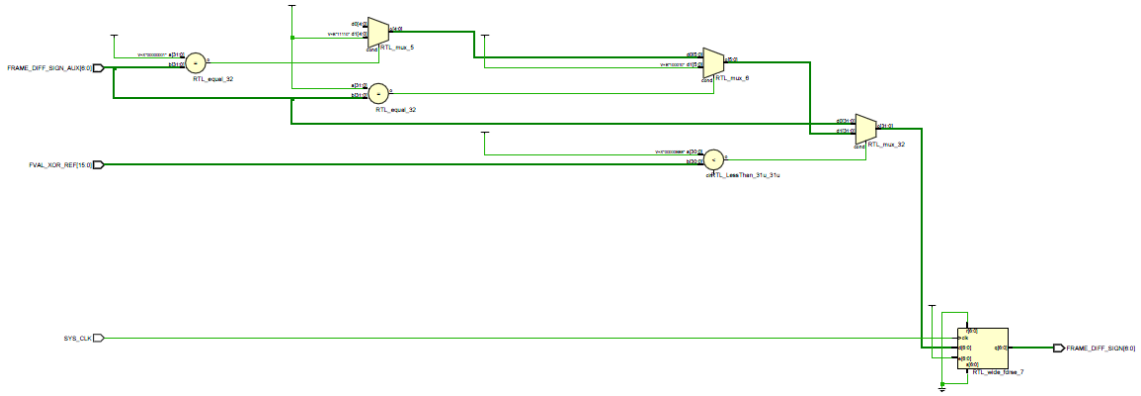


Figura D.4 – Implementação em *hardware* do bloco FRAME_DIFF_EVAL.

D.4 Master slave line valid

Simultaneamente o processo *master_slave_lval* (Fig. D.5) é responsável por medir o período do sinal de linha proveniente da câmara *Master* (*Master* LVAL). Para tal faz uma contagem dentro do período deste sinal tendo por base também o sinal de relógio de sistema (*Sys_Clk*). O propósito desta contagem é poder-se efetuar um ajuste da frequência de operação do sensor local com o *line rate* do sensor *master* (isto considerando operação do sensor em modo *Slave*).

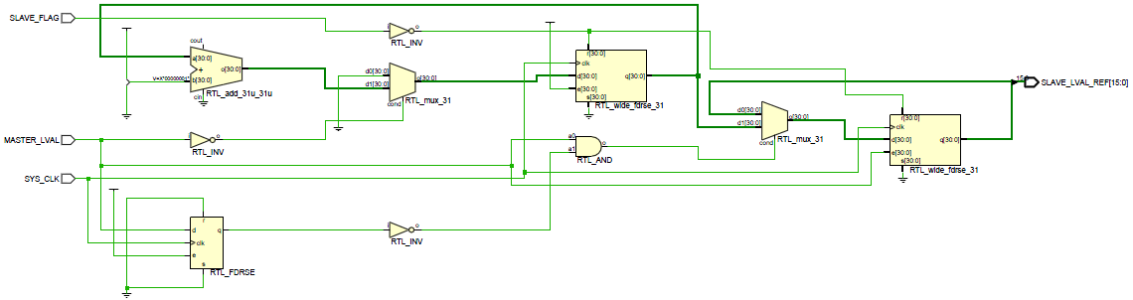


Figura D.5 – Implementação em *hardware* do bloco MASTER_SLAVE_LVAL.

D.5 Target clock counter evaluation

As saídas destes últimos dois processos são utilizadas no bloco denominado *target_clk_cnt_eval* (Fig. D.6) para gerar a contagem alvo para a qual o sensor se deve ajustar,

$$Target_Clk_Cnt = Slave_LVAL_Ref + Frame_Diff_Sign, \qquad (D.1)$$

sendo $Slave_LVAL_Ref$ o valor do período da linha numa *frame* proveniente da câmara *Master* em número de ciclos do sinal de relógio; e $Frame_Diff_Sign$ o valor da correção do erro de fase das *frames* dada pelo processo *frame_diff_eval* também em número de ciclos do sinal de relógio. Consoante o erro de fase entre as *frames* o valor contido em $Frame_Diff_Sign$ será positivo ou negativo, somando-se ou subtraindo-se ao valor base dado por $Slave_LVAL_Ref$ compensando assim o $Target_Clk_Cnt$ pelo erro de fase entre as *frames*. Contudo, se considerar-se que a câmara está a operar em modo *Master* e não *Slave*, o valor dado por $Target_Clk_Cnt$ é diretamente controlado externamente a partir do ambiente gráfico através da interface USB.

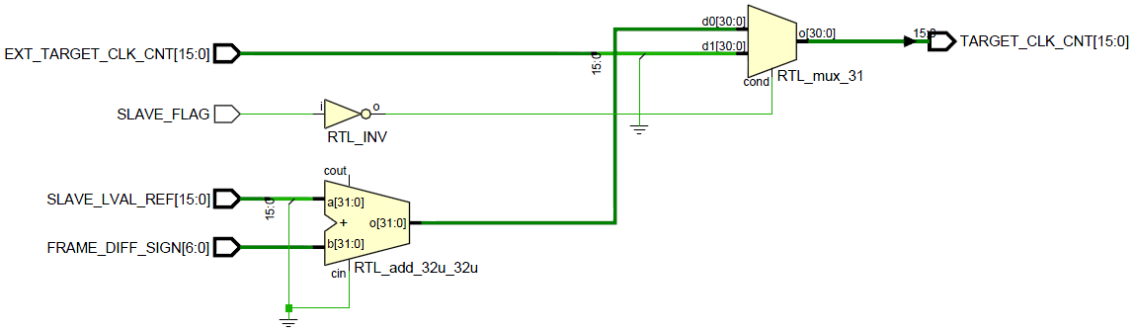


Figura D.6 – Implementação em *hardware* do bloco TARGET_CLK_CNT_EVAL.

D.6 Line counter

O bloco denominado *line_counter* (Fig. D.7) é utilizado para efetuar a contagem do número de linhas lidas pelo sistema. Esta contagem é usada no bloco *enable_gen* para efetuar a sincronização do sinal de *enable* com o fim da *frame*. Com o *enable* ativo é permitida a avaliação dos sinais de controlo no *dac_control* e no *comparator*.

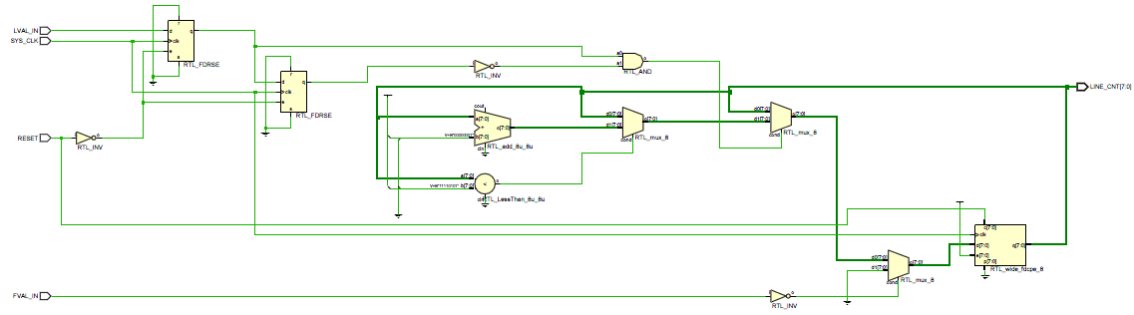


Figura D.7 – Implementação em *hardware* do bloco LINE_COUNTER.

D.7 DAC control

O bloco *dac_control* (Fig. D.8) avalia no início de cada linha da *frame* o sinal de erro dado pelo sinal *Error_Sign*. Se for detetado um “1” significa que a frequência de operação do sensor está acima do valor desejado, logo ao valor escrito na DAC é subtraído 1 DN. Se por outro lado for detetado um “0” implica que a frequência de operação está abaixo do pretendido, logo é somado 1 à palavra de 12 bit escrita na DAC. Deste modo obtém-se um sistema que ajusta a tensão aplicada ao sensor consoante o erro de frequência medido compensando simultaneamente o desvio de fase entre *frames* provenientes de sensores diferentes. Para evitar um possível *overflow/underflow* provocado pelas constantes somas ou subtrações efetuadas na palavra escrita na DAC, este bloco incorpora também um mecanismo limitador. Devido ao fenómeno do *overflow*, se o valor da palavra já for 4095 e a este for somado um valor adicional implicaria um resultado não de 4096 mas sim 0. A saída *Comp_DAC_Word_En* apenas é ativada quando existe a alteração da palavra que será escrita na DAC.

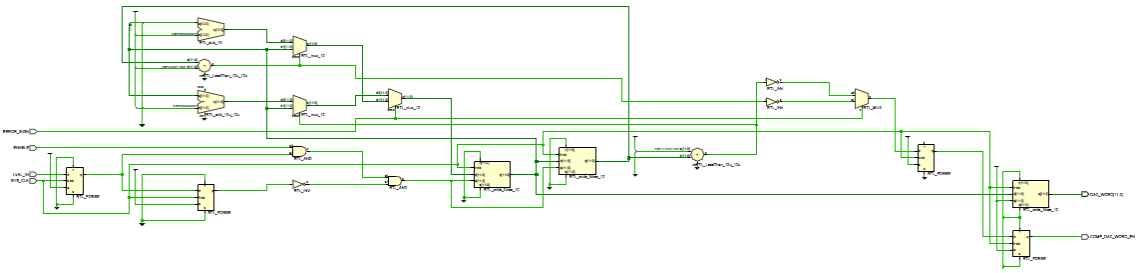


Figura D.8 – Implementação em *hardware* do bloco DAC_CONTROL.

D.8 Start pulse generator

Por fim o bloco *start_pulse_gen* (Fig. D.9), é utilizado para gerar um primeiro impulso de ativação da DAC depois da desativação do sinal de *reset*. Assim garante-se que após um *reset* existirá uma tensão aplicada no sensor permitindo a este enviar dados para a FPGA e possibilitando o resto do módulo de controlo ler as *frames* e ajustar a tensão aplicada de acordo.

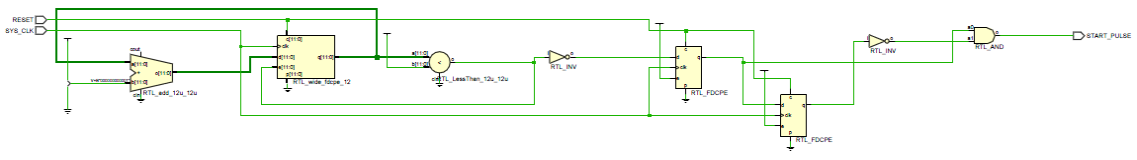


Figura D.9 – Implementação em *hardware* do bloco START_PULSE_GEN.

D.9 Comparator

Determinado o $Target_Clk_Cnt$ através dos processos referidos anteriormente, o bloco *comparator* (Fig. D.10) encarrega-se de fazer a medição do período da linha do sensor de imagem acoplado localmente (neste caso a entrada *Lval_In*). Quando é detetado o fim de linha através da transição do sinal *Lval_In* do estado *High* para *Low*, a contagem pára, e o seu valor é comparado com o valor alvo $Target_Clk_Cnt$. Se a contagem estiver abaixo do desejado, à saída da porta *Error_Sign*, obtém-se o valor lógico "1", caso contrário obtém-se o valor "0". Feita a comparação é feito um *reset* ao contador e a contagem é reiniciada na próxima linha. A saída *Frame_Rate_Ok*, no modo *Master*, apenas é ativada quando a condição dada por,

$$|Clk_Cnt - Target_Clk_Cnt| < Tolerance \tag{D.2}$$

for verdadeira. Com Clk_Cnt o valor da contagem do período da linha numa *frame* proveniente da câmara local em número de ciclos do sinal de relógio; e $Tolerance$ definindo uma tolerância no erro de *line rate* definida internamente na FPGA pelo programador também dada em número de ciclos do sinal de relógio. Ou no modo *Slave*, quando a condição,

$$\begin{aligned} &|Slave_LVAL_Ref - Target_Clk_Cnt - Frame_Diff_Sign| \\ &\quad < Tolerance \;\&\; FVAL_XOR_Ref < 3000 \end{aligned} \tag{D.3}$$

se verificar. Sendo $FVAL_XOR_Ref$ a diferença de fase em módulo entre a *frame* local e a *frame* master dada em número de *clocks*. O valor 3000 foi determinado experimentalmente como sendo o mais adequado às condições de operação sob as quais o sistema foi testado.

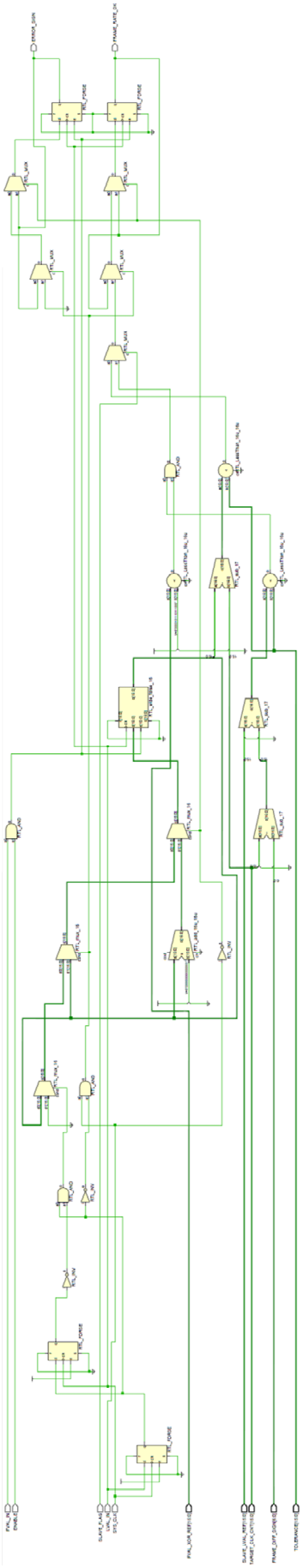


Figura D.10 – Implementação em *hardware* do bloco COMPARATOR.

Anexo E - Implementação em *hardware* dos Sub-blocos do *Illumination Control*

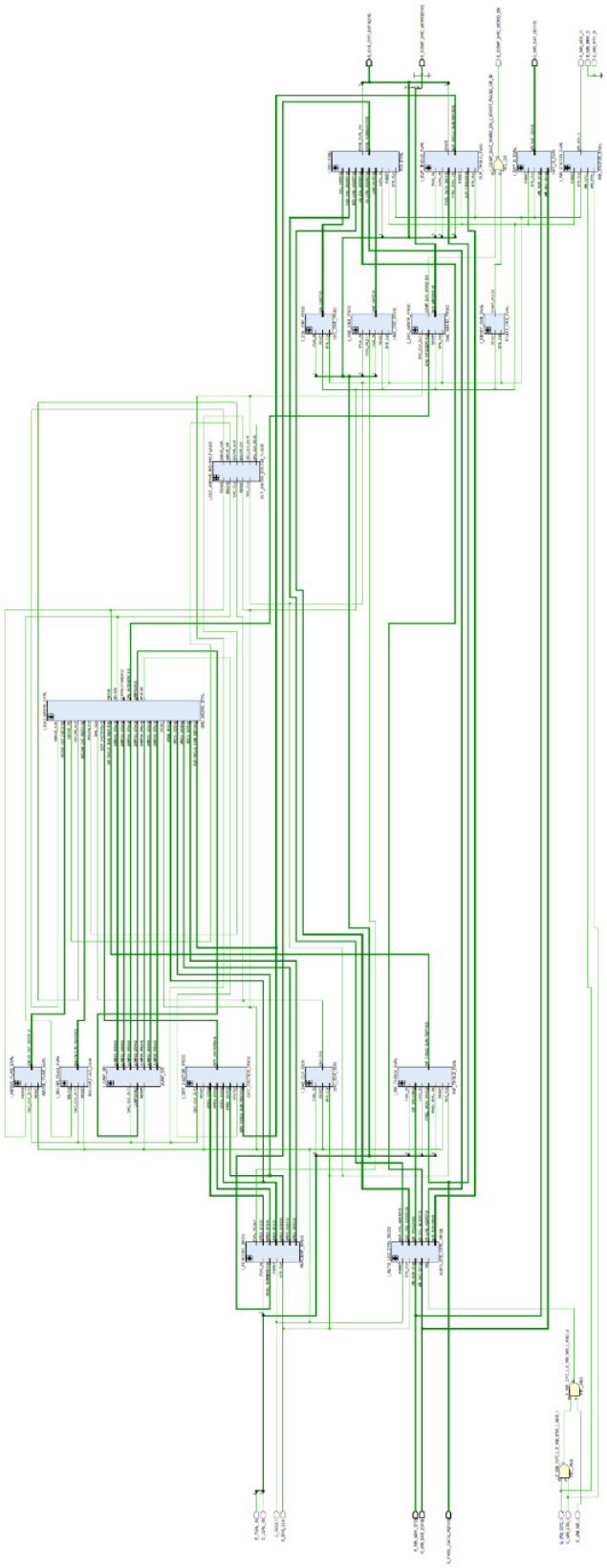


Figura E.1 – Estrutura do módulo *Illumination Control*.

E.1 Start counter evaluation

O bloco *start_cnt_eval* (Fig. E.2), é utilizado para gerar um primeiro impulso de ativação da DAC depois da desativação do sinal de *reset*. Este tem o mesmo princípio de funcionamento que o bloco *start_pulse_gen* integrado no *Frequency Comparator* e já explicado no ponto D.8 do Anexo D. Assim garante-se que após um *reset* existirá uma tensão aplicada no sensor permitindo-lhe enviar dados para a FPGA e possibilitando ao resto do módulo de controlo ler as *frames* e ajustar a iluminação aplicada de acordo.

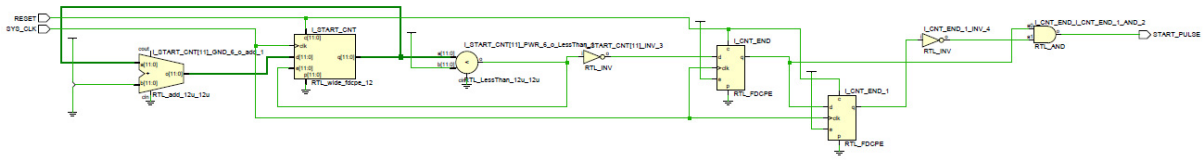


Figura E.2 – Implementação em *hardware* do bloco start_cnt_eval.

E.2 Automatic LED control registers

Este bloco, representado na Fig. E.3, recebe os sinais de controlo, introduzidos pelo utilizador na interface gráfica, a partir do barramento de dados, aqui designado por *wishbone*. Nomeadamente são lidos e interpretados os endereços da ROI (endereços de início e fim de linha e de início e fim de coluna) e os *thresholds* do histograma (nível de referência superior e inferior) e armazenados em registos. Após leitura estes são enviados para a saída.

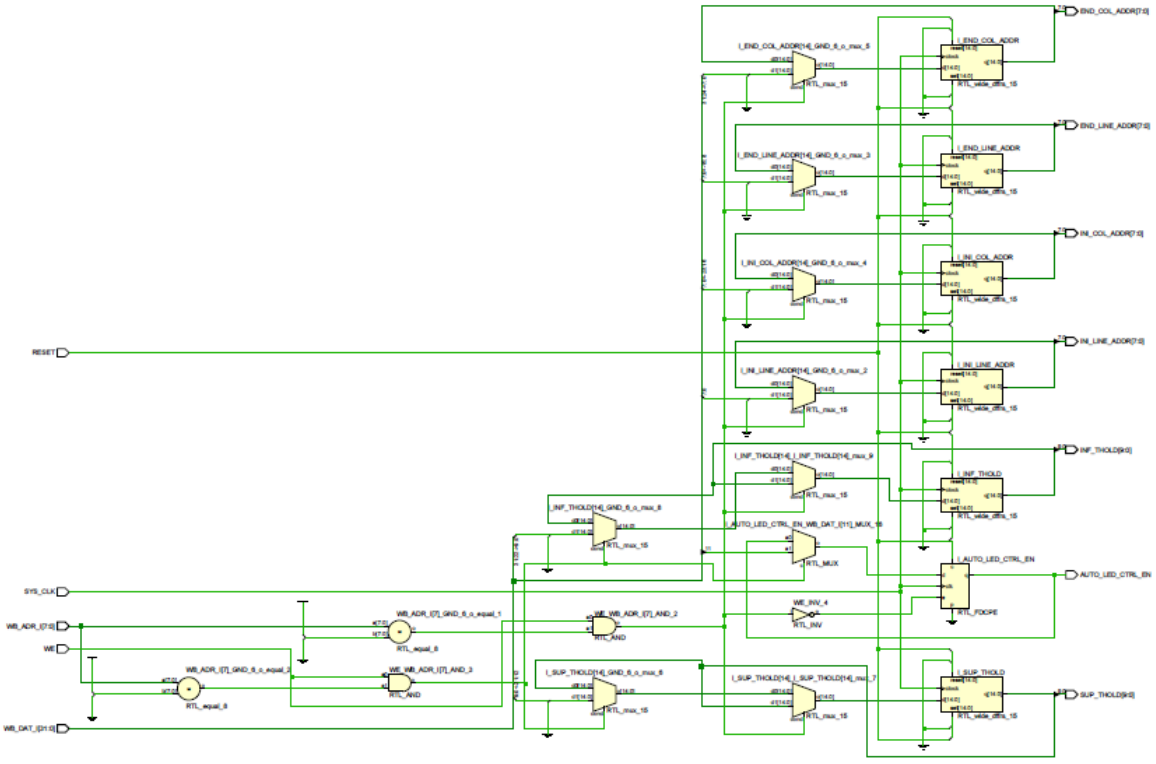


Figura E.3 – Implementação em *hardware* do bloco auto_led_ctrl_regs.

E.3 Line counter & column counter

Com o objetivo de auxiliar a definição da ROI, conforme descrito na secção 4.4.1, são utilizados dois contadores, um de linha e outro de coluna, para percorrerem o índice da *frame* sincronizados pelos sinais FVAL e LVAL. À saída destes obtém-se o valor atual da contagem para cada *frame* a ser lida naquele instante. A sua representação é dada pelas Fig. E.4 e E.5.

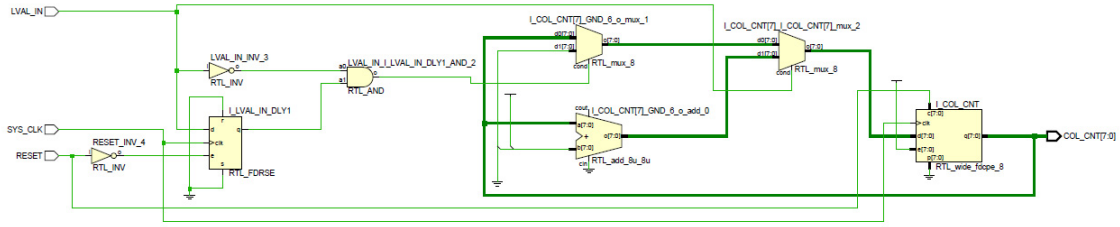


Figura E.4 – Implementação em *hardware* do bloco col_cnt_proc.

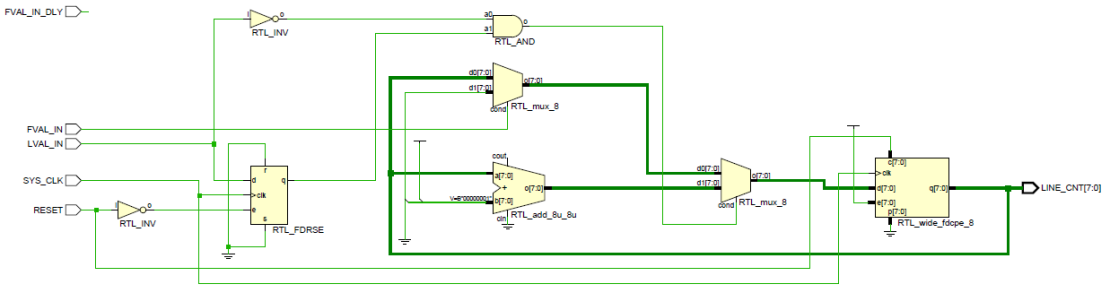


Figura E.5 – Implementação em *hardware* do bloco line_cnt_proc.

E.4 Region of interest evaluation

As saídas dos contadores descritos anteriormente são utilizadas no *ROI Evaluation* (E.6) para definir a região de interesse. É feita uma leitura das contagens de linha e coluna e comparados estes índices com os endereços de linhas e coluna recebidos a partir do *auto_led_ctrl_regs*. Se o índice atual estiver contido dentro dos endereços definidos por,

$$Line_Cnt \geq Ini_Line_Addr \,\&\, Line_Cnt < End_Line_Addr; \qquad \text{(E.1)}$$

$$Col_Cnt \geq Ini_Col_Addr \,\&\, Col_Cnt < End_Col_Addr, \qquad \text{(E.2)}$$

então é ativado o sinal $Pixel_Eval_Ok$, caso contrário este mantém-se a "0". Com $Line_Cnt$ e Col_Cnt os índices atuais da linha e coluna respetivamente, e Ini_Line_Addr, Ini_Col_Addr, End_Line_Addr e End_Col_Addr os endereços de início e fim das linhas e colunas na *frame*.

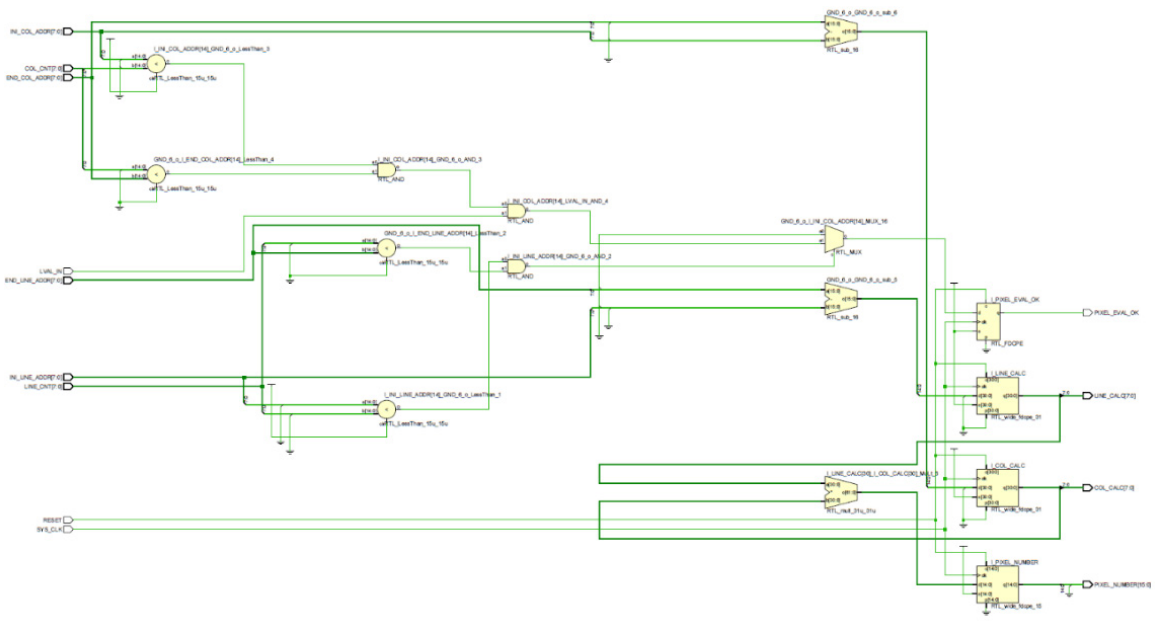


Figura E.6 – Implementação em *hardware* do bloco roi_eval.

E.5 Percentage shift registers

Aqui são utilizados *Shift Registers* (ver Fig. E.7) para definir níveis de percentagem do número total de pixéis contidos na região de interesse (sinal $Pixel_Number$) definida pelo sub-bloco *ROI Evaluation*. Optou-se por esta solução para evitar o uso de um divisor, uma vez que através do deslocamento de bits é possível obter sucessivas divisões por 2 de uma forma simples, poupando recursos. Este procedimento foi já explicado na secção 4.4.2.

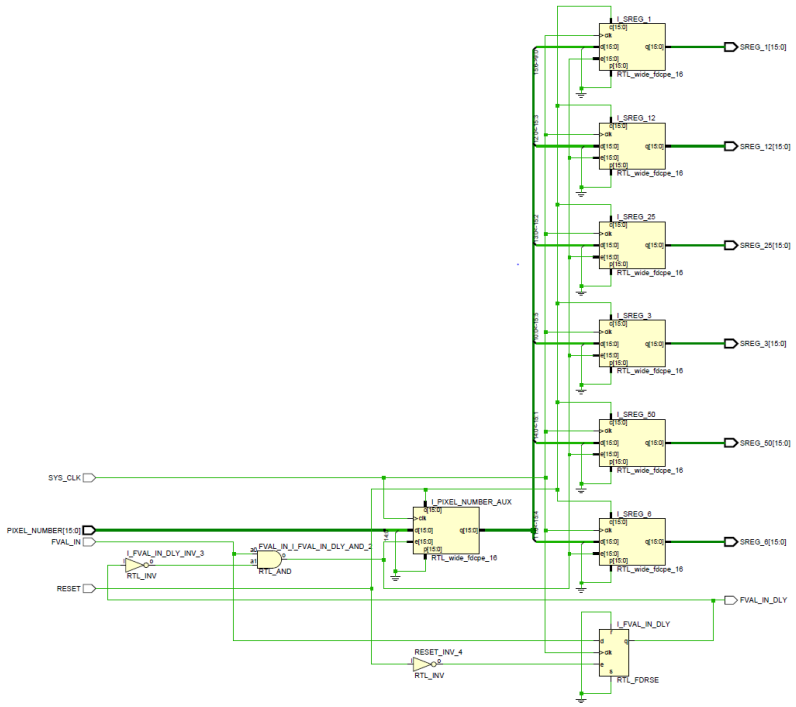


Figura E.7 – Implementação em *hardware* do bloco percent_sreg.

E.6 Superior & inferior threshold evaluation

A ideia inicial era construir um histograma de imagem completo com base no valor dos pixéis recebidos pelo sistema. Contudo, para a aplicação em causa não se justificava a construção do histograma completo tendo em conta que o objetivo era apenas de verificar a presença de sobre ou subexposição à iluminação. Por esse motivo foi construído um histograma rudimentar em que se utilizaram dois níveis de referência, *Superior* e *Inferior Threshold*, lidos do bloco *auto_led_ctrl_regs*.

Em particular, nestes dois blocos (Fig. E.8 e E.9) são utilizados contadores. Estes acumulam um valor cada vez que um pixel lido verifica as condições definidas por,

$$Pixel_Eval_Ok \ = 1 \ \& \ Pixel_Data \geq Sup_Thold \ \& \ LVAL_IN = 1; \qquad \text{(E.3)}$$

$$Pixel_Eval_Ok \ = 1 \ \& \ Pixel_Data \leq Inf_Thold \ \& \ LVAL_IN = 1, \qquad \text{(E.4)}$$

sendo $Pixel_Data$ o valor do píxel lido naquele instante. No final da *frame* o resultado das contagens são colocadas à saída dos blocos para interpretação posterior. O período de tempo que o sinal *Pixel_Eval_Ok* se encontra ativo representa a área definida pelo utilizador como sendo a ROI.

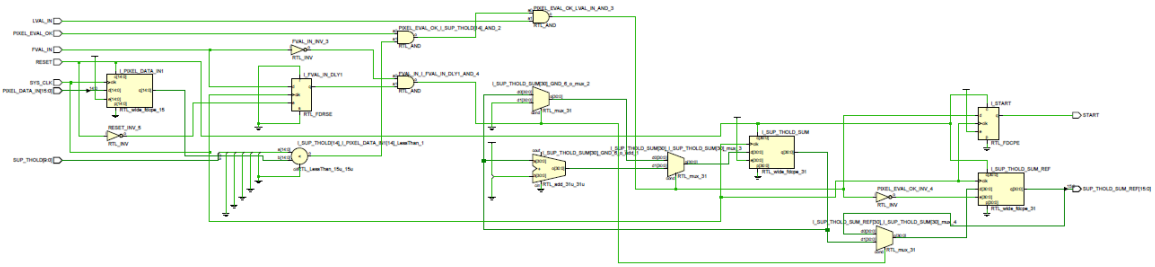


Figura E.8 – Implementação em *hardware* do bloco sup_thold_eval.

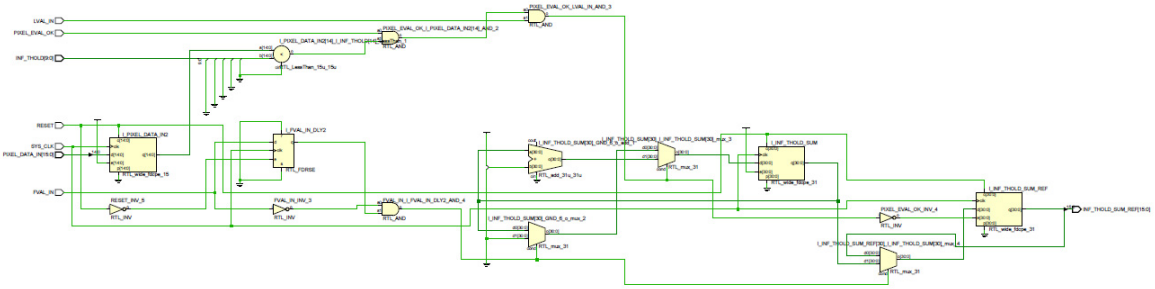


Figura E.9 – Implementação em *hardware* do bloco inf_thold_eval.

E.7 DAC clock generator

Devido ao problema colocado pela integração da imagem no sensor e pelo tempo de avaliação do valor dos pixéis da *frame* no sistema de controlo, foi gerado um sinal de relógio dedicado para sincronizar a avaliação dos pixéis e

geração da palavra de controlo escrita na DAC. Este monitoriza o sinal de período de *frame*, $Fval_In$, e usando como *trigger* o relógio de sistema, Sys_Clk, gera um sinal de relógio DAC_Clk cujo período equivale a 2 períodos de uma *frame*. O sistema digital da Fig. E.10 permite gerar este sinal.

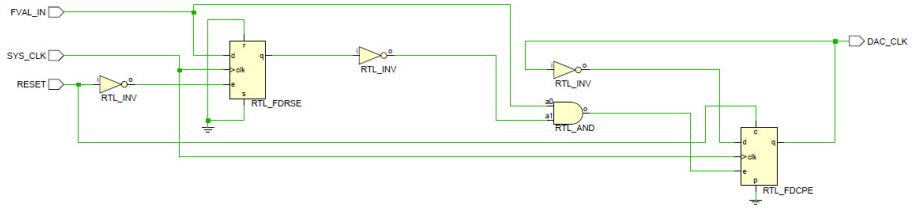


Figura E.10 – Implementação em *hardware* do bloco dac_clk_gen.

E.8 Jump shift registers & reset

A componente integradora do algoritmo é implementada através da soma de 8 variáveis, C_X,

$$Integral\ Correction = \sum_{X=1}^{8} C_X, \tag{E.5}$$

com x um índice de 1 a 8. No entanto, para obter um efeito integrativo, é necessário realizar o deslocamento das variáveis ao longo do tempo através de um *Shift Register* tal como exemplificado na Fig. E.11[15].

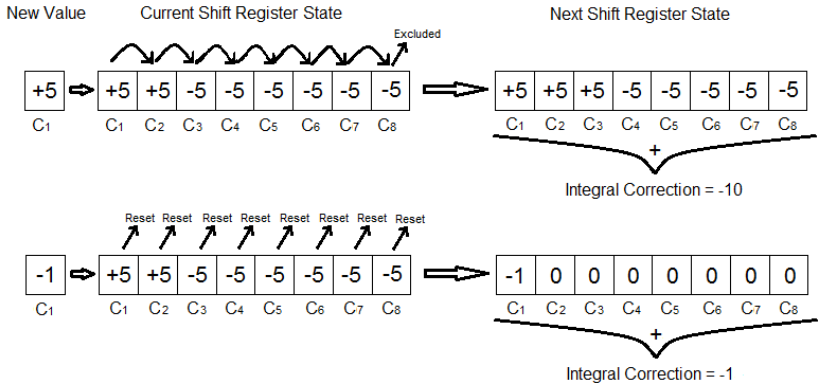


Figura E.11 – Correction Shift Register com Reset.

Adicionalmente, é realizado o *reset* destas variáveis quando o novo valor à entrada do *Shift Register* é,

$$C_1 = 1\ OR\ C_1 = -1\ OR\ C_1 = 0, \tag{E.6}$$

permitindo melhorar a velocidade de estabilização do sistema tal como explicado na secção 4.4.2. A implementação desta função resulta no sistema da Fig. E.12.

[15] Os valores +5 e -5 foram selecionados como um exemplo e podem não representar os valores efetivamente utilizados no algoritmo.

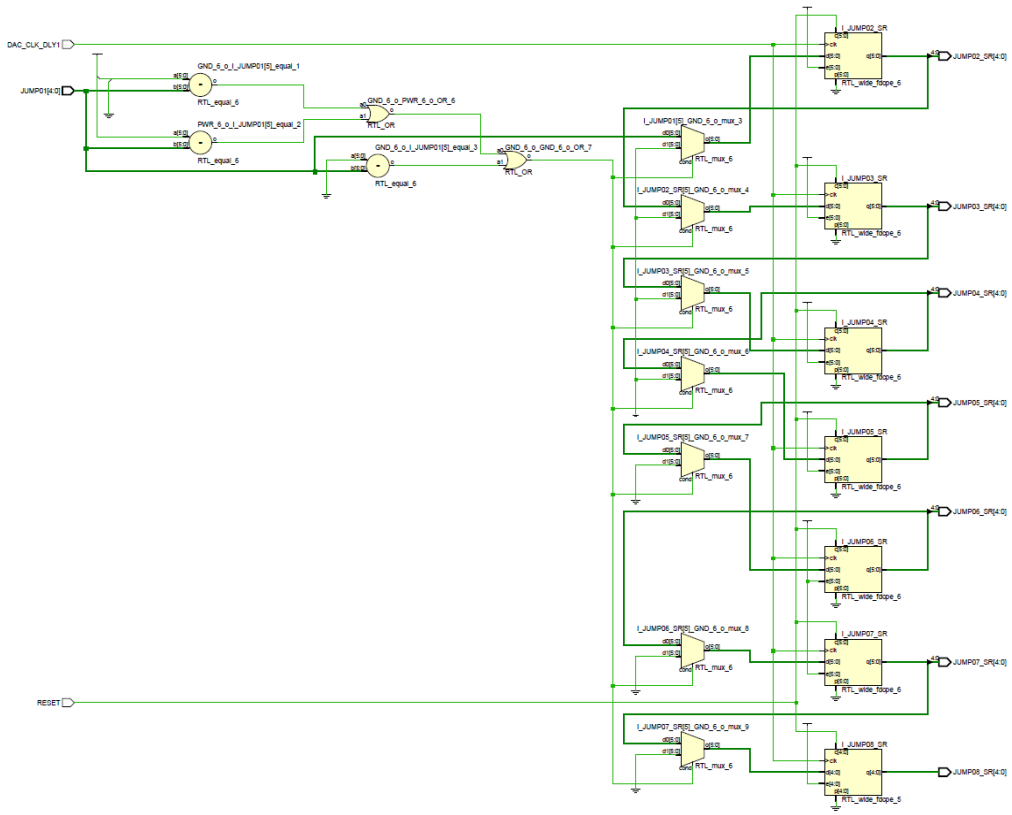


Figura E.12 – Implementação em *hardware* do bloco jump_sr.

E.9 Above & below flag evaluation

Os blocos, representados pelas Fig. E.13 e E.14, permitem determinar o período de tempo no qual o sinal de Above ou Below se encontra ativo. Para tal recorreu-se a contadores que acumulam valores no período ativo destes sinais. À saída é colocado o valor desta contagem. O propósito da avaliação deste tempo de contagem relaciona-se com a necessidade de adicionar um mecanismo no algoritmo de controlo da palavra escrita na DAC para evitar que o valor escrito oscile em demasia em torno do valor ótimo de iluminação. Por exemplo, se o período de tempo for muito curto, significa que estamos na presença de transições sucessivas do valor ótimo. Neste caso, pretende-se então que o sistema faça pequenas correções. Se o período for longo significa que houve apenas um cruzamento do valor ótimo, logo o sistema pode realizar correções ligeiramente mais elevadas.

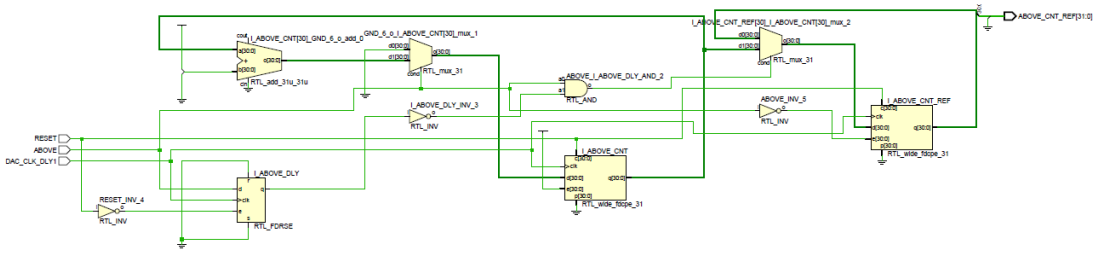


Figura E.13 – Implementação em *hardware* do bloco above_flag_eval.

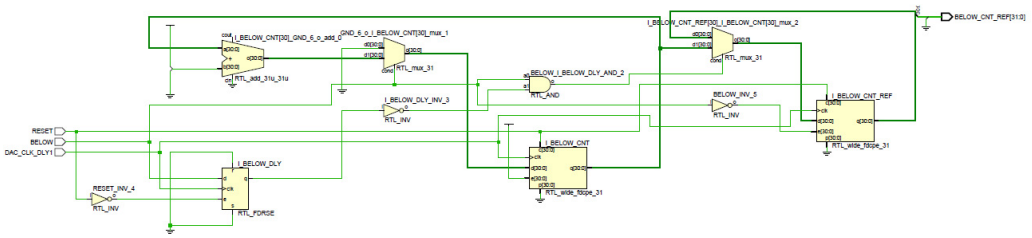


Figura E.14 – Implementação em *hardware* do bloco below_flag_eval.

E.10 Delay above & below flags

No sistema digital da Fig. E.15 é implementado o deslocamento do sinal *Above* e *Below* ao longo do tempo. Isto permite observar o comportamento do erro ao longo do tempo apenas olhando para estes sinais de estado. Estes simplesmente indicam se a correção está a ser feita no sentido descendente ou ascendente. À saída obtém-se os sinais dados por,

$$Above_Aux \ = Above, after\ delay; \tag{E.7}$$

$$Above_OR \ = Above_Aux\ OR\ Above_dly01\ OR\ Above_dly02, \tag{E.8}$$

com $Above_dly01$ e $Above_dly02$ sendo o sinal $Above_Aux$ atrasados por um e dois ciclos de relógio respetivamente.

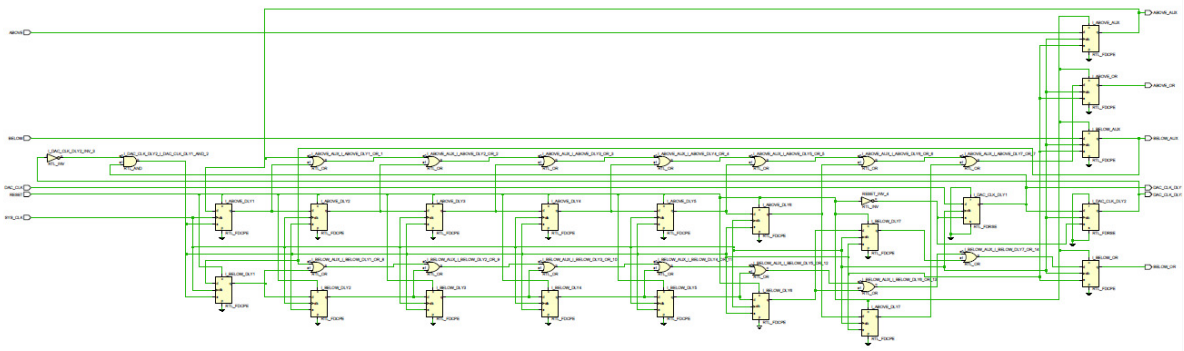


Figura E.15 – Implementação em *hardware* do bloco dly_above_below_flags.

E.11 Wishbone status evaluation

Este sub-bloco (Fig. E.16) encarrega-se de enviar um sinal de *acknowledgment* ao barramento de dados quando o bloco *Illumination Control* recebe os dados provenientes da interface gráfica.

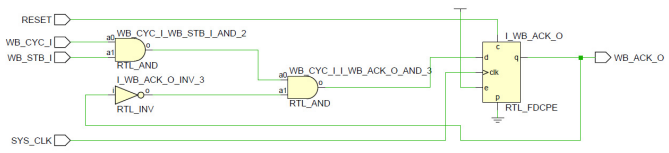


Figura E.16 – Implementação em *hardware* do bloco wb_status_eval.

E.12 Data output evaluation

Devido ao funcionamento da arquitetura do barramento de dados, foi necessário incluir o sub-bloco da Fig. E.17 que basicamente lê os dados do barramento, também lidos pelo bloco *auto_led_ctrl_regs*, e volta a reenvia-los para o barramento. A arquitetura de controlo do barramento posteriormente verifica se os dados recebidos são iguais aos dados enviados pela interface gráfica.

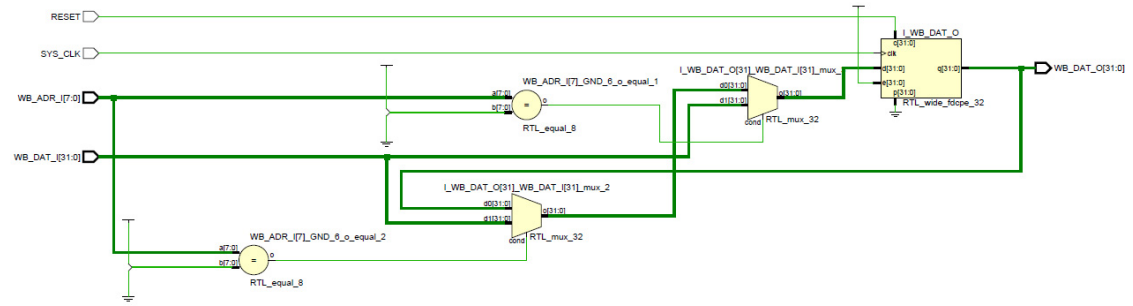


Figura E.17 – Implementação em *hardware* do bloco dat_o_eval.

E.13 Data write

Por fim o bloco *dac_write_proc* (Fig. E.18), é utilizado para escrever a palavra de controlo da DAC na porta de saída sendo sincronizado pelo sinal de relógio *DAC_Clk*. O que permite a escrita na DAC só de dois em dois períodos de *frame*.

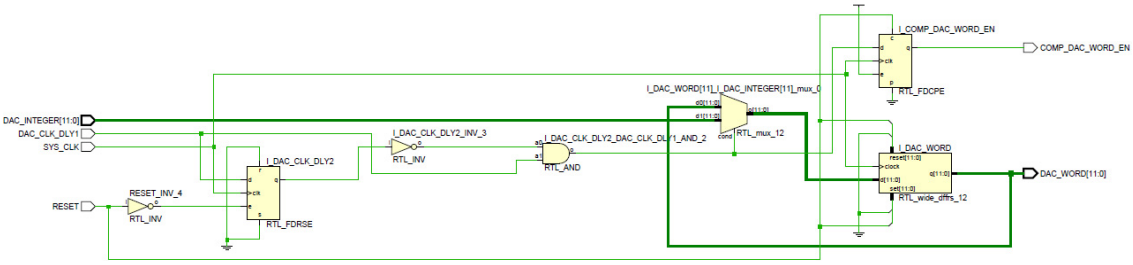


Figura E.18 – Implementação em *hardware* do bloco dac_write_proc.

E.14 Differential factor

Os fatores diferenciais são gerados com base no erro medido em termos de uma diferença de percentagem. Deste modo à entrada do bloco da Fig. E.19 são fornecidos os níveis de percentagem determinados no *percent_sreg* e o número de pixéis acima do nível de referência superior, *Sup_Thold_Sum_Ref*. O sinal de *Status* atua como um *select*, permitindo selecionar entre um fator diferencial mais ou menos elevado.

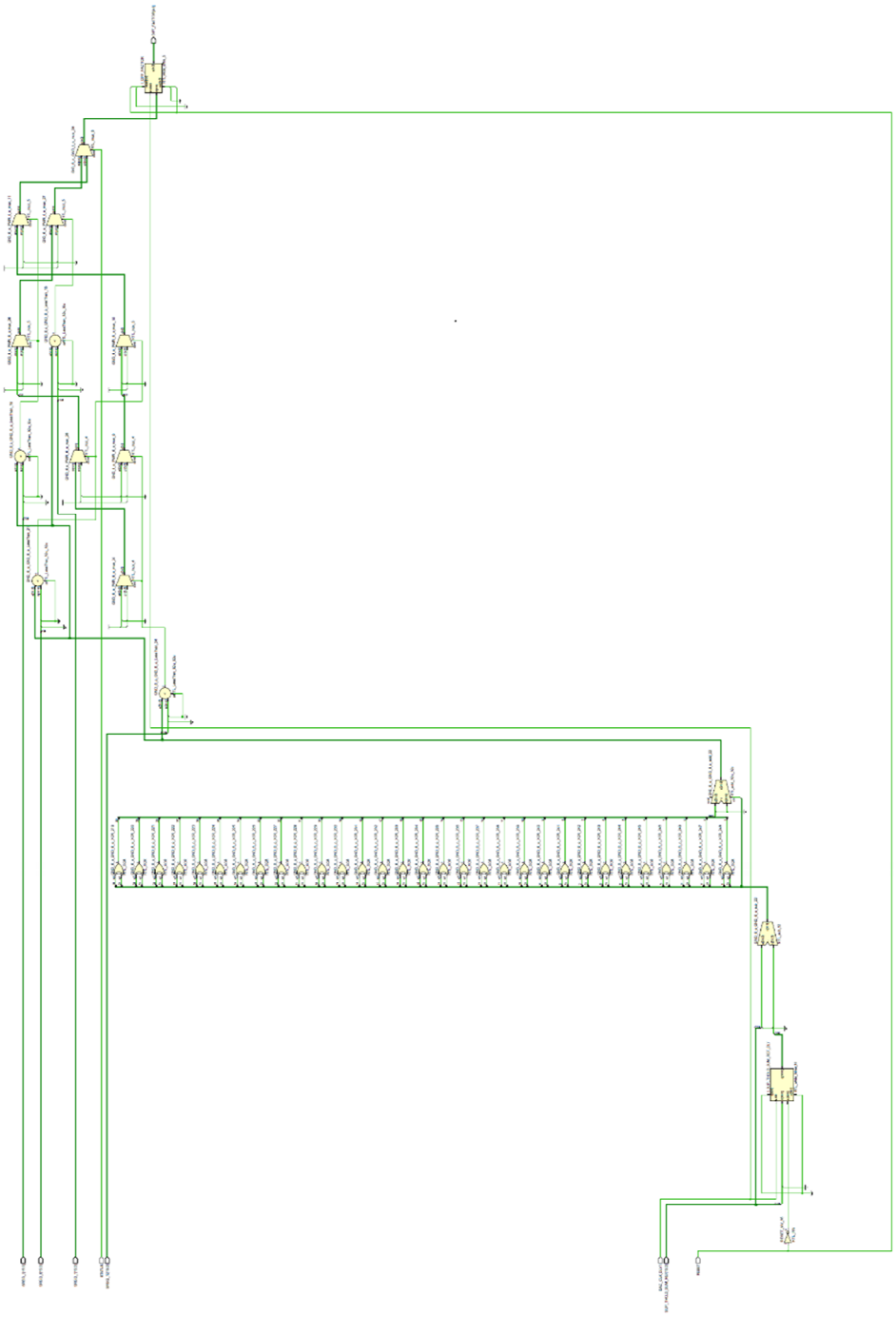


Figura E.19 – Implementação em *hardware* do bloco dif_factor_proc.

E.15 DAC word control

O bloco representado pelas Fig. E.20 e E.21 determina o valor de controlo escrito na DAC de modo a alcançar o valor ótimo de iluminação tendo por base a região de interesse e os *thresholds* superior e inferior. À entrada deste bloco temos os sinais de estado *Above* e *Below*, o sinal de relógio da DAC e o *reset*, o fator diferencial, os *thresholds*, as variáveis usadas no somatório integrativo, e os níveis de percentagem. O funcionamento essencial deste bloco é descrito na secção 4.4.2 com o auxílio do fluxograma da Fig. 4.17.

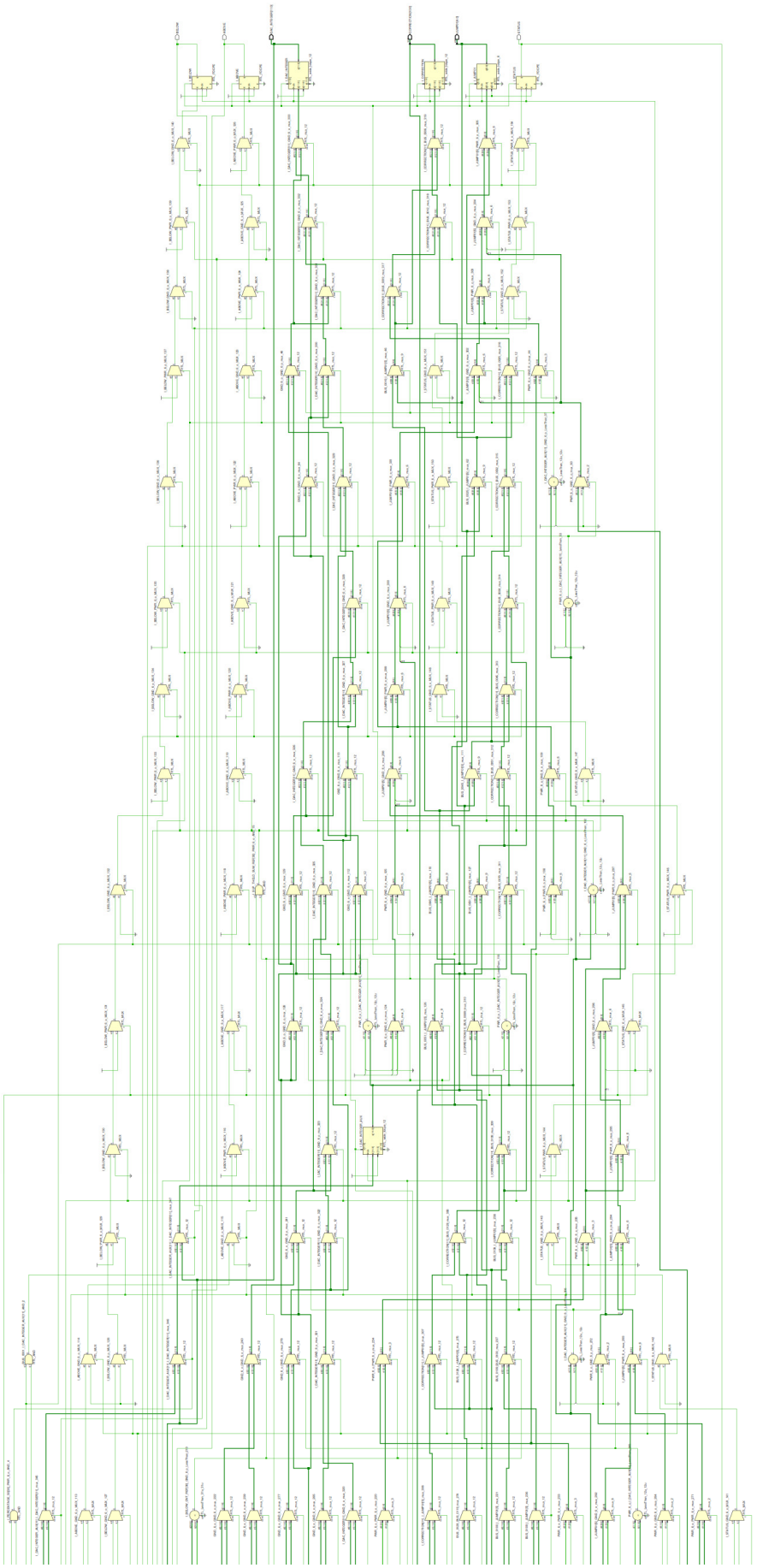


Figura E.20 – Implementação em *hardware* do bloco dac_word_ctrl (parte 1).

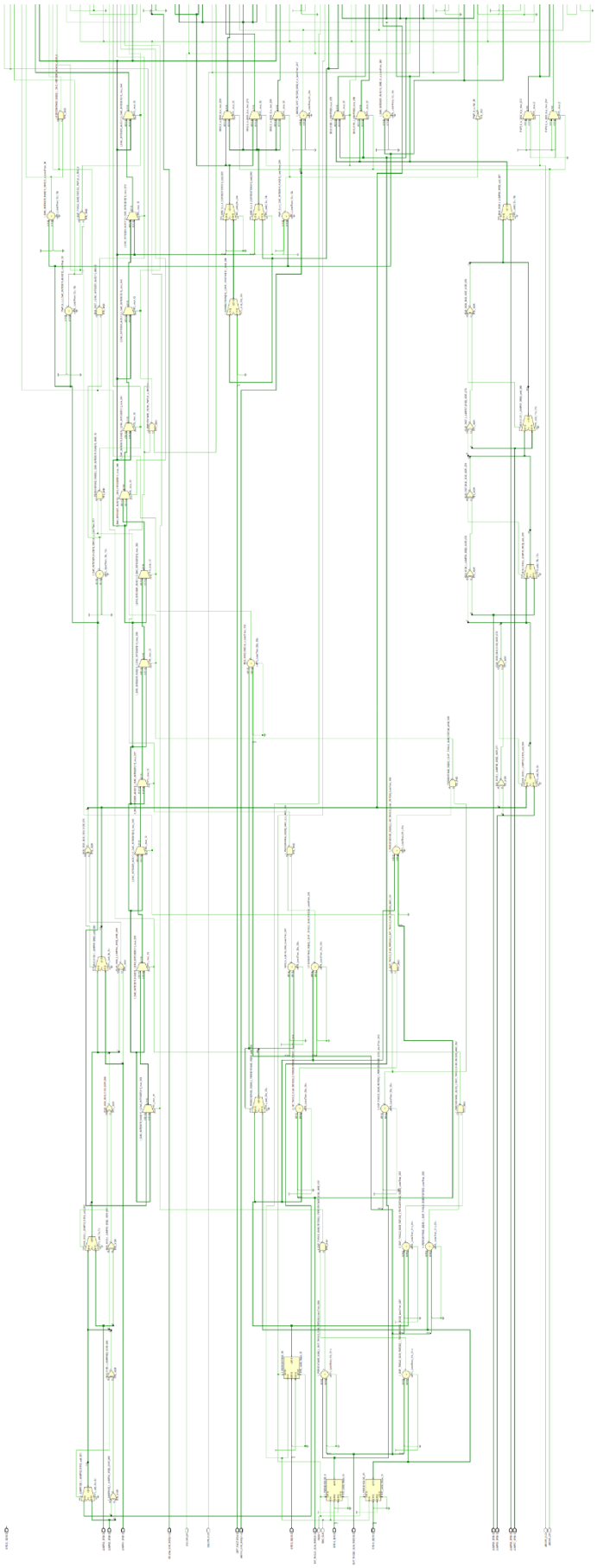


Figura E.21 – Implementação em *hardware* do bloco dac_word_ctrl (parte 2).

Anexo F – Teste de controlo de iluminação

Adicionalmente, para simular o uso da Câmara + AutoLED no interior de um espaço com pouca iluminação, testou-se o sistema colocando a câmara a percorrer um local restrito tal como o interior de uma mão fechada observando o comportamento do ajuste à medida que as condições de iluminação variavam (ver Fig. F.1 a) a f)).

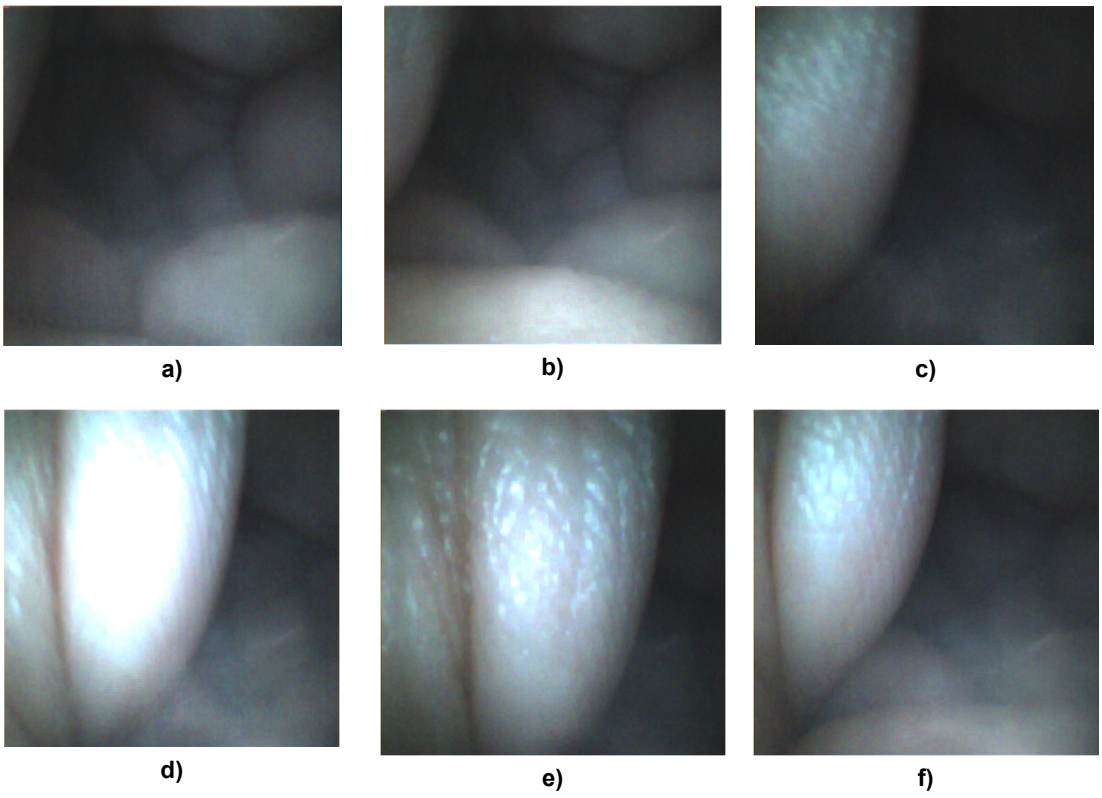


Figura F.1 – Ajuste da iluminação para o nível ótimo.

Verificou-se um ajuste rápido (tempo máximo entre 0,5 s a 1 s) e estável (erro entre 1% e 3%) da intensidade de luz emitida pelos LEDs. Nas figuras é possível observar uma variação elevada da iluminação. Inicialmente, nas situações a) e b) a câmara foi colocada no interior da mão fechada direcionada para o interior do punho. Nesta circunstância, a iluminação emitida pelos LEDs encontra-se no máximo, contudo a imagem continuou a ser algo escura devido à baixa distância focal dos sensores de imagem NanEye [15]. Em c) começou-se a aproximar a câmara a um dos lados do punho direcionado o sensor para a pele. Como a intensidade de luz emitida era máxima, resulta que em d) a imagem se encontrou sobre iluminada. Imediatamente, o algoritmo de controlo começou a ajustar a intensidade da fonte para reencontrar o nível ótimo (neste caso diminuindo a corrente nos LEDs), resultando em e) e f).