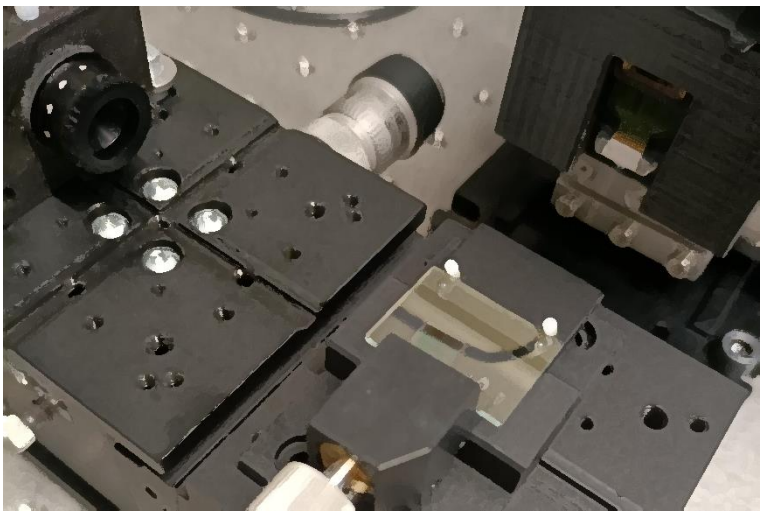
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Eletrónica Telecomunicações e Computadores



Desenvolvimento de um detetor ótico multicanal para circuitos integrados fotónicos planares

PAULO ANDRÉ LOURENÇO SOARES
(Licenciado)

Trabalho de Projeto para obtenção do grau de Mestre
em Engenharia Eletrónica e Telecomunicações

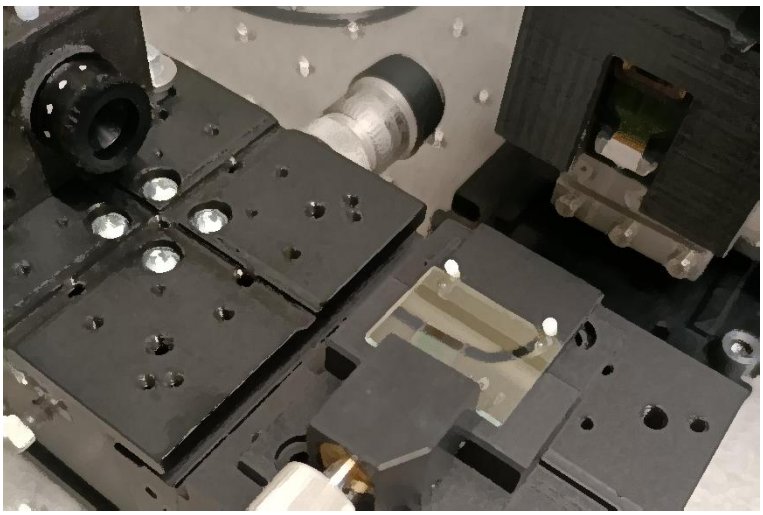
Orientadores:
Professor Alessandro Fantoni
Professor Miguel Fernandes

Júri:
Presidente: Professor Nuno Cota
Vogais:
Professora Paula Louro
Professor Alessandro Fantoni

Dezembro de 2021

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Eletrónica Telecomunicações e Computadores



Desenvolvimento de um detetor ótico multicanal para circuitos integrados fotónicos planares

PAULO ANDRÉ LOURENÇO SOARES
(Licenciado)

Trabalho de Projeto para obtenção do grau de Mestre
em Engenharia Eletrónica e Telecomunicações

Orientadores:
Professor Alessandro Fantoni
Professor Miguel Fernandes

Júri:
Presidente: Professor Nuno Cota
Vogais:
Professora Paula Louro
Professor Alessandro Fantoni

Dezembro de 2021

Resumo

A tecnologia de fotónica integrada em silício, que tem vindo a ser explorada para diversos tipos de finalidades incluindo as aplicações médicas, apresenta-se como uma solução aliciante para a conceção de sistemas de diagnóstico portáteis. Nos últimos anos têm sido demonstrados sensores bioquímicos de elevada sensibilidade com base nesta tecnologia, sendo frequente a utilização de agregados de estruturas interferométricas para uma leitura otimizada ou para a análise de várias patologias. As melhorias que têm sido alcançadas no fabrico de guias de onda em silício amorfo, assim como a sua maior acessibilidade face ao silício cristalino, motivaram a escolha deste material no projeto PhotoAKI que pretende a conceção de um dispositivo de diagnóstico para insuficiência renal aguda de baixo custo, baseado num agregado de interferómetros plasmónicos. O foco deste trabalho é o desenvolvimento de um módulo de aquisição de sinal com a versatilidade necessária para a sua adaptação a estruturas fotónicas com múltiplos guias de onda, como o dispositivo deste projeto. O sistema foi baseado num sensor de imagem linear com funcionamento no espetro infravermelho, podendo os canais de deteção ser configurados de forma flexível, mapeando e combinando, se necessário, os píxeis do sensor de imagem. A gestão do módulo, que inclui um circuito de acondicionamento programável com ajuste de ganho e de nível, um conversor analógico-digital e um controlador para uma fonte emissora, é feita por um microcontrolador da família STM32. A comunicação com um computador é feita através de uma ligação USB utilizando o padrão SCPI, tendo sido desenvolvida uma aplicação para configurar o sistema e fazer a leitura e armazenamento dos dados.

Palavras-chave: PhotoAKI, fotónica, silício amorfo, PECVD, biossensores, aquisição de sinal, sensor de imagem, microcontrolador, SCPI, Aplicação gráfica, Windows Forms.

Abstract

Silicon integrated photonics technology, which has been developed for various applications including medical, stands up as an attractive solution for the conception of portable diagnosis systems. In the past few years, biochemical sensors of high sensitivity have been demonstrated based on this technology, often employing arrays of interferometric structures for either advanced read-out strategies or the analysis of multiple pathologies from the same sample. Developments on the production of amorphous silicon waveguides, as well as it's accessibility, motivated the choice of this material for the conception of a low-cost diagnosis device for acute kidney insufficiency, under the PhotoAKI project, based on an array of plasmonic interferometers. The present work focuses on the development of a signal acquisition module able to adapt to different photonic structures with multiple output waveguides, aiming to be used on this project. This system was based on an infrared linear image sensor, allowing a flexible mapping of the pixels into an arbitrary number of detection channels. The module, which includes a signal conditioning circuit with gain and level adjustment, a high-speed analog-to-digital converter and a light source controller, is managed by an STM32 family microcontroller. It communicates with a computer through USB using the SCPI standard. A graphical application has been developed to configure the device and acquire and represent data.

Keywords: PhotoAKI, photonics, amorphous silicon, PECVD, biosensors, signal acquisition, image sensor, microcontroller, SCPI, GUI Application, Windows Forms.

Agradecimentos

Ao Professor Alessandro Fantoni, por todo o apoio e incentivo para descobrir mais sobre o ramo da fotónica e pela orientação incansável na realização deste trabalho.

Ao Professor Miguel Fernandes por todo o apoio e orientação nos ensaios em laboratório e no desenvolvimento do módulo de aquisição de sinal.

À minha namorada Cristiana Cardoso e à minha avó Catarina Lourenço por todo o suporte, amor e carinho que fez de mim a pessoa que sou hoje.

Ao Professor Rui Melicio pelas palavras simpáticas de incentivo e pela apreciação deste trabalho.

Índice

Índice de Figuras

Lista de siglas e acrónimos

ADC – Analog-Digital Converter
AHB – Advanced High-Performance Bus
APB – Advanced Peripheral Bus
API – Application Programming Interface
ASCII - American Standard Code for Information Interchange
CCD – Charge-Coupled Device
CMOS – Complementary Metal-Oxide-Semiconductor
CMP – Chemical-Mechanical Polishing
COM – Communication port
CPU – Central Processing Unit
CVD – Chemical Vapor Deposition
DAC – Digital-Analog Converter
DFU – Device Firmware Update
DMA – Direct Memory Access
EEPROM – Electrically-Erasable Programmable Read-Only Memory
EOC – End of Conversion
FEDER – Fundo Europeu de Desenvolvimento Regional
FCT – Fundação para a Ciência e a Tecnologia
FSMC – Flexible Static Memory Controller
GPIO – General Purpose Input-Output
HAL – Hardware Abstraction Layer
I2C – Inter-Integrated Circuit
IC – Integrated Circuit
ICP - Inductively Coupled Plasma
IDE – Integrated Development Environment
ISEL – Instituto Superior de Engenharia de Lisboa
IST-ID – Associação do Instituto Superior Técnico para a Investigação e Desenvolvimento
LCD – Liquid Crystal Display
LOD – Limit of Detection
MCU – Micro-Controller Unit
MISO – Master Input Slave Output
MMI – Multi-mode Interferometer
MOSI – Master Output Slave Input
MZI – Mach-Zehnder Interferometer
NMS|FCM - NOVA Medical School - Faculdade de Ciências Médicas
NVIC – Nested Vectorial Interrupt Controller
OC – Output-Compare
OPM – One-Pulse Mode
PCB – Printed Circuit Board
PIC – Photonic Integrated Circuit
POC – Point-of-Care
PECVD – Plasma Enhanced Chemical Vapor Deposition
PGA – Programmable Gain Amplifier
PWM – Pulse Wave Modulation
RCC – Reset Clock Controller
RDF – Radial Distribution Function
RF – Radiofrequency
RIU – Refractive-Index Units
SCB – System Control Block
SCPI – Standard Commands for Programmable Instruments
SMA - SubMiniature version A

SOI – Silicon On Insulator
SPI – Serial Peripheral Interface
SPP – Surface Plasmon Polariton
SPR – Surface Plasmon Resonance
SRAM – Static Random-Access Memory
TE – Transversal Electric
TM – Transverse Magnetic
UART- Universal Asynchronous Receiver-Transmitter
UNINOVA – Instituto de Desenvolvimento de Novas Tecnologias
USB – Universal Serial Bus
VCOM – Virtual Communication Port
VCSEL – Vertical-Cavity Surface-Emitting Laser
WDM – Wavelength Division Multiplexing
YI – Young Interferometer

1. Introdução

O presente trabalho enquadra-se no projeto PhotoAKI, financiado pela Fundação para a Ciência e a Tecnologia e pela Fundo Europeu de Desenvolvimento Regional, com a referência LISBOA-01-0145-FEDER-031311, que tem como objetivo a conceção de um dispositivo portátil para diagnóstico de insuficiência renal aguda baseado em tecnologia de fotónica em silício amorfo. A elevada sensibilidade e rapidez dos biossensores fotónicos demonstrados nos últimos anos, que não necessitam de marcação das substâncias em análise como nos métodos laboratoriais convencionais, juntamente com os custos de produção inferiores do silício amorfo e o seu potencial para o fabrico de dispositivos miniaturizados, motivaram o estudo e a prototipagem de um dispositivo nesta tecnologia. Os circuitos fotónicos irão incluir uma grelha de acoplamento, um acoplador de interferência multimodo (MMI) e uma janela de deteção, que tem por base um agregado de interferómetros plasmónicos e cuja saída é constituída por múltiplos guias de onda.

O foco principal deste trabalho é o desenvolvimento de um módulo de aquisição de sinal que possa fazer a leitura do conjunto de sinais óticos resultantes desta janela de deteção, independentemente da quantidade de canais do dispositivo que seja produzido. A solução para este subsistema foi baseada num sensor de imagem linear CMOS com 128 píxeis. A Figura 1 ilustra os componentes do sistema completo onde o módulo de aquisição de sinal é enquadrado.

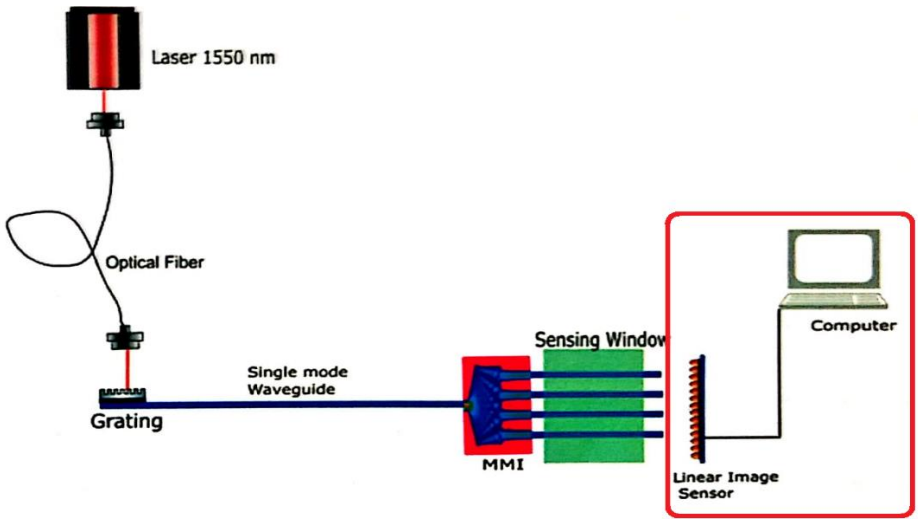


Figura 1 – Esquema dos componentes do projeto no qual se enquadra o dispositivo a desenvolver.

Neste âmbito foi produzido um módulo de aquisição, concretizado numa PCB compatível com a placa de desenvolvimento de um microcontrolador da família STM32. O sensor de imagem é montado na PCB, que inclui um circuito de acondicionamento programável e um sinal de controlo para o laser, entre outros circuitos auxiliares. Foi

desenvolvido o *firmware* para o microcontrolador que comunica com um computador através da porta USB segundo uma implementação do protocolo SCPI, disponibilizando todas as funcionalidades de configuração e de aquisição num porto série virtual (VCOM). Uma aplicação gráfica foi também desenvolvida para uma gestão fácil e intuitiva do módulo, com várias opções de representação gráfica dos dados adquiridos.

Estrutura do documento

O documento começa com uma breve revisão da tecnologia de fotónica em silício e dos biossensores atuais baseados nesta tecnologia, tendo em vista as aplicações na área do diagnóstico médico num paradigma POC (*point-of-care*).

De seguida é documentado o detetor ótico multicanal, começando por estabelecer os requisitos do sistema e as ferramentas e métodos utilizados. As funcionalidades do protótipo concebido são apresentadas, e depois é feita uma descrição do *firmware* desenvolvido e da PCB. A aplicação gráfica para PC será também documentada.

Por fim são apresentados alguns resultados laboratoriais de experiências realizadas com o módulo de aquisição.

2. Estado da Arte

2.1. Fotónica em silício

Uma das grandes motivações para a investigação dos circuitos de fotónica em silício foi a existência de processos de fabrico bem desenvolvidos para este material, resultantes do sucesso da indústria da eletrónica integrada. A utilização dos processos CMOS existentes iria reduzir significativamente os custos para a produção de circuitos de fotónica em larga escala. A disponibilidade de bolachas de SOI (*silicon on insulator* ou silício em isolante) de boa qualidade, havendo um contraste elevado entre os índices de refracção do silício (n ~3,45) e do óxido de silício (n ~ 1,45), permitiriam também realizar guias de onda com dimensões à escala do sub-micrómetro. Dado que a área dos dispositivos tem um impacto significativo no custo, estas dimensões tornariam os circuitos de fotónica economicamente viáveis com estes processos, sendo esta uma segunda motivação. [1]

O espectro de absorção do silício permite realizar guias de onda com baixas perdas na região do infravermelho. No entanto, as pequenas dimensões pretendidas na fotónica integrada fazem com que as imperfeições nas paredes dos guias sejam um obstáculo à propagação devido às perdas por espalhamento. Por consequência, a produção destes circuitos implica o recurso a processos que suavizem as irregularidades nas superfícies. Um exemplo é oxidação térmica, que permite a suavização das paredes dos guias de onda sem deformar o seu perfil. Sujeitando as bolachas a temperaturas elevadas na ordem dos 1100 °C, conseguem-se perdas entre 0,1 dB/cm e 3 dB/cm dependendo das dimensões dos guias [1]. Note-se que este procedimento não é seletivo e pode danificar outros componentes. Outro exemplo mais recente e com maior seletividade é a fotolitografia com recurso a *excimer laser* de XeCl, através da qual foram obtidos guias de onda com irregularidades superficiais entre 13 a 3 nm. [2]

A fotónica de silício tem uma aplicação bastante forte no sector das telecomunicações, moldando de forma significativa da sua evolução. Os comprimentos de onda mais favoráveis para a transmissão através de fibra ótica encontram-se em torno dos ~1,3 µm e dos ~1,55 µm, tendo uma grande parte da investigação incidido sobre dispositivos que operam nesses comprimentos, apesar da fotónica integrada em silício ser viável em comprimentos entre os 1,1 µm e os 7 µm. Dispositivos desde simples guias de onda, a acopladores, até filtros em grelhas de difração ou estruturas em anel ressonante, foram amplamente estudados, assim como moduladores, comutadores e fotodetetores. Atualmente estes são alguns dos blocos de construção de circuitos integrados de fotónica. No entanto, amplificadores e fontes emissoras de luz baseadas em silício têm

sido objeto de pesquisa nos últimos anos e, apesar de haver alguns progressos no sentido de tornar estes componentes completamente integráveis em silício [3], ainda pode ser preferível recorrer a componentes discretos em outros materiais para completar os sistemas. Redes de comunicação de elevada eficiência foram construídas na última década desta forma, assim como variados dispositivos com outras aplicações.

Além da comunicação a grandes distâncias, a fotónica de silício também possibilita comunicações com débitos elevados entre chips e até dentro do mesmo chip. À medida que as dimensões dos processadores diminuem e as suas frequências de funcionamento aumentam, os efeitos parasitas das interligações metálicas colocam limites ao seu desempenho devido à dissipação de calor e à atenuação. A utilização de interligações óticas é uma hipótese para a progressão da tecnologia, possibilitando circuitos mais estáveis, pois estas não estão sujeitas a interferências eletromagnéticas, e débitos binários mais elevados, utilizando portadoras de frequência elevada e multiplexagem por divisão em comprimento de onda (WDM). Também seria importante que a sua implementação seja realizada de forma a dissipar menos calor, motivando pesquisas para reduzir o consumo energético dos circuitos de fotónica. [2], [4]

2.2. Silício amorfo

O silício amorfo é largamente utilizado na produção de monitores de cristal líquido (LCD) e painéis fotovoltaicos. Nos últimos anos tem sido explorado no ramo da fotónica pelas suas propriedades óticas e por haver métodos relativamente económicos de o obter e trabalhar, comparativamente com o silício cristalino. Dado que este pode ser obtido por processos de deposição química, a temperaturas abaixo de 400 °C, não só é possível sobrepô-lo a materiais que não suportam temperaturas elevadas como também permite a sobreposição de vários circuitos de fotónica em camadas.

2.2.1. Características

Ao contrário do silício cristalino que apresenta uma estrutura com um padrão que se repete de forma periódica, e que por isso as suas características podem ser estudadas a partir de uma célula básica, o silício amorfo não possui essa periocidade. As distâncias e os ângulos de ligação entre os átomos variam ao longo da estrutura. O seu estudo através de técnicas de difração permitiu obter medidas da densidade de distribuição radial (RDF, ou *radial distribution function*), cujas características são esquematizadas na Figura 2.

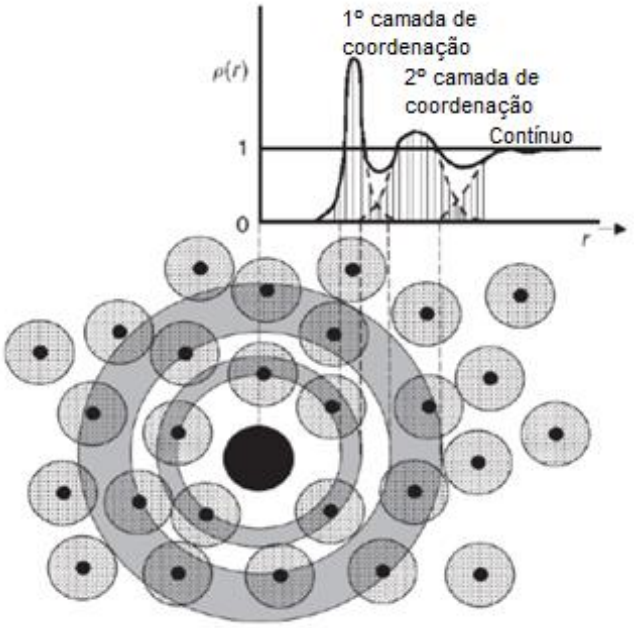


Figura 2 – Esquema ilustrativo da origem das características da função RDF, adaptado de [5].

Esta função é definida pelo número de átomos que distam entre *r e r+dr* de um outro átomo. A área por baixo dos picos no gráfico define o número de coordenação efetivo, que difere da regra de 8-N na qual N é o número de electrões de valência. Isto significa que nestas circunstâncias um átomo de silício pode ficar relativamente estável no reticulo estabelecendo não quatro ligações covalentes, mas, por exemplo, duas ou três.

Na Figura 3 pode-se verificar que os dois primeiros picos de distribuição estão centrados nas distâncias correspondentes às ligações do silício cristalino, mas, a partir do terceiro pico, as distribuições deixam de corresponder, mostrando que existe uma ordem de curto alcance e que a estrutura não é completamente aleatória.

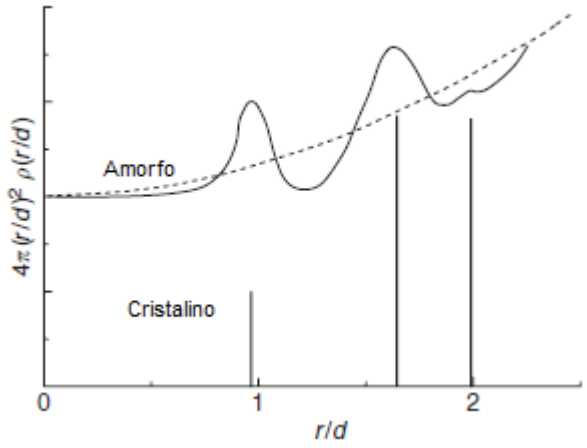


Figura 3 – RDF de silício amorfo e silício cristalino. Adaptado de [5].

As distâncias de ligação de comprimento variável causam o surgimento de ligações covalentes fracas ou tensas. Na teoria das bandas de energia, estas manifestam-se no

prolongamento das bandas de valência e de condução para dentro da banda proibida (Figura 4). Pode-se dizer que quanto maior for o grau de desordem, maior é o prolongamento das bandas de energia comparativamente às bandas bem definidas do semicondutor cristalino. Existem também muitas ligações incompletas (defeitos). Estas causam o aprisionamento de portadores de carga (electrões ou lacunas, conforme o defeito tenha uma natureza do tipo recetor, ou doador, respectivamente), reduzem a fotocondutividade do material e dificultam a sua dopagem [5]. Os defeitos podem, contudo, ser reduzidos com a introdução de hidrogénio, formando silício amorfo hidrogenado (a-Si:H) .

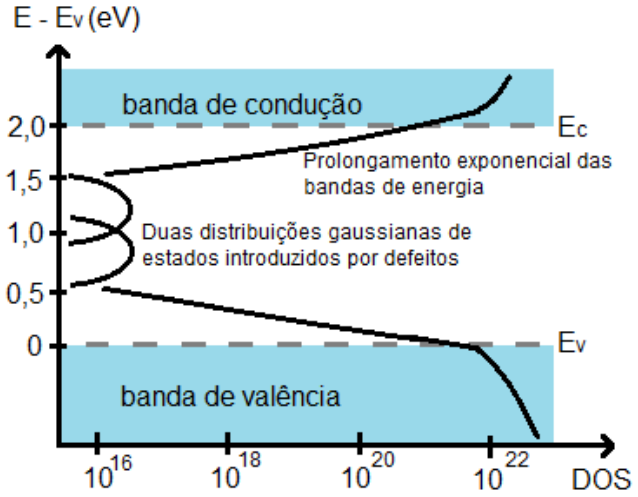


Figura 4 – Esquema das bandas de energia, ilustrando a extensão das bandas de condução e de valência e os estados localizados originados por defeitos.

Dependendo das condições de deposição o material pode apresentar-se em várias fases de desde a fase amorfa (a-Si), passando pelas fases proto-cristalina (pc-Si), nano-cristalina (nc-Si), micro-cristalina (µc-Si), até atingir a fase poli-cristalina (PolySi). O processo de deposição de a-Si tipicamente utilizado para aplicações nas áreas da fotónica e da optoelectrónica é a deposição química em fase de vapor melhorada por plasma (PECVD).

2.2.2. PECVD

O processo de *Chemical Vapor Deposition* (CVD) consiste na deposição de peliculas finas de um material precursor sobre um substrato através de reações químicas em fase de vapor, tipicamente dentro de uma câmara em condições de temperatura e pressão controladas. No processo de PECVD o material precursor é desintegrado pela exposição a plasma, produzido com recurso a energia elétrica, desencadeando a formação de radicais livres. Estes, por sua vez, difundem-se e reagem com o substrato efetuando a sua deposição, formando uma camada. [6]

O sistema típico de PECVD contém uma câmara de reação, onde ocorre a dissociação dos precursores e a deposição do material num substrato aquecido. Normalmente é montada numa configuração com dois elétrodos acoplados capacitivamente, sendo um elétrodo colocado em baixo, alimentado com uma fonte RF (a operar a 13,56 MHz) e o outro em cima, ligado à massa, e onde é colocado o substrato de forma a evitar a deposição de poeiras por efeito da gravidade. O sistema conta também com mecanismos de monitorização e injeção dos gases, assim como de controlo de pressão. A formação de silício amorfo através deste processo utiliza silano (SiH_4) como gás precursor, frequentemente diluído em outros gases como dihidrogénio (H_2) e árgon (Ar), por exemplo. A potência e a frequência RF, a pressão, a temperatura do substrato e a concentração dos gases, determinam as características do semicondutor [7].

Fotografias do sistema existente no ISEL encontram-se na Figura 5. À esquerda pode-se ver a estrutura de suporte da câmara, com tubagens de gás, bomba turbomolecular, caixa de adaptação RF, entre outros sistemas de monitorização e controlo. À direita encontram-se uma fotografia do interior da câmara com elétrodo RF, difusor de gás e suporte aquecido para o substrato, e uma fotografia da janela onde se pode ver a emissão de luz pelos gases ionizados em baixa pressão.

Figura 5 – Sistema de PECVD no ISEL e interior da câmara de reacção.

O hidrogénio presente no processo tem a função importante de completar ligações no silício amorfo, reduzindo a densidade de defeitos que tipicamente é da ordem de 10^{21} cm^{-3} [8]. A concentração de ligações incompletas após a hidrogenação tipicamente chega a atingir os 10^{16} cm^{-3} [5]. Isto é suficiente para melhorar as características de propagação ótica no material, passando a permitir comprimentos de onda na região do infravermelho até perto do espectro visível. O silício amorfo não hidrogenado sofre uma grande absorção nos comprimentos de onda do infravermelho utilizados para telecomunicações. A difusão do hidrogénio ocorre de forma mais eficiente a 400 °C, mas acima dos 250 °C também ocorre a sua libertação, sendo esta uma temperatura ótima para reduzir os defeitos na estrutura segundo o estudo documentado em [4], onde foi obtida uma medida de concentração de defeitos de $4{,}2 \times 10^{15}$ cm^{-3}.

2.2.3. Fabrico dos guias de onda

Tal como no caso do silício cristalino, os guias de onda em silício amorfo à escala do sub-mícron requerem superfícies suaves para reduzir as perdas por espalhamento, tanto nas paredes laterais do guia como nas paredes superiores e inferiores. Para o aplanamento das superfícies podem ser utilizadas técnicas de polimento mecânico-químico (CMP - *chemical mechanical polishing*), se necessário. Este procedimento pode, contudo, provocar maiores perdas por absorção devido à possível reintrodução de defeitos por quebra de ligações de Si-H ou surgimento de ligações de Si-OH [8].

A restante geometria dos dispositivos resulta da remoção do silício através de processos como a fotolitografia e a erosão química, por exemplo. As irregularidades das paredes laterais dos guias dependem da precisão destes processos e da metodologia aplicada, podendo a gravura ser feita em várias etapas. A Figura 6 exemplifica duas geometrias comuns, em *wire* e em *ridge*.

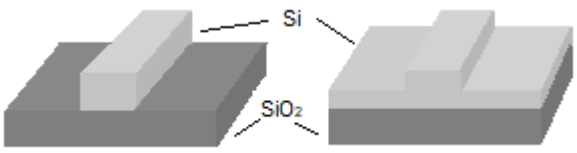


Figura 6 – Geometrias dos guias de onda: à esquerda, wire, à direita, ridge.

Em [9], obtiveram-se perdas de propagação na ordem dos 3,46 dB/cm e dos 1,34 dB/cm (modo *transverse electric* (TE) @ 1550 nm) para um guia de onda em *wire* e outro em *ridge*, respectivamente, com cerca de 220 nm de altura e 480 nm de largura. O processo de fabrico envolveu a deposição de uma camada de óxido de silício (SiO_2) de 2 µm, que foi polida por CMP, seguida da deposição de uma camada de a-Si:H de 220 nm. Os guias de onda foram fabricados através de fotolitografia a 193 nm e erosão por reação

iónica baseada no plasma de acoplamento indutivo (ICP). Nos guias de onda em *ridge* a camada de a-Si:H foi erodida a uma profundidade de 70 nm.

Em [8], foram obtidos guias com 220 nm de altura e 500 nm de largura, com perdas de propagação de 3,2 ± 0,2 dB/cm (modo TE) e 2,3 ± 0,1 dB/cm (modo *transverse magnetic* TM) para um comprimento de onda de 1550 nm. Os passos no processo de fabrico, resumidamente, foram: a deposição por PECVD de uma camada de 2 µm de SiO_2 sobre um substrato de silício, seguida do seu polimento por CMP e da deposição de uma camada de 220 nm de a-Si:H, e de outra camada de 50 nm de SiO_2; o desenho dos padrões dos guias de onda por litografia UV a 248 nm na camada de SiO_2 superficial, aplicando uma máscara foto-resistente e remover o SiO_2 exposto; a remoção da máscara foto-resistente e erosão do a-Si:H até à camada inferior de SiO_2, utilizando a camada superficial de 50 nm de SiO_2 como máscara; e, por fim, a deposição de uma nova camada de 2 µm de SiO_2.

Em [10] foram medidas perdas de propagação de 2,7 ± 0,4 dB/cm (modo TE @ 1560 nm), adicionando uma camada de SiN de 10 nm sobre o guia de a-Si:H. Esta camada foi tratada por acção de plasma para remover ligações de N-H na sua constituição e reduzir o seu índice de refracção de 2,1 para 1,9, reduzindo as perdas de 6,5 ± 0,9 dB/cm obtidas num fabrico anterior. O índice de refracção obtido encontra-se entre os índices do a-Si:H e do SiO_2, reduzindo as perdas por espalhamento nas paredes do guia. Adicionalmente, a camada de SiN previne a difusão de hidrogénio em processos posteriores, preservando a passivação dos defeitos no silício amorfo.

Em [4] foram obtidas perdas de propagação de 0,6 dB/cm no modo TE @ 1550 nm, num guia de onda em *ridge* com uma largura de 780 nm e alturas de 500 nm (no topo) e 400 nm (nas bordas). Para minimizar as irregularidades nas superfícies, aproximando-as das irregularidades típicas do silício cristalino e evitando ao máximo o espalhamento do sinal, o guia de onda foi submetido a um processo de CMP antes da gravura. Depois, optou-se por uma geometria em *ridge* pouco profunda para haver menos irregularidades nas paredes laterais. Desta forma foi obtida a menor perda de propagação em silício amorfo até à data.

Estes resultados mostram uma melhoria significativa do desempenho dos guias de onda de silício amorfo, chegando por vezes a competir com os guias de silício cristalino. No caso destes, em [11] são reportadas perdas de 2,6 e 2 dB/cm para os modos TE e TM, respectivamente, no comprimento de 1319 nm para um guia com geometria *wire*. Em [12] são reportados guias em *ridge* com perdas entre 3,1 dB/cm e 1,3 dB/cm para o modo TE @ 1550 nm. Em [13] são reportados guias para o modo TE @ 1550 nm com

perdas de 0,3 dB/cm através de um método de fabrico que produz guias com paredes extremamente lisas.

Tabela 1 – Resumo das características dos guias de onda.

Material	Geometria	Perdas (dB/cm)	Comprimento de onda (nm)	Modo	Referência
a-Si:H	wire	3,46	1550	TE	[9]
	ridge	1,34	1550	TE	[9]
	wire	3,2 ± 0,2	1550	TE	[8]
	wire	2,3 ± 0,1	1550	TM	[8]
	ridge	2,7 ± 0,4	1560	TE	[10]
	shallow-ridge	0,6	1550	TE	[4]
c-Si	wire	2,6	1319	TE	[11]
	wire	2	1319	TM	[11]
	ridge	1,3 a 3,1	1550	TE	[12]
	shallow-ridge	0,3	1550	TE	[13]

Apesar dos guias em a-Si:H tendencialmente sofrerem de perdas de propagação superiores aos guias em c-Si, as perdas mencionadas não são impeditivas à realização de sistemas óticos com dimensões até a alguns centímetros. Dada a maior facilidade da obtenção de a-Si:H, através de sistemas de PECVD, e a compatibilidade deste processo com diversas soluções em termos de materiais e temperaturas, pode-se tornar vantajosa a opção por este semicondutor para a conceção dos sistemas de análise portáteis.

2.3. Biossensores

Os biossensores óticos constituem uma alternativa sólida às técnicas de teste convencionais por possibilitarem análises rápidas e económicas, sem descurar o seu desempenho em termos de sensibilidade e seletividade [14]. Muitos métodos de teste necessitam da marcação do analito através de compostos ou enzimas para que este possa ser detetado, normalmente num contexto laboratorial e por pessoal qualificado. Entretanto, já foi demonstrada a existência de biossensores que não necessitam desta marcação dos analitos, denominando-se estes por sensores *label-free* (ou “livres de etiquetas”). Este tipo de dispositivo não só permite uma análise em tempo real, como torna mais próxima da realidade a análise direta de pequenas amostras não tratadas de urina, sangue ou outras secreções corporais, feita por qualquer pessoa, ainda que a deteção nestas condições seja um desafio devido à concentração de partículas irrelevantes e à grande variabilidade das amostras [15].

A deteção através do campo evanescente é um dos mecanismos de deteção mais comuns neste tipo de dispositivo. O perfil do modo fundamental que se propaga num guia de onda fica maioritariamente confinado no núcleo, enquanto o resto se propaga no exterior com uma amplitude que decresce exponencialmente. A propagação

caracteriza-se, em parte, pelo índice modal, que depende dos índices de refracção nestes meios e se traduz na velocidade de fase da onda. Pretende-se então que a fase de um sinal varie consoante a constituição de uma amostra colocada junto ao guia de onda. Este fenómeno é tipicamente abordado por um de dois mecanismos (ilustrados na Figura 7) conhecidos por deteção homogénea (ou *bulk sensing)* e deteção superficial (ou *surface sensing)* [16].

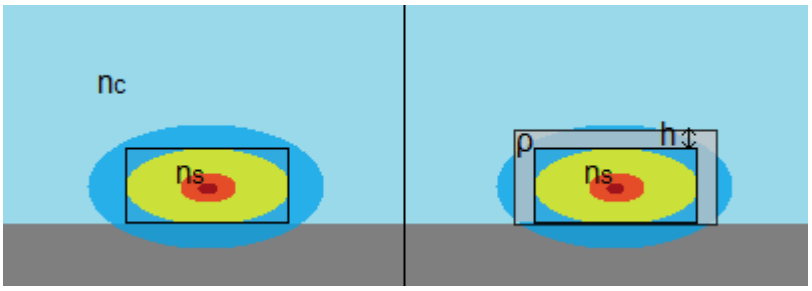


Figura 7 – Ilustração da detecção homogénea (esquerda) e da detecção de superfície (direita) num guia de onda SOI.

Na deteção homogénea, existindo uma correlação entre o índice de refracção da amostra (n_c) e a sua concentração de analito, este índice irá afetar o índice modal (n_s) podendo a sua variação quantificar o analito. Na deteção superficial, procura-se a formação de uma camada de moléculas através da implantação de biorecetores nas paredes exteriores do guia de onda e, como o campo evanescente é mais intenso neste local, o índice modal irá variar consoante a densidade (ρ) e ou a espessura da camada (h). Tipicamente este mecanismo proporciona uma excelente sensibilidade e seletividade do sensor, mas requer um planeamento cuidado dos biorecetores. Uma revisão das estratégias para a biofuncionalização do dispositivo é feita em [14], [17] e [18].

A sensibilidade do sensor que resulta de Δn_s em função de Δn_c, $\Delta\rho$ ou Δh, tem respectivamente as unidades [RIU/RIU], [RIU/nm] e [RIU/(pg/mm^2)] [19], sendo RIU – *Refractive Index Unit*. É também definido o limite de deteção (LOD) que corresponde ao menor Δn_s que pode ser medido e cujo sinal resultante esteja pelo menos três vezes superior ao nível de ruído do sistema [17]. Para isso é necessário um dispositivo que transforme o índice modal numa grandeza ótica que possa ser medida, como a intensidade da luz, a fase ou o comprimento de onda, e um dispositivo que meça essa grandeza.

Diversas estruturas óticas podem funcionar como transdutores, sendo algumas das mais comuns as interferométricas. Tipicamente estes funcionam dividindo um sinal em dois, sendo um sujeito à influência de uma amostra e o outro utilizado como referência, obtendo-se uma medida na saída que depende da diferença de fase entre os dois sinais.

A diferença de fase depende simultaneamente da variação do índice modal e do comprimento da secção de amostragem, pelo que o prolongamento do interferómetro permite obter um maior desfasamento para variações mais pequenas do índice modal, aumentando a sua sensibilidade.

No interferómetro de Mach-Zehnder (MZI) o sinal percorre dois ramos, o que interage com a amostra e o de referência, que depois são acoplados resultando numa interferência mais construtiva ou destrutiva, consoante o ponto de funcionamento e a variação de fase provocada pelo analito. A intensidade na saída apresenta uma variação periódica e sinusoidal em função do desfasamento, pelo que a maior sensibilidade é obtida dimensionando o dispositivo para funcionar na zona de maior declive. A periodicidade pode, contudo, tornar o resultado ambíguo para um grande desfasamento, podendo isto ser solucionado com o seu registo temporal desde o início da análise. A quantificação do analito é realizada a partir da potência ótica na saída, podendo-se recorrer a um simples fotodíodo para esse efeito. Na Figura 8 é ilustrado um MZI.

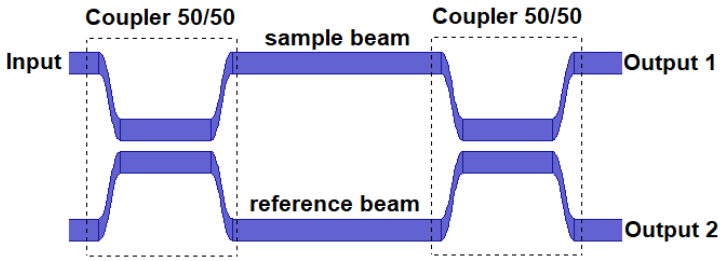


Figura 8 – Ilustração do circuito de fotónica de um MZI.

No interferómetro de Young (YI), ilustrado na Figura 9, o sinal também é separado por dois ramos, mas em vez de estes serem recombinados, os sinais são projetados e capturados por uma câmara CCD (*charge coupled device*) fornecendo uma imagem do padrão de interferência. Este padrão pode depois ser analisado no domínio da Transformada de Fourier e correlacionado com a concentração do analito sem ambiguidade. No entanto a projeção requer um certo distanciamento até à câmara tornando o sinal mais suscetível ao ruído introduzido por vibrações [17].

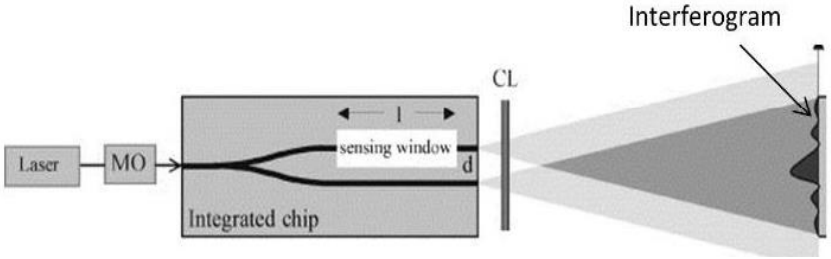


Figura 9 – Esquema de um sistema com um interferómetro de Young, retirado de [17].

Outra possibilidade é o interferómetro de guia de onda bimodal (BiMW), ilustrado na Figura 10. Neste, em vez de o sinal ser dividido por dois ramos, a interferência é obtida permitindo a propagação de dois modos a partir de uma determinada secção no guia de onda. O modo de primeira ordem tem um campo evanescente mais forte do que o modo fundamental, tendo uma maior interação com a amostra [18]. Na presença do analito os índices modais variam em proporções diferentes, resultando numa alteração no padrão de interferência na saída do guia. Tal como no MZI, a saída tem um comportamento periódico [17]. No entanto, tem a vantagem de não necessitar de estruturas em Y ou acopladores.

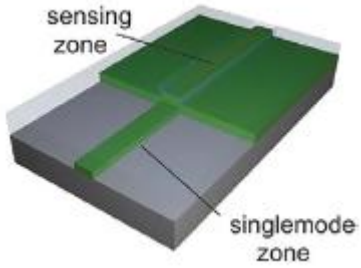


Figura 10 – Interferómetro de guia de onda bimodal. Retirado de [18].

Estruturas de anel ressonante também são utilizadas como transdutores. O acoplamento destas estruturas com guias de onda retos pode ser dimensionado para ter o comportamento de um filtro, deixando passar uma parte estreita do espectro centrada no comprimento de onda de ressonância. Como este é determinado em parte pelo índice modal, a presença do analito provoca então um deslocamento no comprimento de onda visto na saída, podendo ser medido por análise espectral. Alternativamente, entre outras topologias possíveis, o sinal pode ser filtrado novamente por forma a obter um máximo ou um mínimo de amplitude no comprimento de onda de ressonância correspondente à maior concentração do analito e indicar uma medida através da potência ótica. A Figura 11 ilustra uma estrutura em anel ressonante e a Figura 12 as suas características de transferência para as duas saídas.

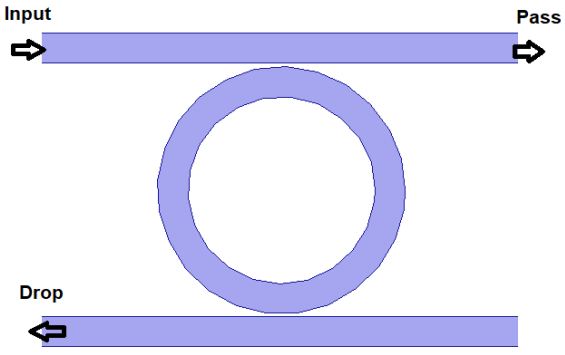


Figura 11 – Ilustração de um anel ressonante acoplado a dois guias de onda retos.

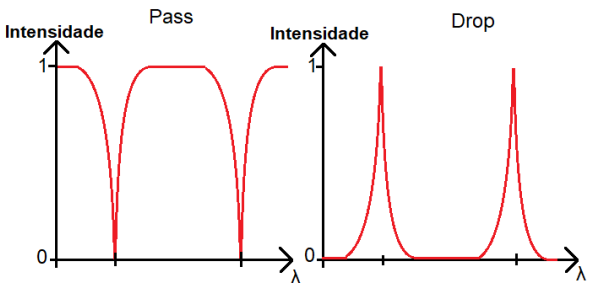


Figura 12 - Esboços da intensidade da luz ao longo do espectro nas duas saídas.

A expressão para o comprimento de onda de ressonância poderá ser dada pela seguinte expressão, em função do índice modal (n_{eff}), raio do anel (r) e um inteiro (m).

$$\lambda_{res} = \frac{n_{eff} 2\pi r}{m}, \qquad m = 1,2,3 \ldots$$

Existem também biossensores plasmónicos, que se baseiam no fenómeno de SPR (*surface plasmon ressonance*). Este consiste na oscilação de densidades de carga na fronteira entre um material condutor e um meio dielétrico com permitividades de sinais opostos [20]. Estes biossensores incluem peliculas ou nanoestruturas de metal cujos electrões livres entram em ressonância por absorção do campo elétrico quando a orientação e a frequência deste coincidem com o seu movimento. No caso das peliculas, esta ressonância pode ser acoplada e propagada ao longo de um guia de onda sob a forma de um SPP (surface plasmon polariton). O campo evanescente desta onda é particularmente intenso e por isso propicio à interação com as partículas no meio de análise. Diversas topologias de deteção podem ser concebidas com base nestes efeitos, como por exemplo, interferómetros.

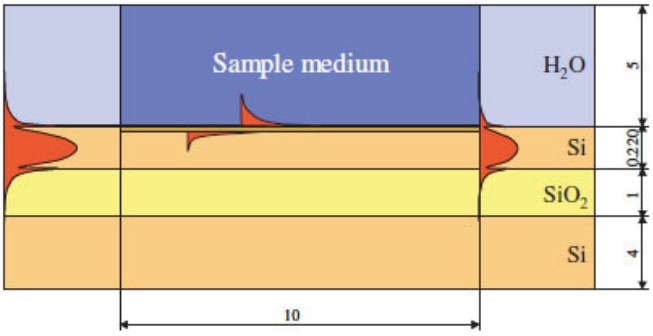


Figura 13 – Esquema de interferómetro plasmónico, retirado de [21].

A Figura 13 ilustra o conceito de um interferómetro plasmónico descrito em [21]. Este contém uma pelicula de ouro entre o guia de onda de silício e o meio de amostragem líquido. Os índices de refracção distintos (n_{Si} ~ 3,45 e n_{agua} ~ 1,33) garantem que os SPP nas duas superfícies da pelicula de ouro nunca cheguem a acoplar, propagando-se sem se influenciar. Desta forma as características do meio de amostragem não afetam o sinal de referência. No fim da pelicula de ouro, os dois sinais interferem consoante as suas fases dando lugar ao sinal que se pode ler na saída.

Foram apresentados diversos exemplos de transdutores. É também comum a implementação de uma destas estruturas em agregado, cada uma com uma dimensão ligeiramente diferente, tanto para garantir a existência de um ponto de funcionamento útil face a desvios do processo de fabrico ou a efeitos causados, por exemplo, pela temperatura, como para permitir outras técnicas de leitura do sinal. Um agregado de biossensores pode também ser concebido com o intuito de detetar diferentes analitos na mesma amostra, podendo fazer uma análise de saúde mais completa e despistar várias doenças.

Uma revisão mais aprofundada destas e outras topologias de sensores pode ser encontrada em [14], [15], [17] e [18]. Em [19] são fornecidas indicações para otimização dos sensores e é demonstrado que estruturas interferométricas são preferíveis às ressonantes quando o tamanho do sensor não está limitado pelo volume disponível da amostra. Também é referido que para a maior parte das aplicações clínicas é critica a capacidade de deteção de concentrações de analito numa amostra na ordem dos ng/ml, que corresponde a variações no índice de refracção na ordem dos 10^{-6} RIU (*refractive index unit*).

Seguem-se os limites de deteção de alguns dispositivos e as suas topologias e analitos.

Tabela 2 – Resumo de limites de detecção de vários sensores de fotónica integrada.

Topologia	LOD [RIU]	LOD [massa/area]	Fonte e ano
Anel ressonante	$8{,}3\times10^{-6}$	-	[16] - 2012
SPR	$1{,}4\times10^{-7}$	-	
SPR	10^{-5} a 10^{-7}	1 a 5 pg/mm^2	[14] - 2012
MZI	7×10^{-6}	-	
MZI	1×10^{-7}	0,06 pg/mm^2	
MZI	2×10^{-8}	0,01 pg/mm^2	
MZI	$9{,}2\times10^{-7}$	-	
YI	5×10^{-6}	-	
YI	9×10^{-8}	0,75 pg/mm^2	
YI	9×10^{-9}	0,013 pg/mm^2	
YI	$8{,}5\times10^{-8}$	0,020 pg/mm^2	
BiMW	$2{,}5\times10^{-7}$	-	
Anel ressonante	10^{-5}	-	
Anel ressonante	-	3,4 pg/mm^2	
Anel ressonante	$7{,}6\times10^{-7}$	1,5 pg/mm^2	
MZI	2×10^{-8}	0,01 pg/mm^2	[17] - 2020
YI	6×10^{-8}	0,02 pg/mm^2	
BiMW	5×10^{-8}	2,8 fg/mm^2	
MZI	10^{-7} a 2×10^{-8}	0,01 a 0,06 pg/mm^2	[15] -2020
YI	6×10^{-8} a 9×10^{-9}	0,01 a 0,75 pg/mm^2	
BiMW	10^{-8}	0,01 pg/mm^2	

Os limites de deteção reportados ao longo da última década são bastante promissores, atingindo os níveis necessários que são críticos para a maior parte das aplicações clínicas.

2.4. Sistemas com biossensores

Os sistemas de análise baseados em tecnologia de fotónica podem-se considerar como sendo constituídos pelas seguintes partes: uma fonte de luz, um biossensor de fotónica e um detetor para fazer a leitura do sinal resultante.

As montagens típicas destes sistemas costumam incluir um laser com um comprimento de onda selecionável e um fotodíodo, especialmente para biossensores que operam em estruturas interferométricas cuja deteção é feita através da intensidade do sinal. No caso dos biossensores assentes em estruturas ressonantes é também possível utilizar uma fonte de luz com uma maior largura espetral e fazer a deteção com um espetrómetro [25]. Estes sistemas habitualmente necessitam de instrumentação sofisticada e peças mecânicas para alinhar os feixes de luz entre os vários componentes, tornando-os volumosos e dispendiosos [26].

Desde há vários anos têm sido feitos esforços no sentido de concretizar estes sistemas com dimensões inferiores e componentes mais económicos, procurando viabilizá-los para a aplicação na área do diagnóstico num paradigma de POC. Várias plataformas de fotónica foram desenvolvidas para facilitar a investigação e a prototipagem, recorrendo a diferentes materiais, utilizando tecnologias com silício em isolante (SOI) ou com nitreto de silício. A Figura 14 resume algumas das plataformas existentes.

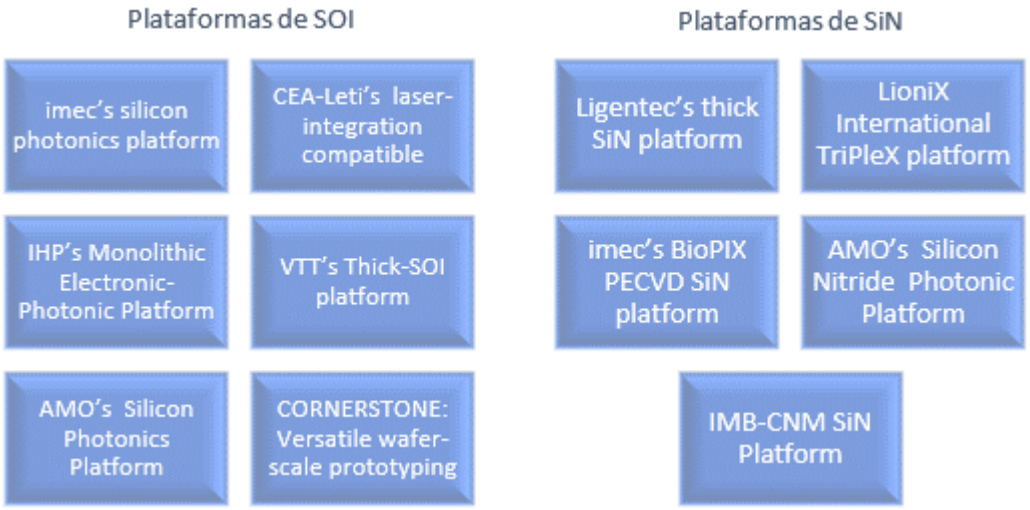


Figura 14 – Plataformas de fotónica em SOI e em SiN existentes.

O acoplamento eficiente de fibras óticas aos circuitos de fotónica continua a ser desafiante devido principalmente às diferentes dimensões da fibra e dos guias de onda de silício. A opção pelo acoplamento vertical com grelhas de difração facilita o

acoplamento, apresentando uma maior tolerância do ângulo de incidência e sendo de um fabrico normalmente mais simples do que as soluções para acoplamento lateral. No entanto, esta abordagem é mais sensível à variação do comprimento de onda e à polarização da luz [27].

Em termos de leitura do sinal do biossensor, métodos de deteção baseados na intensidade da luz acabam por ser preferenciais face aos métodos baseados na espetroscopia pois têm um maior potencial para a sua miniaturização [28].

Em [28] é apresentada uma solução para um sistema que aproveita o espectro de emissão de um LED e o deslocamento do comprimento de onda de ressonância do circuito do sensor para obter uma leitura baseada na intensidade com um fotodíodo. A Figura 15 ilustra este conceito e os componentes do sistema.

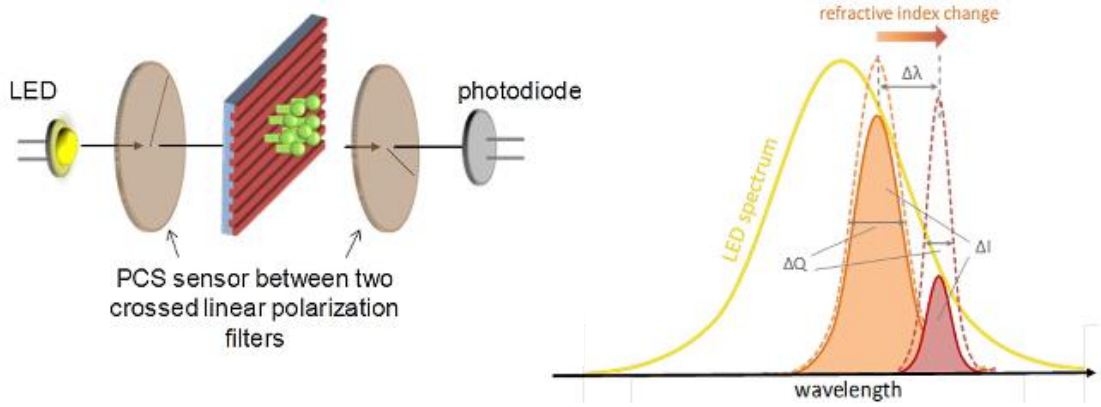


Figura 15 – Sistema de biossensor baseado em guias de onda com grelha ressonante e ilustração do mecanismo de deteção baseado na intensidade. Retirado de [28].

Em [25] é apresentado um sistema cujo biossensor é constituído por um agregado de quatro anéis ressonantes em tecnologia SOI, e é alimentado por um laser sintonizado. Cada anel pode também ser sintonizado através da aplicação de sinais elétricos. A leitura dos canais é realizada através de um único fotodíodo, utilizando uma técnica de multiplexagem no domínio do tempo. A Figura 16 ilustra o conceito e os vários componentes do sistema.

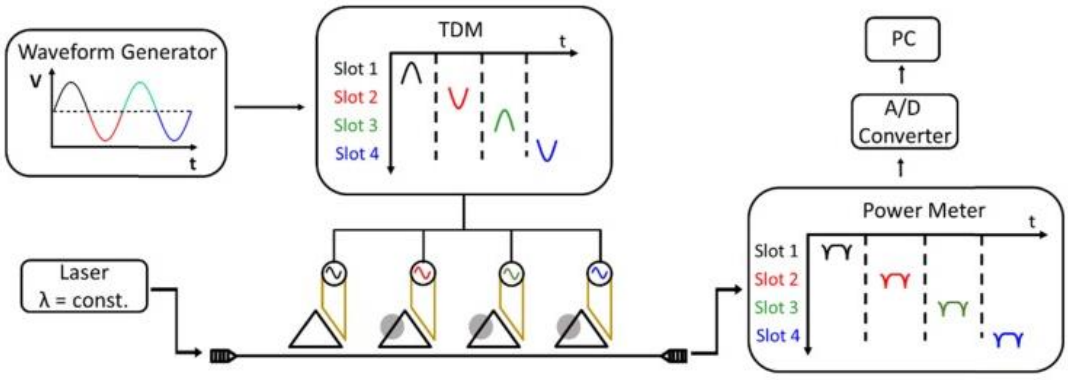


Figura 16 – Sistema baseado num agregado de anéis ressonantes em SOI, com multiplexagem no domínio do tempo e leitura com um único fotodíodo. Retirado de [25].

Outra solução para a leitura de biossensores com múltiplos canais passa pelo acoplamento de um agregado de fibras óticas, encaminhando o sinal por sua vez a um agregado de fotodíodos.

Em [29] é estudado um biossensor que utiliza MZIs recombinando os guias de referência e de amostragem num acoplamento com três sinais de saída, com a informação dos quais é possível obter a informação de fase sem ambiguidade, ruído de intensidade relativo ou desvanecimento da sensibilidade. A Figura 17 ilustra o circuito fotónico utilizado e os pontos de acoplamento com as fibras. O sistema é alimentado por um laser de Fabry-Perot, e a saída é acoplada a um conjunto de fibras óticas dopadas com Érbio, que encaminham o sinal para os fotodiodos.

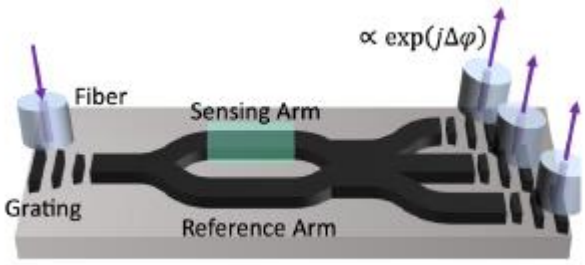


Figura 17 – Ilustração de um dos interferómetros de MZI do dispositivo, retirado de [29].

Além de fotodíodos, a deteção com câmaras CCD ou CMOS também é possível. Esta solução apresenta uma maior tolerância na montagem facilitando o alinhamento dos sinais óticos e permite também a leitura de múltiplos canais em simultâneo.

Em [30] é apresentado um dispositivo com biossensor interferométrico baseado em tecnologia SOI. Um laser emitindo no comprimento de onda de 1560 nm é acoplado através de fibra ótica e uma lente, recorrendo a uma estrutura de afunilamento para adaptar o guia de onda de entrada aos restantes circuitos de fotónica. Estes constituem um agregado de 6 MZIs, cuja saída é lida com uma câmara IR de InGaAs com 320x240 píxeis.

Em [31] descreve o desenho e fabrico de um biossensor baseado num interferómetro multimodo com ranhuras em tecnologia em oxinitreto de silício (SiON) com um comprimento de onda de operação de 633 nm. A Figura 18 ilustra o dispositivo. As experiências realizadas com o dispositivo fabricado foram feitas acoplando um laser de He-Ne a 633 nm através de uma fibra ótica ao guia de onda de entrada do sensor MMI. A leitura do sinal foi realizada com uma câmara BeamPro da empresa Photon Inc.

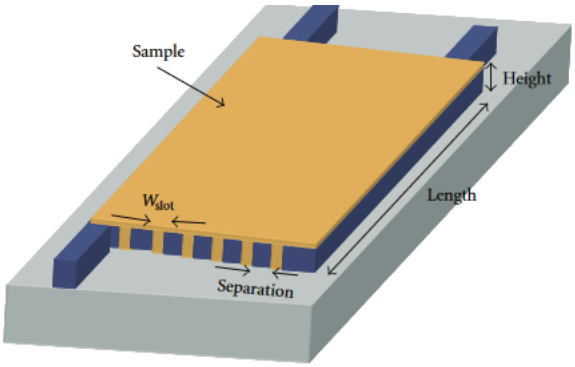


Figura 18 – Representação esquemática do interferómetro multimodo com seis ranhuras. Retirado de [31].

A concretização de um sistema completo no mesmo chip pode ser desejável, evitando as dificuldades na interconexão dos componentes, tornando mais fácil a sua produção em massa. No entanto, isto implicaria a introdução de um laser, ou LED, e de um ou mais fotodetetores no mesmo *chip* que o biossensor. Supondo que se pretende aproveitar as vantagens económicas provenientes do fabrico do dispositivo em plataformas baseadas em silicio, a introdução de semicondutores compostos do grupo III-V para implementação de emissores de luz e fotodetetores iria contrariar essa vantagem pelos custos introduzidos. Consequentemente, ao evitar esses custos os comprimentos de onda de operação possíveis iriam ficar mais restringidos. Sendo o silicio um semicondutor de banda indireta, significando que um eletrão na sua banda de condução não pode emitir um fotão ao transitar para a banda de valência, infelizmente até à data não foi possível concretizar uma fonte emissora baseada exclusivamente neste material. No entanto, é possível integrar os biossensores e os fotodetetores num *chip* de silicio, e utilizar um emissor externo, desde que este funcione num comprimento de onda cujo coeficiente de absorção do silicio permita a sua deteção. Este é o caso do último exemplo a ser apresentado.

Em [32] é descrito um sistema integrado em tecnologia de nitreto de silício, incluindo um agregado de MZIs e um agregado de fotodiodos para fazer a leitura. O sistema é alimentado por um laser VCSEL (*Vertically-Coupled Surface-Emission Laser*) a 850 nm, sendo este acoplado verticalmente através de uma grelha de difração, sendo uma das soluções mais frequentes. Tal como mencionado anteriormente, esta solução permite uma maior tolerância ao ângulo de incidência e é de um fabrico normalmente mais simples, mas ao mesmo tempo é menos tolerante a variações no comprimento de onda e na polarização, aumentando a exigência da qualidade da fonte emissora.

A Figura 19 ilustra o PIC concebido.

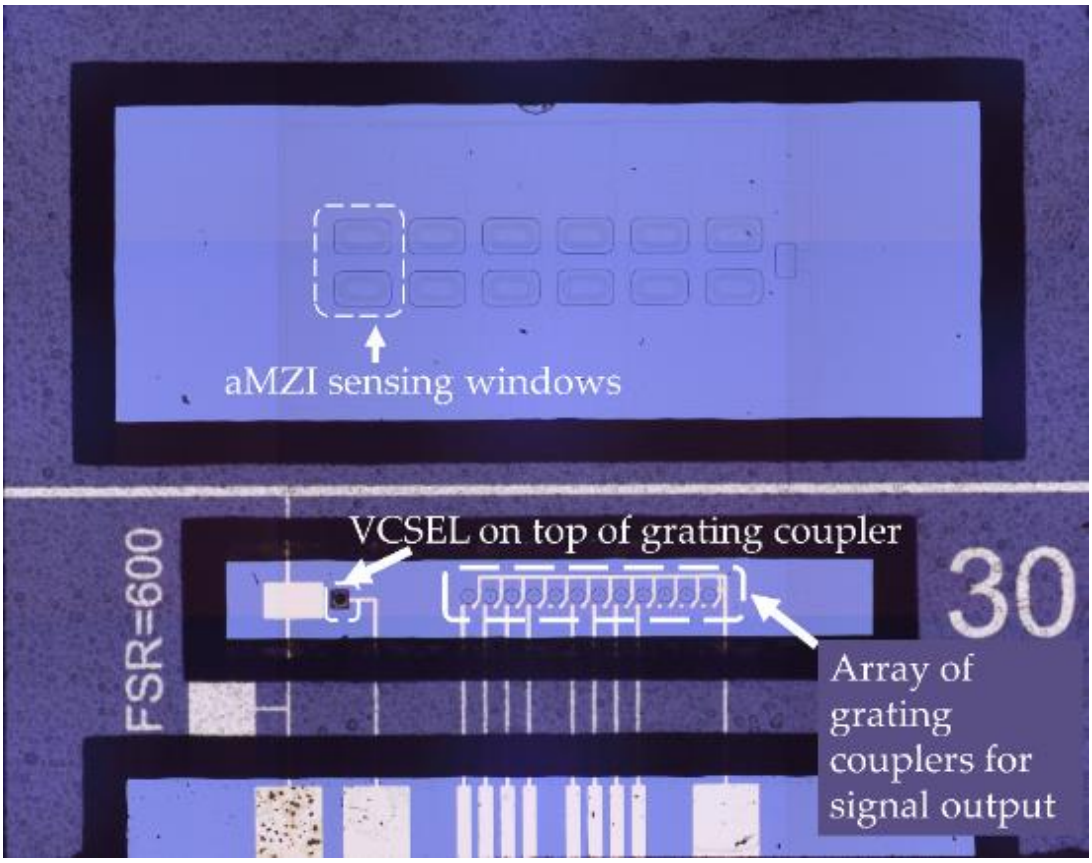


Figura 19 – Chip de fotónica integrada com interferómetros de MZI, um laser VCSEL e um agregado de fotodíodos em tecnologia de Si_3N_4. Retirado de [32].

Apesar da integração do fotodetetor e do biossensor no mesmo *chip* otimizar a interligação entre estes componentes, isto torna a produção do sistema economicamente inviável para uma operação nos comprimentos de onda das telecomunicações, em torno dos 1300 nm e dos 1550 nm. No caso de o projeto ter como requisito a operação nestes comprimentos de onda, poderá ser necessária uma montagem do sistema com biossensores e fotodetetores em *chips* separados.

3. Detetor ótico multicanal

Neste trabalho propôs-se a conceção de um módulo de aquisição de sinal para o dispositivo de diagnóstico em desenvolvimento no âmbito do projeto PhotoAKI. O projeto, realizado em parceria entre o ISEL, a NMS|FCM, a UNINOVA e o IST-ID, e financiado pela FEDER e pela FCT, pretende desenvolver e caracterizar um sensor fotónico para diagnósticos rápidos e monitorização para a condição da insuficiência renal aguda num ponto de atendimento para pacientes.

No presente capítulo começa-se por descrever o sistema que é pretendido, justificando a escolha do sensor de imagem e explicando as suas características, levando aos restantes requisitos do seu sistema de controlo. As ferramentas escolhidas para abordar o problema são apresentadas, e de seguida é explicado o funcionamento do protótipo. É feita uma descrição breve da arquitetura do *firmware*, da PCB e da solução de teste do sistema. Uma descrição mais detalhada do desenvolvimento é disponibilizada em anexo, juntamente com o código.

3.1. Requisitos do sistema

3.1.1. Contextualização

O dispositivo de diagnóstico em questão irá efetuar as análises através de um biossensor constituído por um agregado de interferómetros plasmónicos em paralelo, baseados numa tecnologia de fotónica em silício amorfo. O princípio de funcionamento deste tipo de interferómetro foi descrito no capítulo anterior. Cada interferómetro do agregado irá constituir um canal de deteção de biomarcadores cujo sinal ótico resultante, com um comprimento de onda de 1550 nm [1], será propagado através de um guia de onda até à saída do PIC onde poderá ser medido em termos de intensidade por um fotodetetor externo. O número de canais do dispositivo não foi determinado inicialmente, estando o bloco do biossensor em fase de desenvolvimento ao mesmo tempo que o módulo de deteção.

A solução desenvolvida para a deteção dos sinais recorre a um sensor de imagem linear em tecnologia CMOS. Um alinhamento correto dos feixes óticos é critico para uma leitura correta dos resultados. Estando todas as saídas do dispositivo fotónico no mesmo plano, a utilização de um sensor linear facilita o alinhamento dos sinais pois permite

[1] O motivo deste dimensionamento deve-se ao facto de que as instalações onde serão produzidos os protótipos do dispositivo estão preparadas para trabalhar com circuitos fotónicos nesse comprimento de onda, amplamente utilizado na atualidade em sistemas de telecomunicações.

alguma liberdade na posição do componente no eixo horizontal. Dispondo de mais de uma centena de píxeis, é possível a leitura simultânea de um número indeterminado de canais com um único componente[2], dispensando também o dimensionamento de estruturas fotónicas adicionais para o acoplamento de cada canal a uma fibra ótica.

Esta solução é menos dispendiosa do que a integração de fotodetetores no mesmo chip que o biossensor. Estes nunca poderiam ser realizados com o mesmo material que o restante sistema, em silício, pois este é transparente no comprimento de onda de funcionamento do dispositivo.

Os detetores para o comprimento de onda pretendido (1550 nm) são feitos com materiais semicondutores menos abundantes do que o silício pelo que apresentam sempre um custo considerável, frequentemente ultrapassando os vários milhares de euros por unidade. O fabricante Hamamatsu possui uma gama de sensores lineares de InGaAs com preços competitivos. Optou-se pelo modelo G13913-128FB por ser o mais económico da família, tendo um preço inferior a um milhar de euros. Este possui uma linha de 128 píxeis, tendo a janela de deteção uma área total de 6,4 mm por 0,25 mm. Outros modelos na mesma família apresentam janelas de deteção com larguras até 25,6 mm e até 1024 píxeis. Dado que a largura de 6,4 mm será suficiente para abranger todas as saídas óticas do PIC, e que não são esperados mais do que poucas dezenas de canais de deteção, considerou-se que o sensor em questão satisfaz os requisitos. Uma hipótese alternativa poderia ser o modelo G13913-256FG que é em tudo semelhante ao modelo escolhido, mas possui 256 píxeis numa janela de deteção com uma área idêntica, tendo cada pixel metade da largura.

3.1.2. Sensor de imagem G13913-128FB

O sensor de imagem G13913, ilustrado na Figura 20, é um dispositivo optoeletrónico em tecnologia CMOS que contém um agregado de fotodiodos de InGaAs para a deteção de radiação no espetro infravermelho próximo. Cada fotodiodo tem o seu próprio circuito de amplificação, onde é feita a integração da fotocorrente gerada durante um determinado período temporal. Os valores de tensão para cada pixel são depois transferidos sequencialmente para um andar de saída, juntamente com sinais auxiliares de sincronismo, resultando num sinal de vídeo analógico que descreve as intensidades da luz IR incidente ao longo da linha de píxeis. A Figura 21 ilustra a responsividade do sensor ao longo do espetro, com um máximo no comprimento de onda de 1550 nm.

[2] O número de canais de deteção não poderá ser superior ao número de píxeis e estará limitado por fatores como as dimensões destes e a largura dos feixes incidentes, que também poderá depender da distância entre o sensor e o PIC.

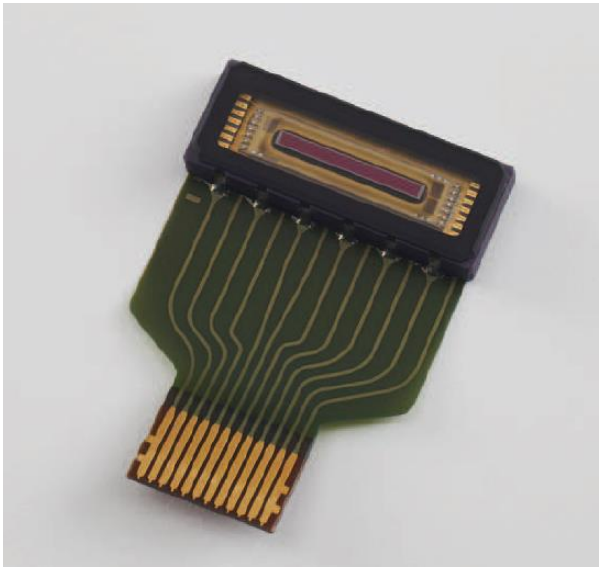


Figura 20 – Imagem do sensor G13913.

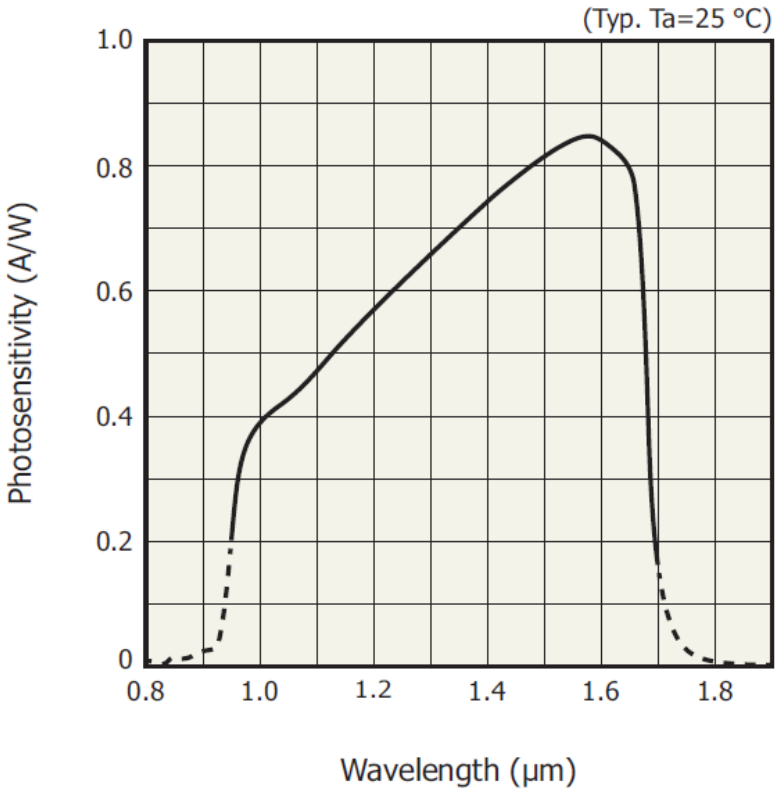


Figura 21 – Espetro de sensibilidade do dispositivo G13913, retirado de [33].

Além dos circuitos descritos, o dispositivo contém um circuito de temporização, que recebe os sinais de controlo e gera os sinais de sincronismo. A Figura 22 mostra o diagrama de blocos do sistema.

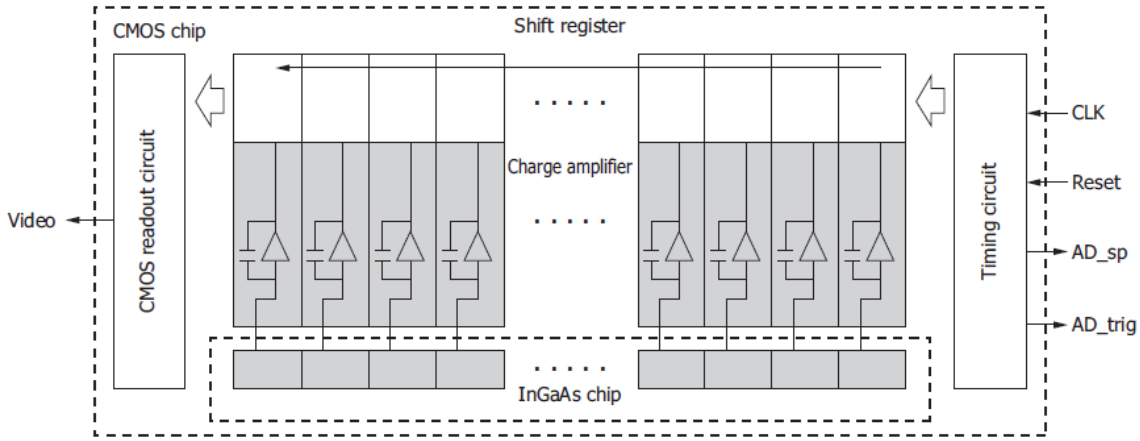


Figura 22 – Diagrama de blocos do sensor de imagem, retirado de [33]

A operação do dispositivo é controlada através dos sinais de relógio, 'CLK', e de reinício, 'Reset'. A aplicação de um pulso assíncrono do sinal 'Reset' inicia uma aquisição de imagem, e a sua duração determina o tempo de integração. O tempo de integração efetivo depende do sinal de relógio, podendo apenas tomar valores múltiplos do período de um ciclo de relógio. Isto é ilustrado na Figura 23, juntamente com a evolução temporal dos restantes sinais ao longo de um ciclo de aquisição de imagem.

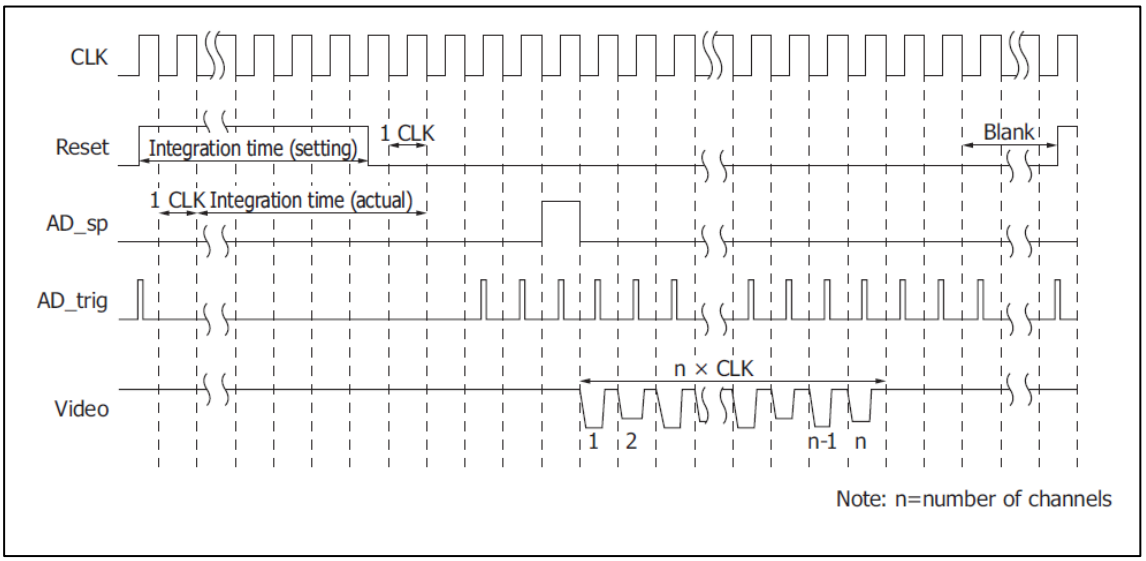


Figura 23 – Diagrama temporal dos sinais do sensor de imagem em funcionamento, retirado de [33].

Após a integração das fotocorrentes, segue-se a transmissão do sinal de vídeo. O sinal de sincronismo 'AD_trig' começa a ser gerado previamente. Este é constituído por uma sequência de impulsos que indicarão os momentos em que o circuito de saída estará preparado para a leitura, e poderá ser utilizado como gatilho para a conversão de cada amostra de tensão para o formato digital. O sinal 'AD_sp', é um único impulso que indica o início da transmissão da sequência de vídeo. O sinal de vídeo terá uma gama de valores limitada entre os valores de tensão no escuro e de tensão de saturação.

A Figura 24 mostra o circuito equivalente dos andares de deteção, integração e saída do dispositivo. A capacidade no circuito de integração é selecionável de entre dois valores através do sinal 'Cf_select', resultando em duas opções na razão da conversão de eletrões para tensão, de 16 ou 160 nV/e^-.

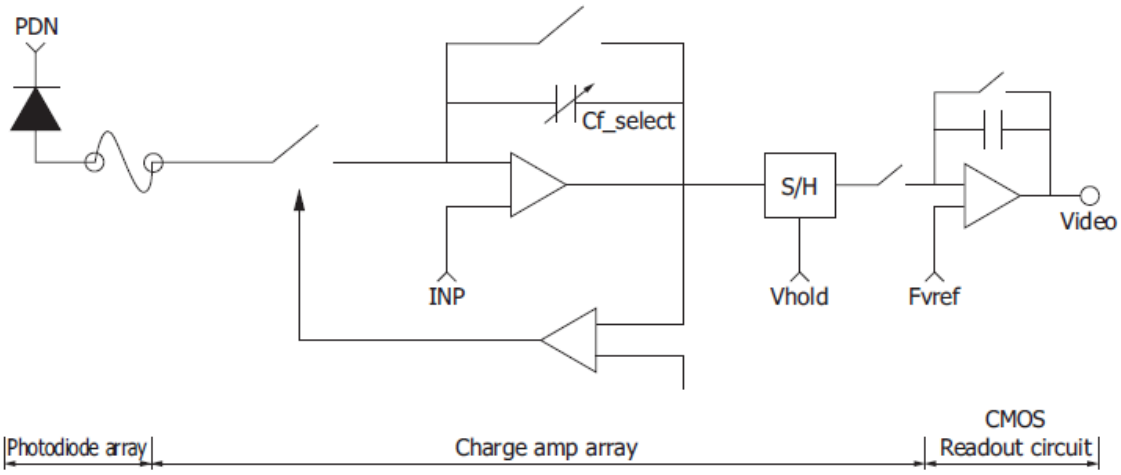


Figura 24 – Circuito equivalente do dispositivo, retirado de [33].

A Figura 25 ilustra as ligações do dispositivo integrado num sistema. Além dos sinais de entrada e de saída já mencionados, serão necessárias uma fonte de alimentação, e uma fonte de referência para os circuitos dos fotodíodos e do andar de saída.

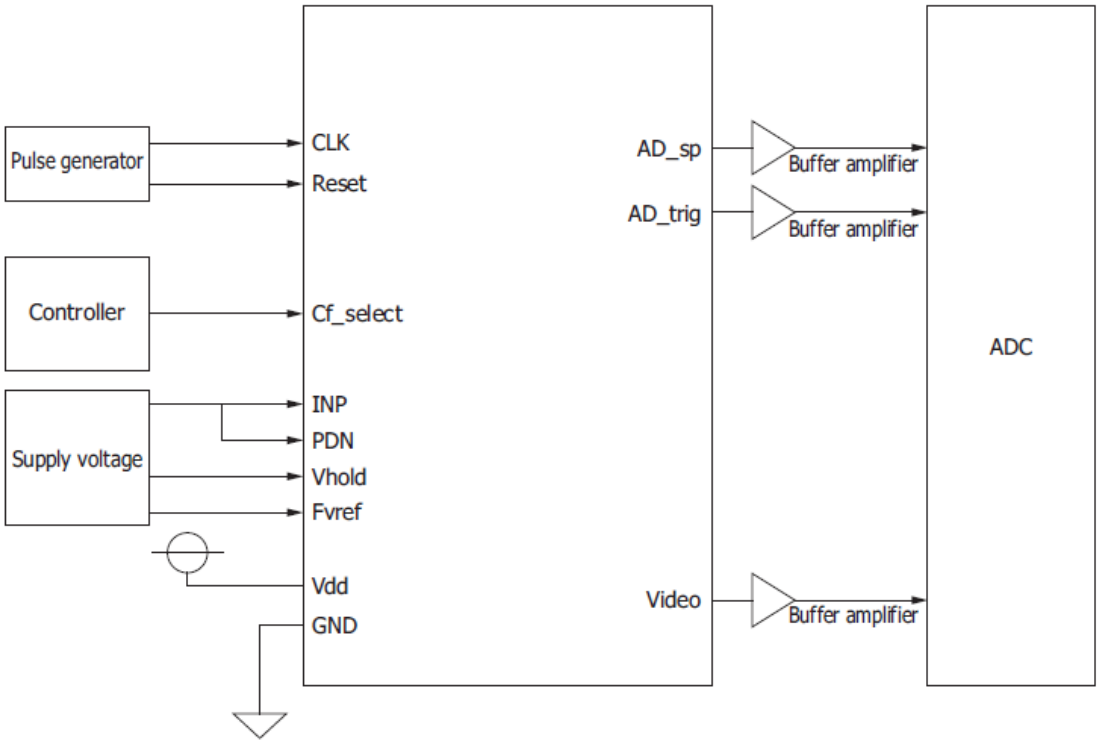


Figura 25 – Exemplo das ligações do dispositivo, retirado de [33].

3.1.3. Objetivos e requisitos

Pretende-se que o módulo de aquisição desenvolvido em torno do sensor de imagem seja portátil e possa ser utilizado e configurado a partir de qualquer dispositivo com as características de um computador pessoal. Para isso propõe-se a utilização de um conetor USB e a implementação do protocolo de comunicação SCPI. Este protocolo é um padrão utilizado em aparelhos de instrumentação que simplifica a configuração do dispositivo e a requisição de medições, e será descrito na secção seguinte.

Sendo o propósito do módulo de aquisição a integração posterior no sistema de diagnóstico descrito anteriormente, terá que permitir a configuração dos canais de deteção do sistema e efetuar a aquisição de imagem a um ritmo à escala do segundo, suficiente para descrever a evolução temporal dos sinais óticos à saída do biossensor. Um ritmo de aquisição superior poderá ser opcional, não sendo necessário para a aplicação em causa.

A operação do sensor de imagem poderá ser feita através de um microcontrolador. Este deverá gerar os sinais de controlo e receber os sinais de saída de acordo com as especificações descritas na ficha técnica do sensor. Algumas das mais importantes encontram-se na seguinte tabela.

Tabela 3 – Resumo das especificações elétricas dos sinais do dispositivo, segundo a ficha técnica [33].

Parâmetro	**Mínimo**	**Típico**	**Máximo**	**Unidade**
Alimentação (V_{DD})	3,0	3,3	3,6	[V]
Consumo de corrente	-	15	25	[mA]
Tensão de referência	2,4	2,5	2,6	[V]
Tensões máximas dos sinais digitais	V_{DD} - 0,25	V_{DD}	V_{DD} + 0,25	[V]
Tensões mínimas dos sinais digitais	-	0	0,25	[V]
Frequência de sinal de relógio (f_{op})	0,1	1	2	[MHz]
Frequência do sinal de vídeo	-	f_{op}	-	[MHz]
Tensão do sinal de vídeo no escuro	-	2,5	2,9	[V]
Tensão do sinal de vídeo em saturação	0,2	0,3	-	[V]

Os sinais digitais do dispositivo apresentam uma tensão típica de 3,3 V, pelo que a utilização de um microcontrolador com esta tensão de alimentação simplificaria a compatibilidade entre os dispositivos. É necessária a geração de um sinal de relógio com a frequência de operação do dispositivo, e um conversor analógico-digital (ADC) capaz de operar com uma frequência de amostragem idêntica.

Um circuito de acondicionamento entre a saída do sensor de imagem e o ADC é também importante para aproveitar correctamente a gama dinâmica do conversor e obter uma boa resolução, especialmente em situações de baixa intensidade do sinal ótico. A

conversão de um sinal com uma excursão de poucas dezenas de milivolts, num conversor com uma gama dinâmica de vários volts, estaria mais suscetível à perda da distinção entre os níveis de diferentes amostras devido aos erros de quantificação. Por isso propõe-se a introdução de um circuito de acondicionamento que se possa ajustar conforme as necessidades.

Por fim, a montagem do sensor de imagem com o microcontrolador (ou MCU, do inglês *Micro-Controller Unit*), em conjunto com o circuito de acondicionamento e outros circuitos auxiliares (p.e. fonte de tensão de 2,5 V), requererá a elaboração de uma placa de PCB.

A Figura 26 ilustra um esquema simplificado do sistema de aquisição pretendido.

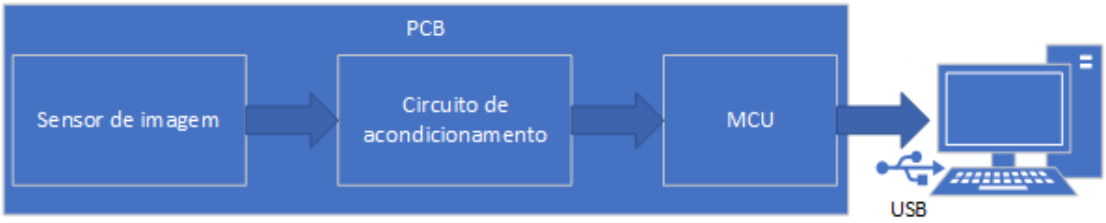


Figura 26 – Esquema da solução pretendida para o módulo de aquisição de sinal.

Dado que o sensor de imagem, apesar de ser o mais económico no mercado, continua a ser um dispositivo frágil e com um custo considerável, propõe-se a adoção de medidas para o desenvolvimento e verificação do funcionamento correto do sistema antes da montagem do mesmo. Uma dessas medidas será a substituição deste dispositivo por outro que possa gerar os sinais de saída com características semelhantes.

3.1.4. Protocolo SCPI

Sendo um requisito do sistema a capacidade de comunicar com um computador através de uma porta USB, escolheu-se como solução uma implementação de um protocolo com um padrão amplamente difundido e comprovado para dispositivos de instrumentação, nomeadamente o SCPI - *Standard Commands for Programmable Instruments*. Este é um padrão aberto que define a sintaxe e um conjunto de comandos num formato de texto que pode ser facilmente interpretado por humanos ou máquinas, e que pode ser expandido com os comandos para um dispositivo em particular, facilitando a introdução de novas funcionalidades que venham a ser necessárias no futuro.

O padrão suporta dois tipos de comandos, comandos comuns e comandos numa estrutura em árvore. Os comandos comuns iniciam-se com um asterisco (`*<COMANDO>`) e podem ser chamados em qualquer contexto. Os restantes comandos iniciam-se com dois pontos (`:<COMANDO>:<SUBCOMANDO>`) e guardam a informação do nó a que

pertence o último subcomando executado. Execuções subsequentes de um subcomando pertencente ao mesmo nó podem omitir os dois pontos e o comando superior (<SUBCOMANDO>). O início de um comando com dois pontos remete o caminho para a raiz da árvore. Os comandos podem ser de SET ou QUERY, neste caso com um ponto de interrogação (<COMANDO>?), e podem ser seguidos de um ou mais parâmetros separados por virgulas (:<COMANDO> <PARAMETRO>, <PARAMETRO> ou :<COMANDO>? <PARAMETRO>). Os comandos podem ser identificados pelo seu nome completo, ou por uma abreviatura que é constituída pela parte do comando que é descrito em maiúsculas (p.e. CONFigure). No entanto o padrão não distingue maiúsculas e minúsculas e ignora espaços, excepto entre um comando e os seus parâmetros. Vários comandos podem ser chamados na mesma linha, separados por um caracter ponto e vírgula.

Exemplifica-se o funcionamento do protocolo considerando os comandos representados na Figura 27. Suponha-se que o sistema que serve de base para este exemplo contem duas variáveis de configuração, 'clock' e 'int_time', um comando de medição 'acquire' e outra variável que define o tipo dos dados transmitidos, 'datatype'.

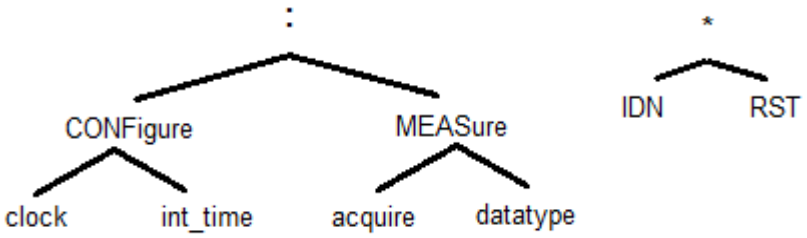


Figura 27– Exemplo de estrutura de comandos, com comandos comuns (*) e árvore (:).

Suponha-se que num terminal é introduzida a seguinte sequência de comandos:

```
:CONF:clock 1000; *IDN; int_time?; :MEAS:datatype string; acquire
```

O comportamento esperado seria que o primeiro comando atribuísse o valor 1000 à variavel 'clock'; o segundo comando iria retornar o código de identificação do dispositivo; o terceiro comando iria retornar o valor da variável 'int_time', pertencente ao mesmo nó que 'clock'; o quarto comando iria definir a apresentação dos valores medidos em formato de *string* e, por fim, o último comando iria efetuar uma medição e retornar o resultado.

3.2. Ferramentas principais

Neste subcapítulo descrevem-se as ferramentas utilizadas no desenvolvimento do módulo de aquisição de sinal, nomeadamente o microcontrolador, o ambiente de desenvolvimento, o *software* CAD para o projeto da PCB e outros equipamentos.

3.2.1. Microcontrolador

O microcontrolador que se considerou mais adequado para este projeto foi o núcleo STM32F103C8, presente na placa de desenvolvimento STM32 Smart V2.0 (Figura 28), por disponibilizar a capacidade de processamento e os periféricos necessários, sem exceder exageradamente as necessidades, por ser compatível com os níveis de tensão do sensor de imagem escolhido e pela dimensão contida da placa de desenvolvimento. O MCU integra uma unidade de processamento ARM Cortex-M3 de 32 bits (um computador RISC - *reduced instruction set computer*), juntamente com memórias Flash e SRAM, um DMA e um conjunto de periféricos, nomeadamente interfaces de comunicação SPI, I2C, UART e USB, vários temporizadores, um par de conversores analógico-digitais, e portos GPIO. O manual de referência desta família de dispositivos encontra-se em [34], uma descrição resumida da sua arquitetura está disponível no Anexo A.1. e o esquemático da placa no Anexo A.2.

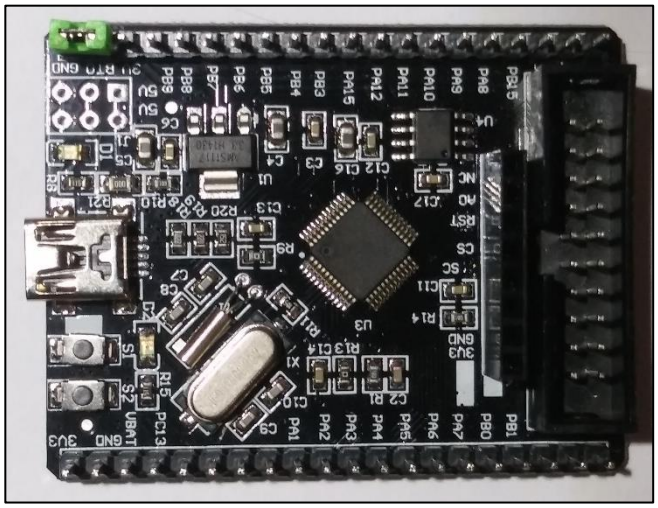


Figura 28 – Fotografia do microcontrolador STM32F103C8 na placa de expansão STM32 Smart V2.0.

A configuração e utilização dos diversos componentes são feitas lendo e escrevendo nos seus registos, encontrando-se no manual de referência descrições abrangentes das suas funcionalidades e mapas de memória. Isto pode ser feito diretamente, com recurso à biblioteca dos dispositivos STM32F1 que contém as definições dos endereços e estruturas de dados úteis para manipular os dispositivos, ou utilizando a biblioteca de controladores da camada de abstração de *hardware* ou HAL – *Hardware Abstraction Layer*, que contém estruturas e funções que tratam da gestão dos dispositivos da família STM32F1 de forma genérica. A segunda abordagem traz vantagens sobretudo a nível

de portabilidade do código para outros dispositivos da família, mas a disponibilidade de controladores pré-feitos para os vários periféricos também pode ser uma mais-valia. Contudo a primeira abordagem, mais personalizada, possibilita por vezes a escrita de código mais simples e eficiente para uma utilização especifica do microcontrolador, pois evita as estruturas por vezes complexas que apenas são necessárias para tornar o programa compatível com vários dispositivos e com a sua diversidade de aplicações.

Neste trabalho foram utilizadas as duas abordagens, utilizando os controladores da camada de abstração quando estes se mostraram adequados, ou a afetação direta dos registos quando isto se mostrasse mais simples. Para a base do programa foi utilizada uma biblioteca do Arduíno que, recorrendo também às bibliotecas da família STM32, trata a inicialização do sistema e disponibiliza uma interface programática amplamente conhecida. Esta suporta algumas das funcionalidades mais comuns nos microcontroladores.

3.2.2. Ambiente de desenvolvimento para o firmware

O ambiente de desenvolvimento escolhido foi o IDE do Arduino devido à familiaridade, assim como a existência de uma biblioteca Arduino que inclui a biblioteca de *drivers* de abstração de hardware para a família de dispositivos STM32, o que permitiu uma abordagem expedita ao problema. Após a instalação do *firmware* do *bootloader* no dispositivo, e dos *drivers* DFU Maple, iniciaram-se os testes aos pinos GPIO, temporizadores, conversores e interfaces de comunicação, seguindo-se para o desenvolvimento dos controladores do sensor de imagem. As documentações consultadas, para além do manual de referência da família de núcleos STM32F1XX de 32 bits [34], foi a ficha técnica do núcleo STM32F103x8 [35] e o manual de referência dos *drivers* HAL STM32F1 [36]. A linguagem de programação utilizada foi C, recorrendo à linguagem C++ apenas para a utilização da biblioteca do Arduino.

3.2.3. *Software* CAD para projeto de PCB

A placa de circuito impresso (ou *printed circuit board* - PCB) foi desenhada através do *software* KiCad, sendo o projeto apresentado no capítulo 3.5. Este permite desenhar o esquemático de um circuito com os símbolos dos dispositivos e depois colocar e interligar os componentes no desenho de uma placa com dois planos de metal, fazendo verificação das regras de desenho e da equivalência entre a placa e o seu esquemático. Os ficheiros de fabrico podem depois ser gerados e enviados para uma fábrica para produzir a PCB. Para esse efeito recorreu-se aos serviços da empresa JLCPCB.

3.2.4. Outros equipamentos

Durante o desenvolvimento do sistema com o microcontrolador foi utilizado um osciloscópio digital Hantek de dois canais e com gerador de funções, como ilustrado Figura 29, juntamente com uma ponta de prova de alta impedância para observar sinais com frequências superiores a vários quilohertz.

Figura 29 – Osciloscópio digital Hantek com dois canais e gerador de sinais.

Para substituir o sensor de imagem durante o desenvolvimento do sistema foi utilizado um microcontrolador Arduino Due, como ilustrado na Figura 30.

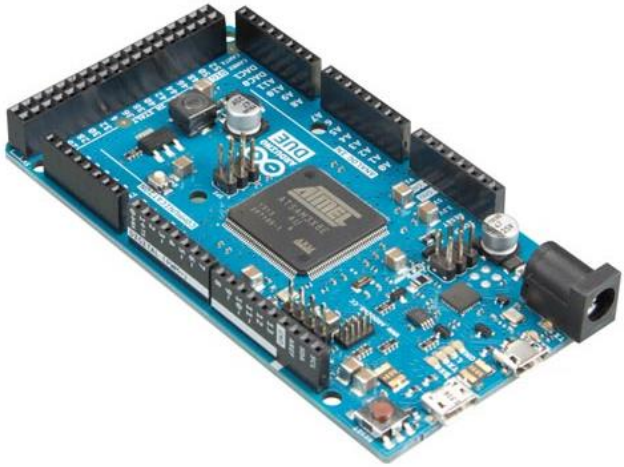


Figura 30 – Arduino Due, utilizado para gerar sinais de teste e substituir o sensor de imagem durante o desenvolvimento.

A utilização destes equipamentos permitiu, em tempo de confinamento devido à pandemia causada pelo vírus Covid-19, realizar uma parte significativa do desenvolvimento do *hardware* a partir de casa. Assim foi possível confirmar o funcionamento correto dos sinais de controlo gerados pelo microcontrolador, tal como avaliar a resposta dos circuitos de acondicionamento aos sinais de teste e efetuar outros testes ao longo do desenvolvimento.

3.3. Protótipo

Sumário

Para o controlo do sensor de imagem foi concebida uma PCB que contém um conector FPC para a montagem do sensor, um circuito de acondicionamento de sinal programável, um conversor analógico-digital externo, circuitos de alimentação, e dois conectores SMA, para um sinal de controlo do laser que alimenta o circuito de fotónica, e para uma saída analógica do sinal processado. A PCB encaixa na placa do microcontrolador STM32 Smart V2.0 (Figura 31).

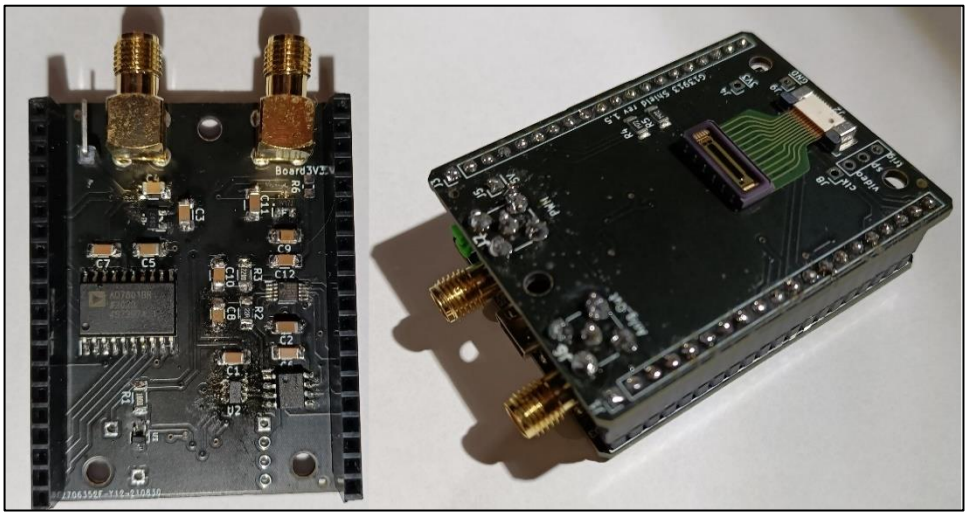


Figura 31 – À esquerda, o lado interior da PCB do módulo de aquisição. À direita, a o módulo de aquisição montado na placa STM32 Smart V2.0.

O *firmware* desenvolvido para o microcontrolador permite a um utilizador, através da introdução de comandos SCPI num terminal COM ou através de uma aplicação com interface gráfica, manipular os parâmetros do sistema e fazer a aquisição de sinal dos circuitos de fotónica. A Figura 32 mostra de forma simplificada a relação do microcontrolador com os restantes dispositivos.

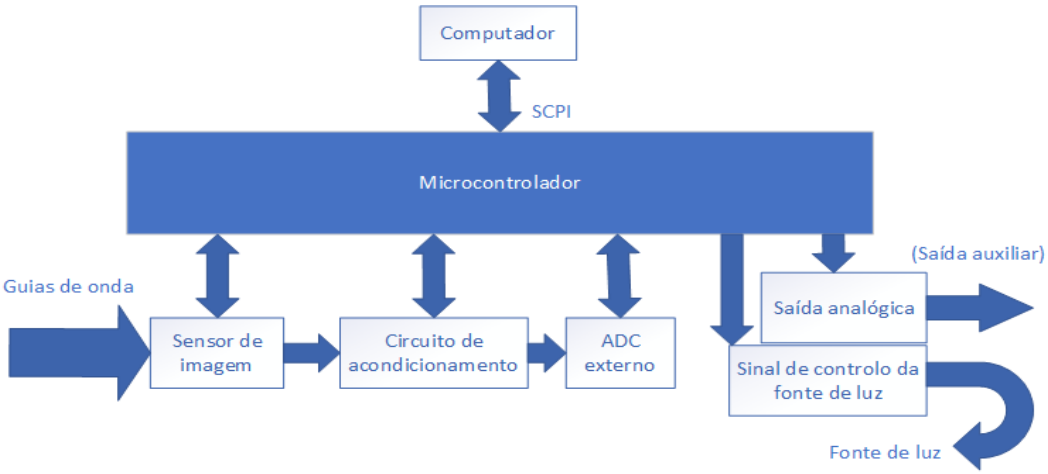


Figura 32 - Esquema simplificado do microcontrolador e das suas ligações aos vários componentes do sistema.

A comunicação com o computador é feita através de um porto série virtual (VCOM) no barramento USB. O programa principal do microcontrolador lê e executa os comandos SCPI, dos quais se realçam os comandos de configuração e os comandos de medição. Os 128 píxeis do sensor de imagem podem ser agrupados e mapeados em qualquer número entre 1 e 128 canais, por forma a facilitar a leitura e pós-processamento/armazenamento dos dados. Pretende-se assim que, após uma configuração correta, cada canal do módulo de aquisição represente a intensidade de luz na saída do respectivo canal de deteção do biossensor.

Funcionamento

Os comandos de medida fazem atuar os sinais de controlo sobre o sensor de imagem. A Figura 33 ilustra as ligações do sensor de imagem e os periféricos do microcontrolador. Para gerar os sinais de relógio e de *reset* utilizaram-se temporizadores de *hardware*. Estes permitem um controlo preciso, à escala do microssegundo, tanto da frequência de operação do dispositivo como da duração período de integração, o que não seria viável, nem eficiente, por *software* nas escalas temporais em questão e no tipo de solução pretendida (com um microcontrolador económico e com um consumo relativamente baixo). A precisão destes sinais é importante pois quaisquer variações no tempo de integração efetivo poderiam introduzir variações indesejadas, ou seja, ruído, no sinal medido.

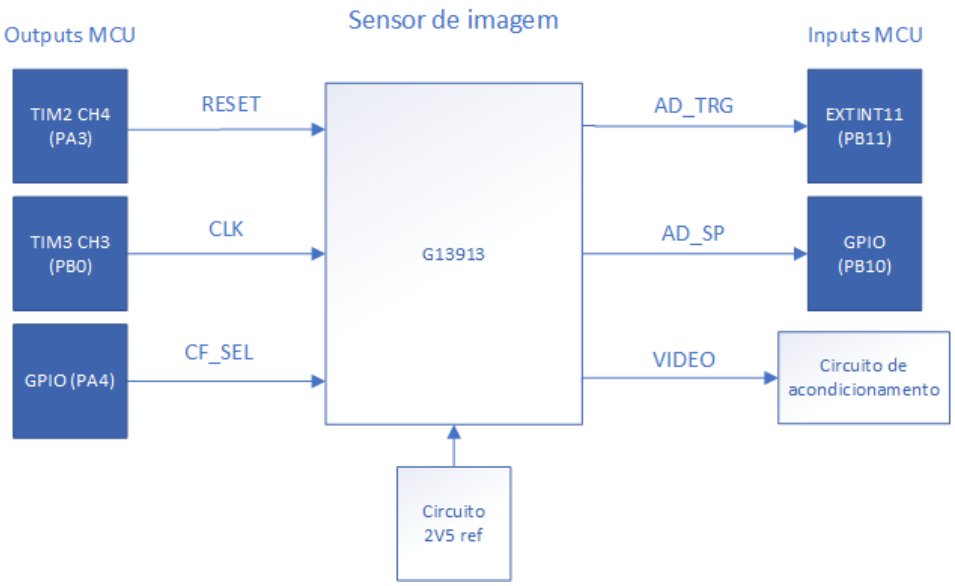


Figura 33 – Esquema das ligações ao sensor de imagem e periféricos do microcontrolador utilizados.

O sinal de vídeo passa por um circuito de acondicionamento com oito configurações otimizadas para vários níveis possíveis de intensidade do sinal ótico. As oito configurações resultaram do estudo das características do sinal de vídeo e do circuito baseado num amplificador de ganho programável (PGA), por forma a efetuar uma

amplificação e deslocamento do sinal aproveitando ao máximo a gama dinâmica do ADC. A Figura 34 mostra um diagrama de blocos do circuito de acondicionamento. Este descreve o circuito, constituído por um PGA cuja tensão de referência é ajustada com um DAC, e as ligações e barramentos utilizados. Esta solução permite uma configuração do circuito com um elevado grau de liberdade na amplificação e no deslocamento de nível do sinal, podendo-se, no entanto, optar simplesmente pela utilização das oito configurações otimizadas.

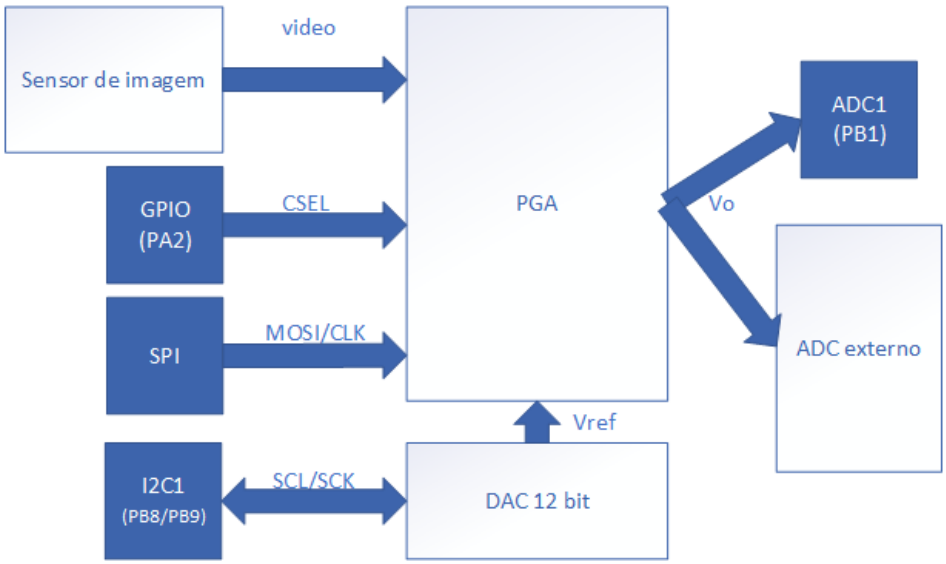


Figura 34 – Diagrama de blocos do circuito de acondicionamento de sinal e recursos do MCU utilizados.

Cada amostra do sinal de vídeo é convertida para o formato digital, tendo sido introduzido um ADC externo com uma resolução de bits superior ao ADC do microcontrolador. Mantiveram-se ambos os ADCs no sistema por uma questão de redundância. Após o processo de conversão, o microcontrolador continua para as etapas de processamento e de transmissão.

O processamento consiste no cálculo da média do valor medido em cada grupo de píxeis que constitui um canal do sistema. Os resultados são transmitidos de seguida através do porto COM, sobre a forma de caracteres ASCII ou sequência de valores binários. Após a transmissão no porto COM os resultados são apresentados na saída analógica, se esta opção estiver ativa.

Comandos SCPI e parâmetros

Seguem-se as configurações possíveis através dos comandos SCPI, assim como os valores de entrada permitidos, a sintaxe e a possibilidade de questionar o valor configurado (*query*), para cada um dos blocos descritos anteriormente no esquema da Figura 32.

A tabela seguinte resume os comandos relacionados com o sensor de imagem e o processo de aquisição e processamento dos canais.

Tabela 4 – Comandos SCPI relacionados com o sensor de imagem e medições.

Comando	Descrição	Valores aceites	Query	Obs
:CONF:clock <x>	Frequência do sinal de relógio	300 a 2 000 000	Sim	Hz
:CONF:intg <x>	Tempo de integração	1 a 5 000 000	Sim	µs
:CONF:csel <x>	Opção de eficiência de conversão.	booleano[3]	Sim	
:MEAS:single	Única aquisição	N/D	Não	
:MEAS:cont <x>	Aquisição contínua	booleano[3]	Sim	
:MEAS:interval <x>	Intervalo entre aquisições cont.	10 a 300000	Sim	ms
:MEAS:string	Sequência de valores em *string*	N/D	Sim	
:MEAS:bytes	Sequência de valores em bytes	N/D	Sim	
:CONF:n_ch <x>	Define o número de canais	1 a 128	Sim	
:CONF:ch_lmt <x>,<y>,<z>	Define o limite inferior <y> e o limite superior <z> para o canal <x>.	0 a 127	Sim	
:CONF:ch_equal	Distribui de igual forma os 128 píxeis pelo número de canais.	N/D	Sim	
:TEST:resetpulse <x>	Gera pulso de *reset de* <*x*>.	1 a 5 000 000	Não	µs
*IDN	Devolve o código de identificação.	N/D	Não	
*RST	Reinicia o dispositivo.	N/D	Não	

O limite superior da frequência de relógio foi determinado pela especificação do sensor de imagem, e o limite inferior pelo valor máximo do divisor de relógio do temporizador. A possibilidade de aplicar frequências de relógio inferiores à especificação do sensor de imagem pode ser útil em testes do sistema.

Os limites do tempo de integração foram escolhidos, por um lado, para permitir um aproveitamento da resolução temporal permitida pela frequência típica do sensor de imagem (1 MHz, ou seja, passos de 1 µs) e por outro, para possibilitar um tempo de integração até 5 segundos.

A eficiência da conversão da carga elétrica para tensão pode ser selecionada entre dois níveis, tal como é permitido pelo sinal de controlo que o sensor de imagem aceita.

O processo de aquisição pode ser realizado de forma singular, ou num modo de aquisição contínua com um intervalo definido com uma precisão à escala do milissegundo. O intervalo entre aquisições terá de ser superior à soma dos tempos de integração, de transmissão do sinal de vídeo, de processamento e de transmissão dos dados processados incluindo a apresentação do resultado na saída analógica. Uma

[3] O parâmetro pode ser escrito de diferentes formas: 1/0, true/false, high/low ou on/off.

frequência de aquisição até 100 Hz é possível, com um tempo de integração na ordem do milissegundo e um formato de dados de saída em valores binários. O limite máximo para o intervalo entre aquisições foi limitado arbitrariamente em 5 minutos.

Os dados podem ser transmitidos sob a forma de caracteres ASCII ou valores binários.

A seguinte tabela resume os comandos SCPI relacionados com o circuito de acondicionamento.

Tabela 5 – Comandos SCPI relacionados com o circuito de acondicionamento.

Comando	Descrição	Valores aceites	Query?	Obs
:CONF:pga_gain <x>	Configura o ganho do PGA	1, 2, 4, 5, 8, 10, 16, 32.	Sim	
:CONF:pga_vref <x>	Configura a tensão Vref do PGA	0 a 3300.	Sim	mV
:CONF:pga_conf <x>	Configura o ganho do PGA e a tensão Vref ótima.	1, 2, 4, 5 ,8 ,10, 16, 32.	Não	

Os valores disponíveis foram limitados pelas opções de ganho do PGA e pela resolução do DAC utilizado. Os valores de configuração ótimos encontram-se armazenados no microcontrolador podendo ser aplicados com um único comando.

A seguinte tabela resume os comandos SCPI relacionados com os conversores.

Tabela 6 – Comandos SCPI relacionados com os conversores analógico-digitais.

Comando	Descrição	Valores aceites	Query?	Obs
:CONF:adc_ext <x>	Configura a seleção do conversor, externo ou interno	booleano[3]	Sim	O estado verdadeiro corresponde ao uso do ADC externo.
:CONF:spi_freq <x>	Configura a frequência do barramento SPI	100 000 a 50 000 000	Sim	Hz
:TEST:adc_conv	Executa uma única conversão e devolve o valor.	N/D	Não	O valor é retornado em forma de *string*, entre 0 e 32767.

A seguinte tabela resume os comandos SCPI relacionados com a saída analógica.

Tabela 7 – Comandos SCPI relacionados com a saída analógica.

Comando	Descrição	Valores aceites	Query?	Obs
:CONF:anlg_out <x>	Liga ou desliga a saída analógica	Booleano[3]	Sim	
:CONF:anlg_freq <x>	Configura a frequência do sinal	1000 a 100 000	Sim	Hz
:TEST:anlg_val <X>	Coloca um nível de tensão na saída analógica.	0 a 3300	Não	mV

A frequência do sinal de vídeo analógico gerado foi limitada entre 1 kHz e 100 kHz, podendo ser programada entre qualquer um destes valores. O limite inferior impede a ocorrência de um atraso alargado do programa devido à utilização de uma frequência baixa, enquanto a frequência máxima foi limitada pelo tempo de execução das funções.

A tabela seguinte descreve os comandos SCPI do sinal PWM.

Tabela 8 – Comandos SCPI relacionados com o sinal PWM.

Comando	Descrição	Valores aceites	Query?	Obs
:CONF:pwm_period <x>	Configura o período do sinal PWM.	2 a 200 000	Sim	µs
:CONF:pwm_duty <x>	Configura o *duty-cycle* do sinal PWM.	0 a 100	Sim	%
:CONF:pwm_state <x>	Liga ou desliga o sinal PWM.	booleano[3]	Sim	
:TEST:pwm_pulse <x>	Liga o sinal PWM durante x milissegundos.	0 a 5 000	Não	ms

O período do sinal PWM foi limitado pela base temporal utilizada, à escala do microssegundo. Esta, permite a escolha de qualquer valor inteiro entre 0 e 100 para o *duty-cycle* para qualquer frequência inferior a 10 kHz.

Tendo sido apresentado o protótipo, descrevendo o seu funcionamento e os comandos SCPI disponíveis, nos próximos subcapítulos serão abordados a arquitetura do *firmware*, a PCB e a solução de teste para o sistema completo.

3.4. Arquitetura do *firmware*

Sumário

Na Figura 35 ilustram-se os módulos desenvolvidos e as suas posições entre o programa principal e o *hardware*.

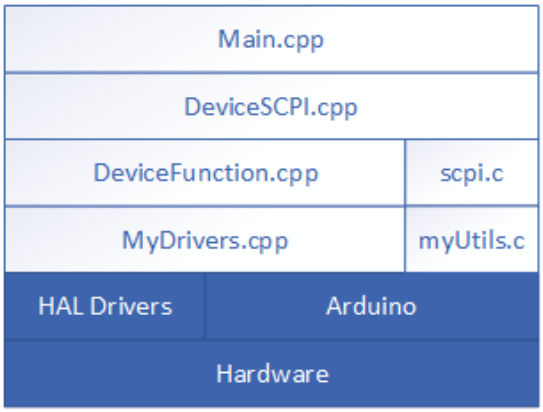


Figura 35 – Diagrama de blocos do stack de software desenvolvido para o microcontrolador.

A configuração do sistema, assim como as funções principais de aquisição, processamento e transmissão, são resolvidas no módulo `DeviceFunctions`. Este utiliza as funções do módulo `MyDrivers`, que trata os controladores dos vários subsistemas (sensor de imagem, circuito de acondicionamento, ADC externo, saída analógica, etc…).

O módulo `SCPI` dispõe de uma interface programática para uma configuração dinâmica do protocolo SCPI. Este foi desenvolvido para possibilitar uma implementação genérica, sendo possível através da sua API construir uma árvore de comandos e adicionar funções conforme as necessidades, assim como resolver a leitura dos comandos. O módulo `myUtils` contem as funções auxiliares desenvolvidas para a leitura dos comandos SCPI.

O módulo `DeviceSCPI` usa a API do módulo `SCPI` para tratar a inicialização da estrutura de comandos do protocolo SCPI, registando as funções de configuração e operação do dispositivo que constam no módulo `DeviceFunctions`.

Ao longo do desenvolvimento dos controladores foram efetuados testes unitários aos sinais de controlo e aos circuitos auxiliares de acondicionamento e de saída analógica, mostrando o seu funcionamento correto dentro dos parâmetros aceites nos comandos SCPI.

Desenvolvimento

A descrição completa do desenvolvimento do *firmware* que foi carregado no microcontrolador encontra-se no Anexo B, juntamente com o código, e com os resultados de testes unitários.

Nesse anexo os módulos serão descritos de baixo para cima, começando pelos controladores de nível mais baixo que atuam sobre o sensor de imagem e outros subsistemas, continuando para o sistema de configuração e funções do dispositivo, e por fim, a implementação do protocolo SCPI.

3.5. Projeto de PCB

Sumário

Foi projetada uma PCB com as seguintes características:

- Conector FPC de 12 pinos com espaçamentos de 0,5 mm para encaixe do sensor de imagem no lado dianteiro da PCB;
- Dimensões idênticas à placa de desenvolvimento do MCU, para encaixe através dos *pin-headers* laterais;
- Montagem dos componentes dos circuitos auxiliares no lado traseiro, ficando enclausurados entre as duas placas;
- Conjunto de pinos para acesso fácil aos sinais de controlo do sensor de imagem no lado superior, para efeitos de teste e depuração;
- Regulador de tensão de 3,3 V para uma alimentação externa de 5 V;
- Opção de *shunt* com a alimentação do MCU de 3,3 V;
- Perfuração em três pontos para eventual necessidade de montagem em caixa.

O planeamento da posição dos componentes na PCB, juntamente com os seus circuitos auxiliares (condensadores de desacoplamento da alimentação, por exemplo), foi feito seguindo as sugestões dos guias de desenho nas fichas técnicas dos dispositivos.

O esquemático que corresponde ao circuito da PCB encontra-se no Anexo C.

A Figura 36 mostra uma previsualização dos dois lados da placa no *software* KiCad.

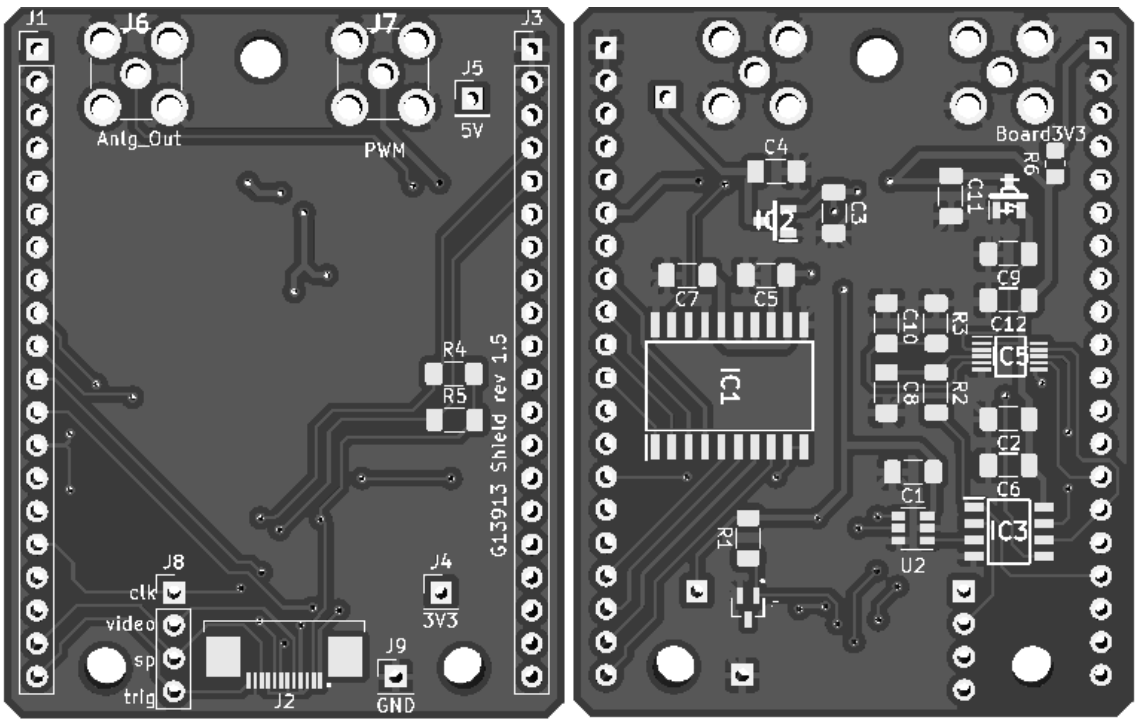


Figura 36 – Visualização 3D da placa no software KiCad.

Esquema de planificação da placa

A Figura 37 mostra o desenho final da PCB, que teve os cuidados necessários para que esta fosse compatível com a placa de expansão do microcontrolador. Isto incluiu uma medida cuidada das dimensões da outra placa, assim como da posição de componentes volumosos como o conector USB. O pino para a alimentação externa de 5 V foi colocado na mesma posição que o pino de 5 V da placa do MCU para uma ligação fácil.

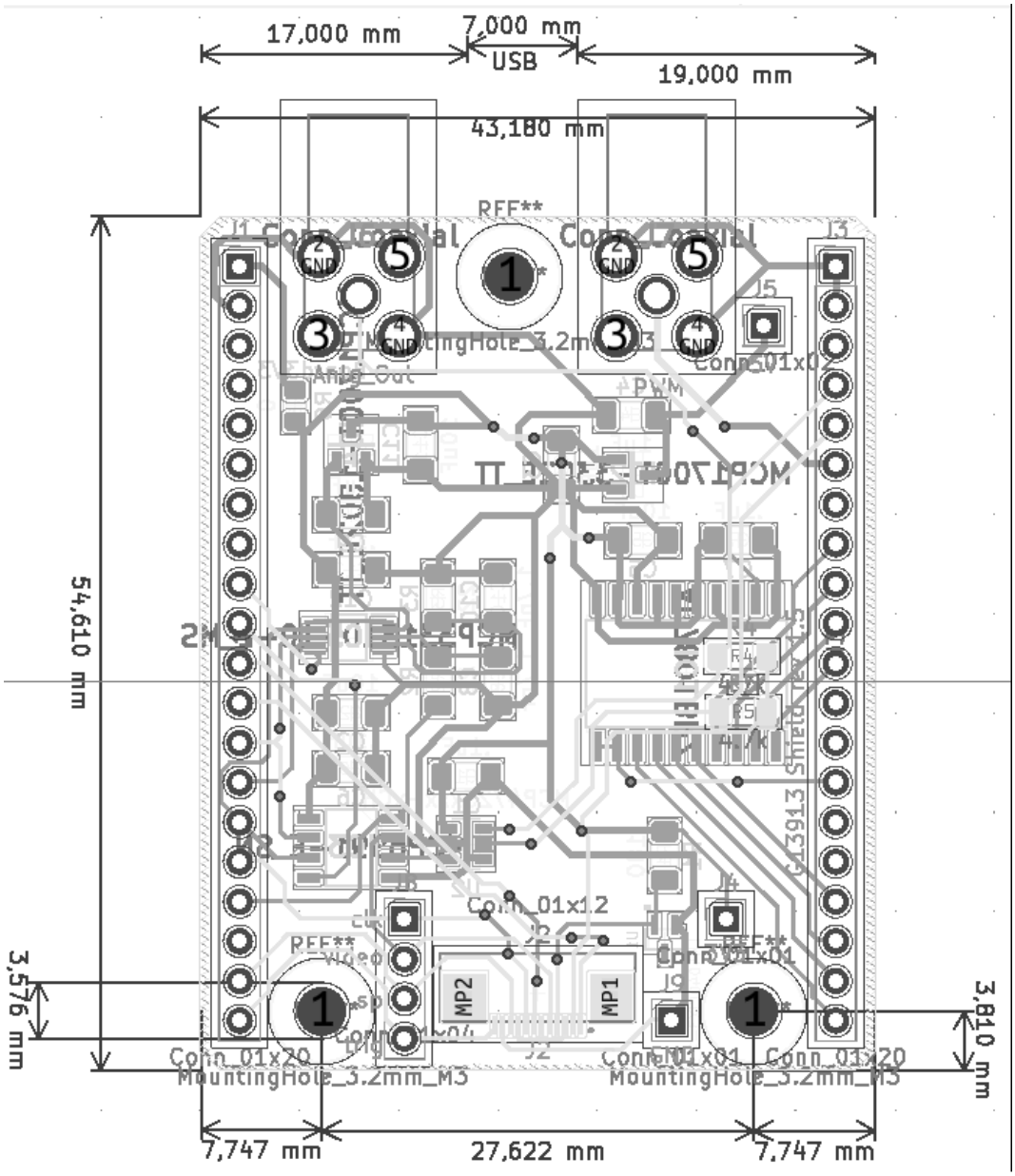


Figura 37 – Planificação da PCB, com a posição dos componentes, ligações e anotações.

Pinos para teste e depuração

No desenho da placa foram acrescentados pinos para efeitos de teste, acedendo-se facilmente a sinais importantes do sensor de imagem como os que se podem ver na Figura 38, incluindo estes o sinal de relógio, de vídeo, ‘AD_sp’, ‘AD_trig’, a massa e a alimentação de 3,3 V.

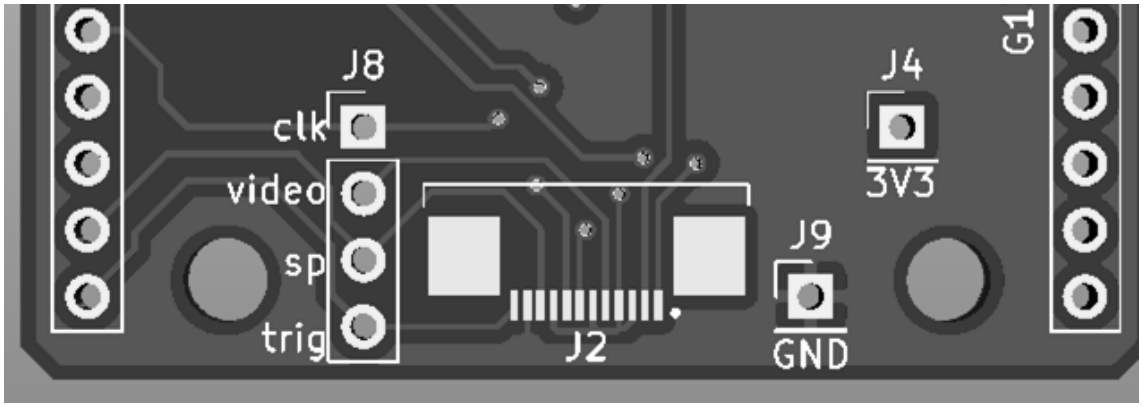


Figura 38 – Pinos J8, J9 e J4, perto do conector do sensor de imagem no fundo da PCB.

Através destes conectores foi possível introduzir um substituto para o sensor de imagem durante uma parte significativa do desenvolvimento do módulo de aquisição, ligando os sinais a um segundo microcontrolador, e testar o funcionamento dos conversores e do circuito de acondicionamento.

3.6. Teste do sistema sem sensor de imagem

Sumário

Com o intuito de testar o funcionamento do sistema antes da introdução do sensor de imagem, foi utilizado um microcontrolador Arduino Due para gerar os seguintes sinais:

- Sinal em rampa;
- Sinal sinusoidal em módulo com variação temporal;
- Sinal para teste de sincronismo ao nível dos pixéis;

Estes sinais foram escolhidos para efetuar testes a diferentes aspetos do funcionamento do sistema de aquisição. O sinal em rampa permite averiguar a resposta do circuito de acondicionamento e dos conversores analógicos-digitais sem variações temporais. O sinal sinusoidal permite observar a resposta do sistema a um sinal variante no tempo, tendo sido útil no desenvolvimento da representação gráfica temporal da aplicação para PC. O sinal de teste de sincronismo permite averiguar se os canais apresentados se encontram alinhados aos pixéis correspondentes.

O sistema respondeu corretamente aos testes a uma frequência de operação de 50 kHz. Mais tarde, com o sensor de imagem, foi verificado um funcionamento correto até à frequência de 250 kHz. O *firmware* do controlador do ADC externo foi validado, mas este deixou de funcionar suspeitando-se de uma falha no *hardware* que não foi possível resolver até à data.

Sinais

Os sinais de vídeo são acompanhados por dois sinais auxiliares. Um sinal de pulso único (sinal 'AD_sp') antecede o início do sinal de vídeo, e os pulsos de sincronismo ('AD_trig') indicam os momentos para a amostragem. O sinal de vídeo gerado nesta solução pode ter uma frequência até cerca de 50 kHz. A Figura 39 mostra os sinais auxiliares.

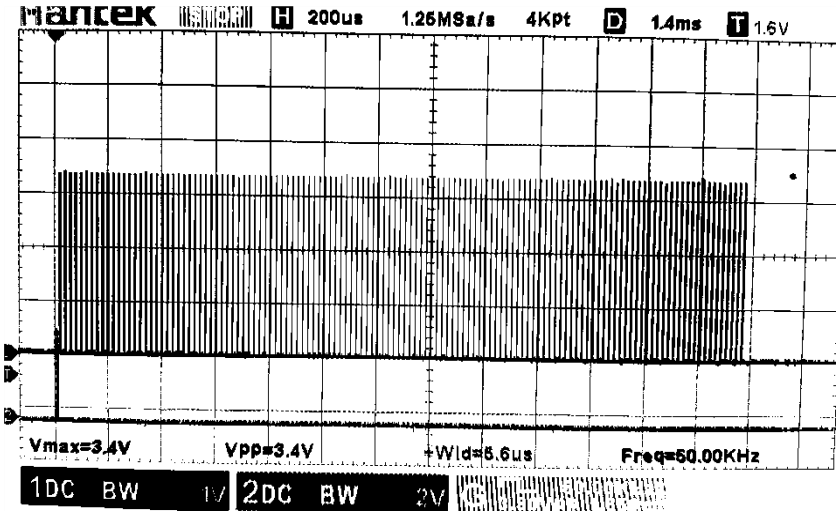


Figura 39 – Sinais auxiliares. No canal 1, os 128 pulsos de sincronismo e no canal 2, o pulso inicial.

A Figura 40 mostra o sinal de teste em rampa, com tensões entre 2,5 V e 0,54 V, sendo a tensão mínima limitada pela gama dinâmica do DAC do Arduino Due.

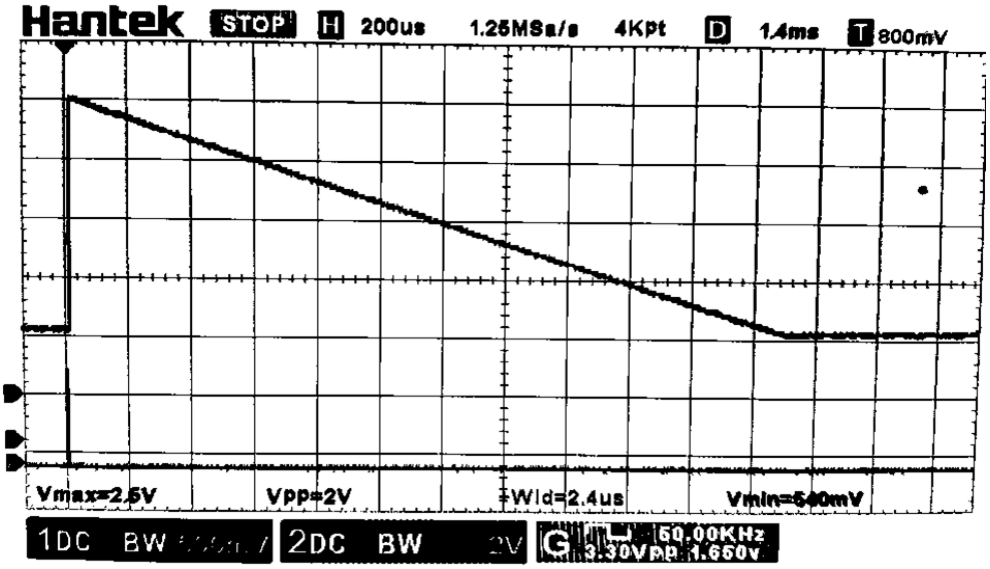


Figura 40 – Sinal de teste em rampa. Tensão máxima de 2,5 V e mínima de 0,54 V.

A Figura 41 mostra um sinal de teste sinusoidal em módulo, cuja fase varia ao longo do tempo.

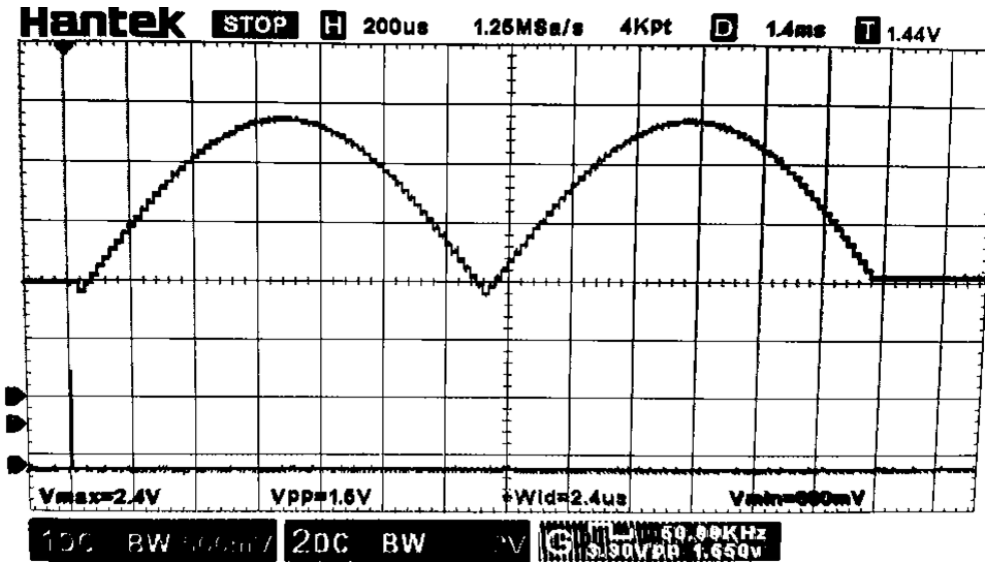


Figura 41 – Sinal de teste sinusoidal em módulo, com fase variável no tempo.

Por fim o sinal de teste na Figura 42 é constituído por níveis altos nos tempos correspondentes aos píxeis 1, 64 e 128, e níveis baixos nos restantes, servindo para testes de sincronismo.

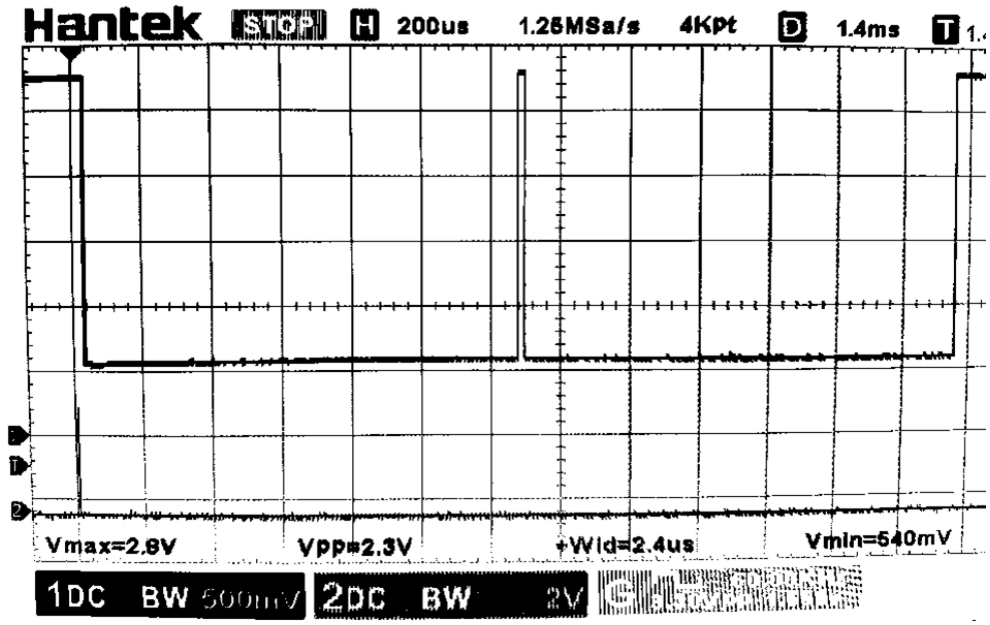


Figura 42 – Sinal de teste com nível alto nos tempos correspondentes aos píxeis 1, 64 e 128.

Resultados

As próximas imagens mostram alguns resultados de aquisições destes sinais de teste, utilizando o ADC interno do microcontrolador e a saída analógica. Estes verificaram o funcionamento correto do sistema como um todo.

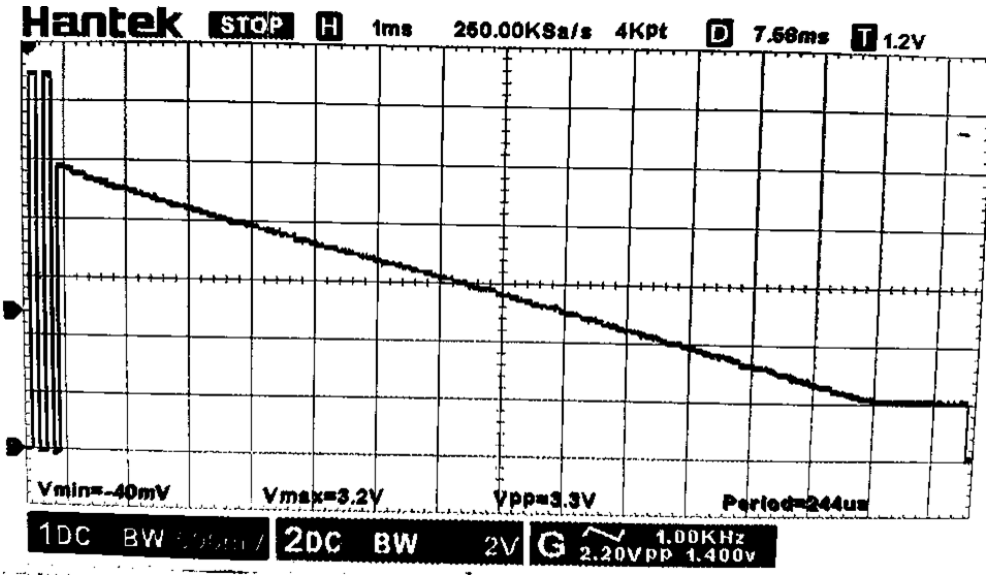


Figura 43 - Resultado do sinal de teste em rampa, com o circuito de acondicionamento configurado com ganho unitário.

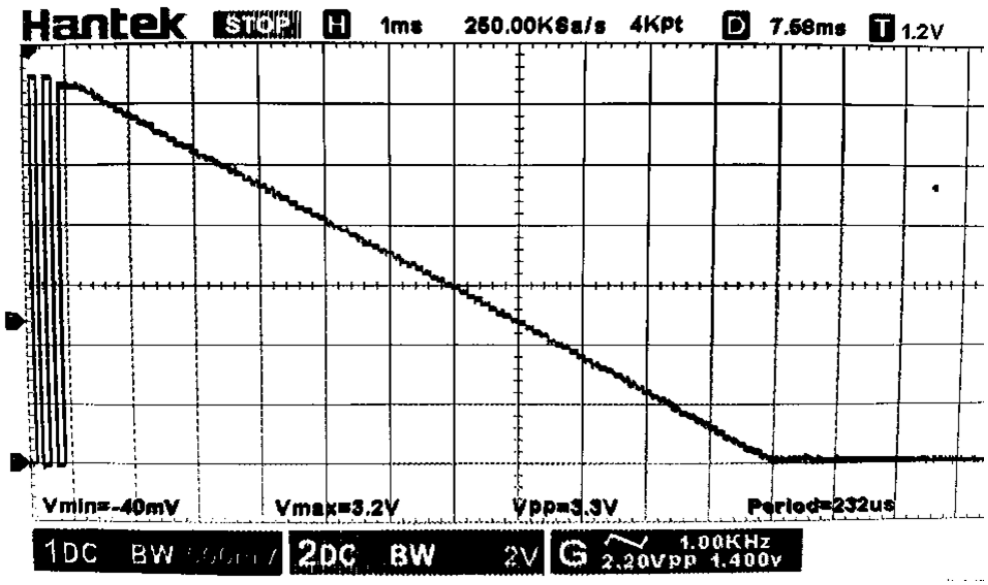


Figura 45 – Resultado do sinal de teste em rampa, com o circuito de acondicionamento configurado com ganho = 2.

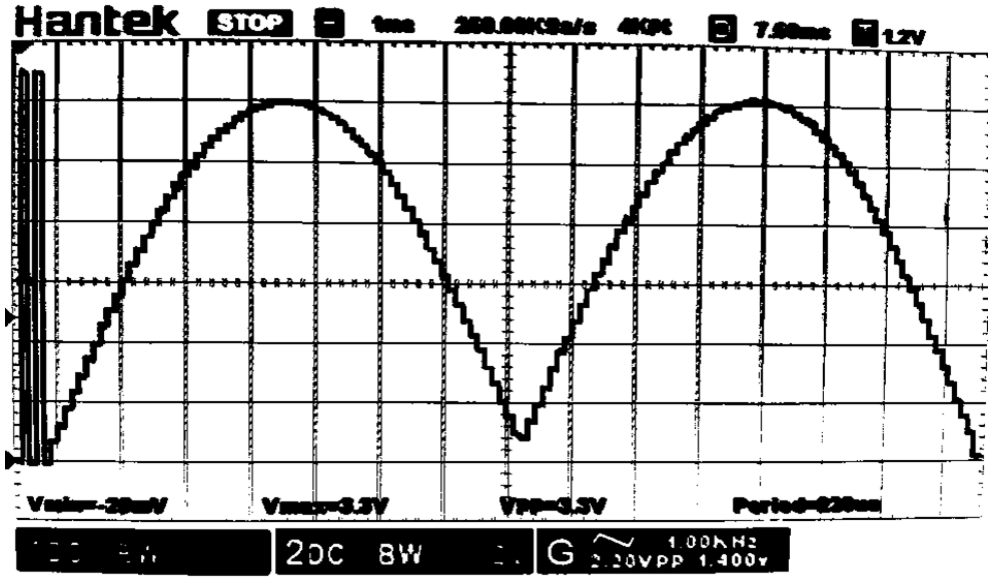


Figura 44 – Resultado do sinal de teste sinusoidal, com o circuito de acondicionamento configurado com ganho = 2.

O sinal de teste de sincronismo apresentou o resultado presente na Figura 46, contendo os níveis de tensão esperados para uma frequência de 50 kHz. Não foi possível testar frequências superiores com esta solução, contudo mais tarde após a montagem do sensor de imagem verificou-se um sincronismo correto até à frequência de 250 kHz. Esta terá sido limitada pela frequência de amostragem do ADC interno, e poderia ser aumentada recorrendo à configuração com DMA.

3222.75	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	3216.31
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	3216.31

Figura 46 – Resultado do teste de sincronismo em formato de texto, com níveis de tensão altos nos canais correspondentes aos pixéis 1, 64 e 128.

Os sinais continuaram a ter utilidade durante o desenvolvimento da aplicação, permitindo efetuar testes abrangentes a todo o sistema antes de introduzir o dispositivo de imagem CMOS.

A Figura 47 mostra a aquisição do sinal de teste em rampa a uma frequência de operação de 10 kHz, utilizando os dois conversores, durante o desenvolvimento da aplicação para PC. O sinal obtido com o ADC externo encontra-se a azul, mostrando um ligeiro deslocamento de nível relativamente ao sinal obtido com o ADC interno.

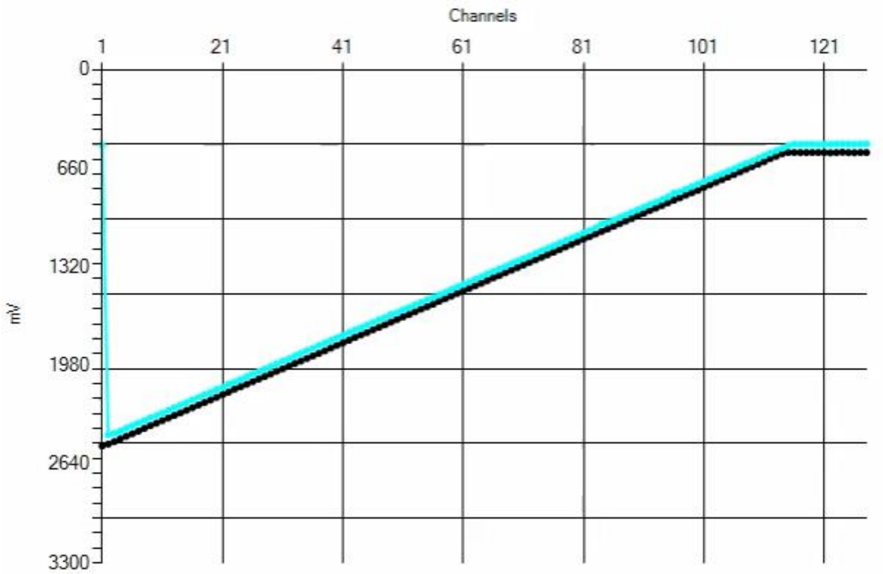


Figura 47 – Aquisição do sinal de teste em rampa na aplicação em desenvolvimento. A preto, o sinal convertido com o ADC interno, a azul, o sinal convertido com o ADC externo.

Contudo, o ADC externo deixou de funcionar, suspeitando-se de uma falha no *hardware*. Poderá ser necessário a montagem de uma nova placa para resolver o problema sem correr o risco de danificar o protótipo existente. Durante o restante desenvolvimento, e até à data, a utilização do sistema restringiu-se ao ADC interno com menos 3 bits de resolução, mas sem perda de funcionalidade devido à redundância.

4. Aplicação para PC

A implementação do protocolo SCPI utilizada no microcontrolador que gere o módulo de aquisição permite o seu manuseamento através de uma aplicação, enviando os comandos através da comunicação série virtual que decorre através da porta USB. Com esse intuito foi desenvolvida uma aplicação com interface gráfica para o sistema operativo Windows.

4.1. Sumário

A aplicação pretende fornecer ao utilizador uma interface gráfica para utilizar o módulo de aquisição de forma intuitiva, dispondo de controlos para configuração do dispositivo e para obtenção de imagem, com várias opções de representação dos dados. Em conjunto com o módulo de aquisição, esta constitui uma solução completa de aquisição de sinal com as seguintes funcionalidades:

- Ligação ao detetor ótico multicanal via comunicação série através da porta USB;
- Configuração e consulta de todos os parâmetros do sistema abrangidos pelos comandos SCPI apresentados anteriormente;
- Armazenamento e carregamento das configurações em ficheiro;
- Requisição de aquisições de imagem em modo singular ou em modo contínuo com uma taxa de aquisição até 100 imagens por segundo;
- Visualização dos dados em gráfico de linha ou em gráfico temporal, em vários formatos (intensidade relativa, milivolts ou escala logarítmica);
- Exportação e importação dos dados para ficheiros com extensão .csv.

O projeto de *software* inclui um pacote de instalação que assiste na instalação das dependências e adiciona a aplicação à lista de programas instalados. As dependências são essencialmente a versão 4.7.2. da *framework* .NET, uma plataforma desenvolvida pela Microsoft que se encontra amplamente difundida e é utilizada em sistemas e aplicações na plataforma Windows.

Os detalhes do desenvolvimento da aplicação encontram-se no Anexo D, juntamente com o código mais relevante.

Segue-se uma apresentação da aplicação, seguindo os passos necessários para a ligação do dispositivo, configuração, aquisição de imagem, visualização e exportação dos dados.

4.2. Funcionamento da aplicação

Ao correr o programa é apresentada a seguinte janela (Figura 48). Será necessário ligar o módulo de aquisição a uma porta USB do computador e conectar o dispositivo para as funcionalidades de configuração e aquisição ficarem disponíveis. Os resultados das medições serão apresentados no centro desta mesma janela.

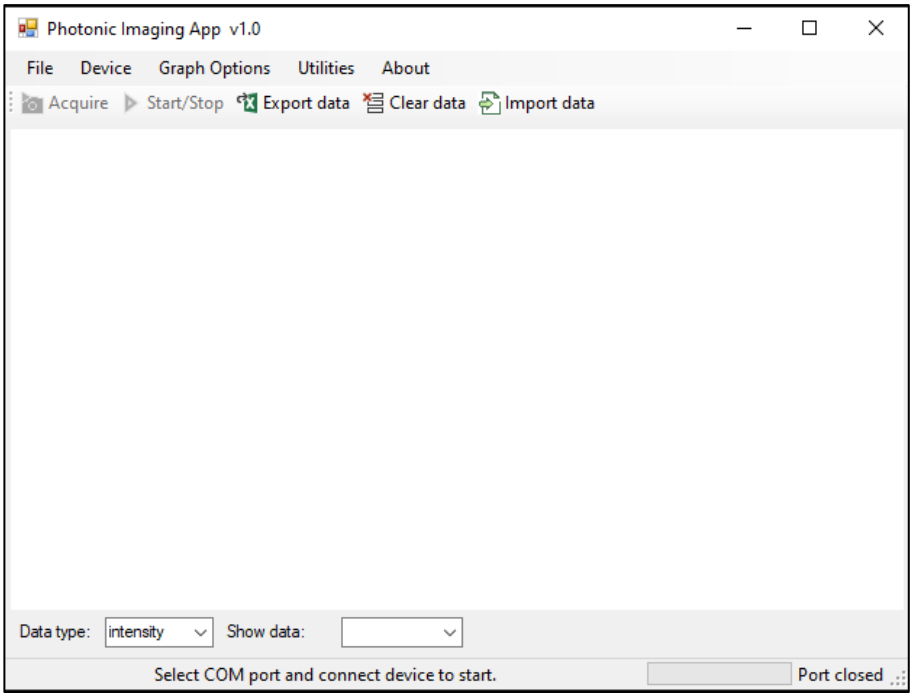


Figura 48 – Janela principal ao iniciar.

Através da barra de Menu é possível aceder a todas as funcionalidades, consoante o estado do dispositivo de aquisição. Na Figura 49 apresentam-se os conteúdos dos menus.

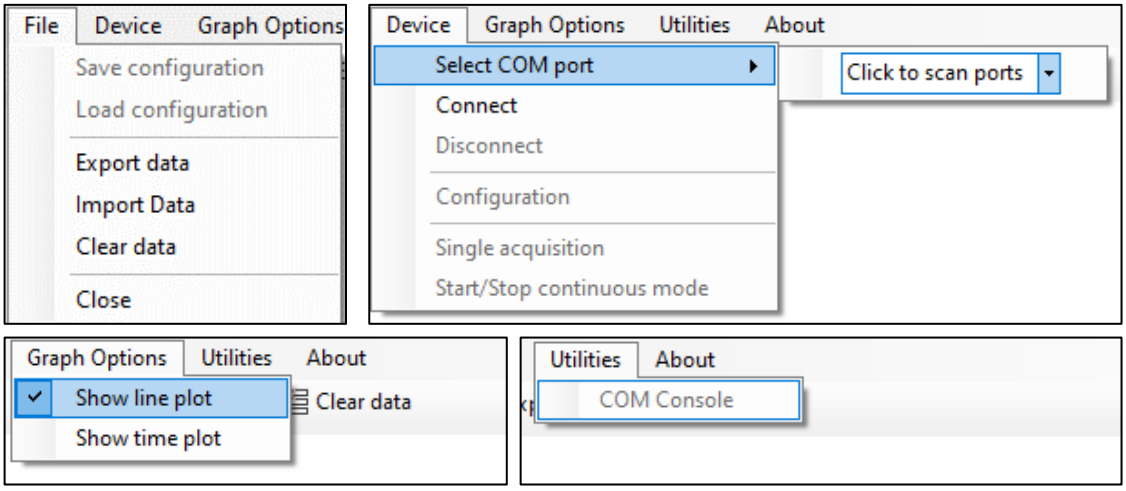


Figura 49 – Conteúdo dos itens na barra de Menu.

No menu 'Device' é feita a seleção da porta COM e a ligação com o dispositivo. Ao conectar o dispositivo com sucesso aparece uma janela com a sua *string* de identificação, resultante do comando `*IDN` (Figura 50).

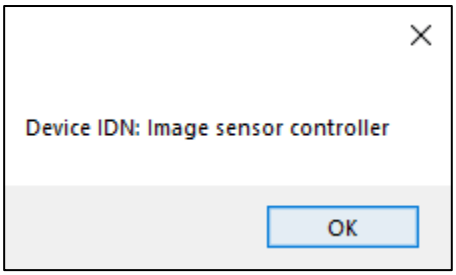


Figura 50 – Janela com texto de identificação do dispositivo.

No caso da primeira tentativa de conexão não ser bem sucedida, é enviado um comando de *reset* e é feita uma segunda tentativa. Ao conectar o dispositivo, a definição padrão da aplicação é aplicada. Na barra de estado é visível uma barra de progresso, assim como as mensagens de estado e a identificação da porta COM aberta (Figura 51).

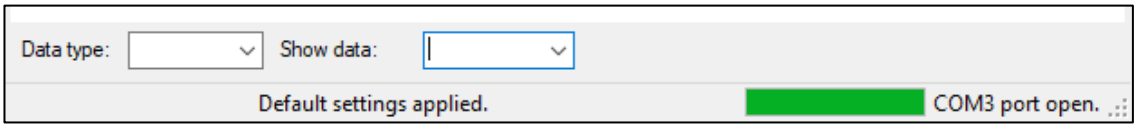


Figura 51 – Barra de estado, com barra de progresso e estados da aplicação e da porta COM.

Quando o dispositivo está conectado, as operações de configuração, aquisição e carregamento de ficheiros de configuração passam a estar disponíveis.

A Figura 52 e a Figura 53 mostram as abas da janela de configuração. Nestas são disponibilizados os parâmetros do sistema, e qualquer alteração é aplicada de imediato. Também é possível aplicar rapidamente a configuração padrão ou forçar todos os parâmetros e confirmar com as respostas do dispositivo. Estas operações indicam o progresso na barra de estado. As configurações poderão ser guardadas num ficheiro através do menu 'File' da janela principal, para serem carregadas mais tarde.

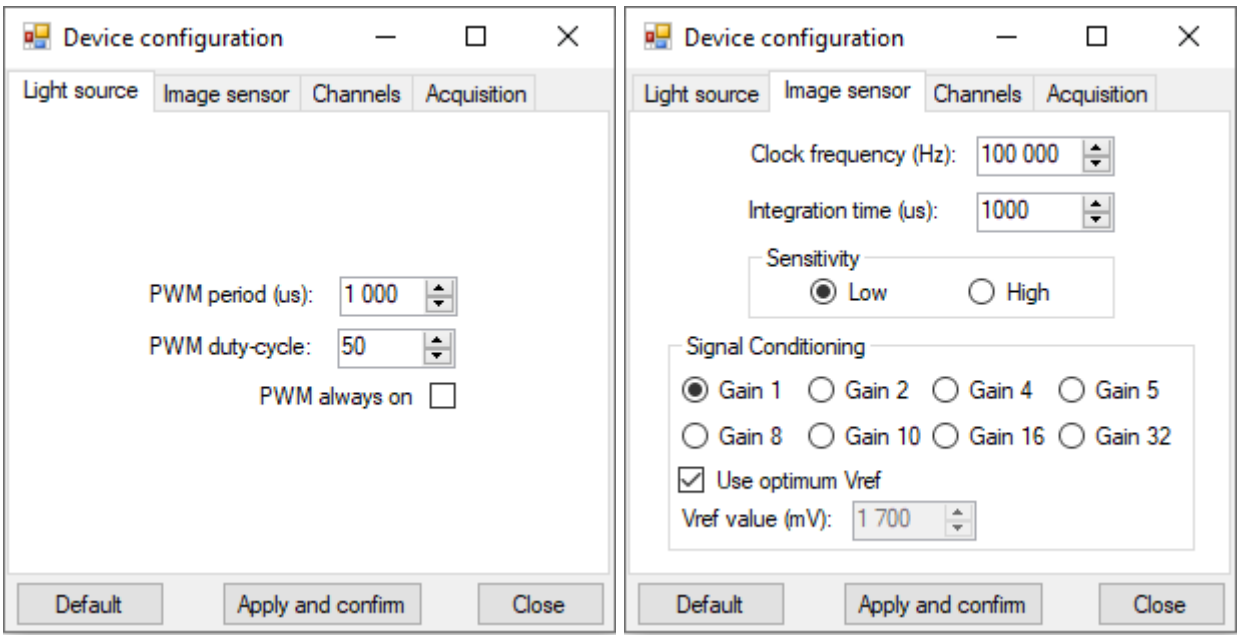


Figura 52 – Primeiras duas abas da janela de configuração.

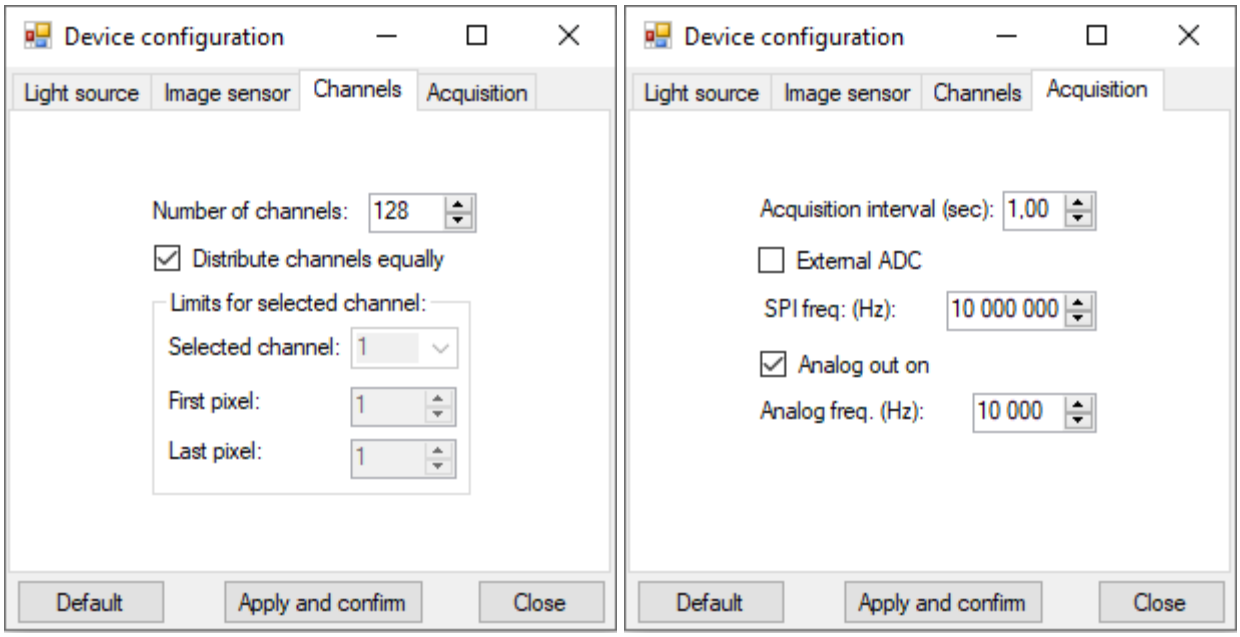


Figura 53 – Terceira e quarta abas da janela de configuração.

Com o dispositivo conectado, a barra de ferramentas permite rapidamente executar as operações de aquisição, única ou no modo contínuo (Figura 54). A exportação, importação ou a eliminação dos dados também está acessível de imediato.

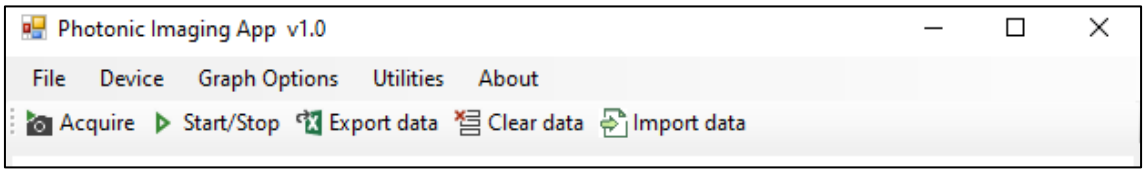


Figura 54 – Barra de ferramentas com o dispositivo ligado.

A Figura 55 mostra o resultado da aquisição com o sensor de imagem, com o índice '3', apanhando apenas luz ambiente. O gráfico mostra os valores em termos de intensidade relativa. Pousando o ponteiro do rato em cima de um ponto é mostrada uma legenda com o número do canal e o valor lido.

A Figura 56 mostra um gráfico temporal com uma sequência de aquisições (obtidas clicando várias vezes no botão de aquisição). Com o ponteiro do rato é possível obter mais informações sobre cada valor, incluindo o momento em que foi obtido. Nas duas figuras é obtida a informação sobre o mesmo conjunto de dados.

A Figura 57 mostra um novo conjunto de aquisições, obtidas utilizando o modo de aquisição contínuo.

Havendo um conjunto de dados na aplicação, a operação de exportar dados é feita com auxílio de uma janela de diálogo onde se escolhe a localização e o nome do ficheiro resultante, acrescentando a extensão '.csv', como se pode ver na Figura 58.

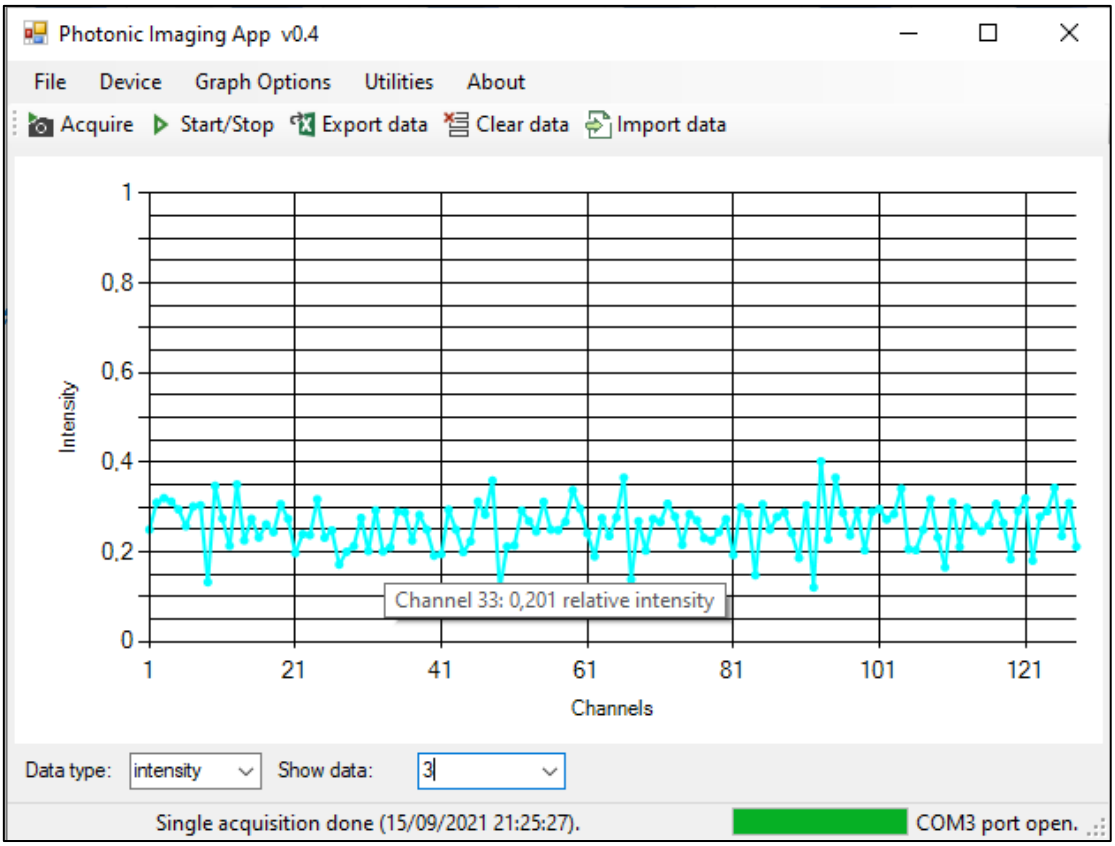


Figura 55 – Representação gráfica em linha.

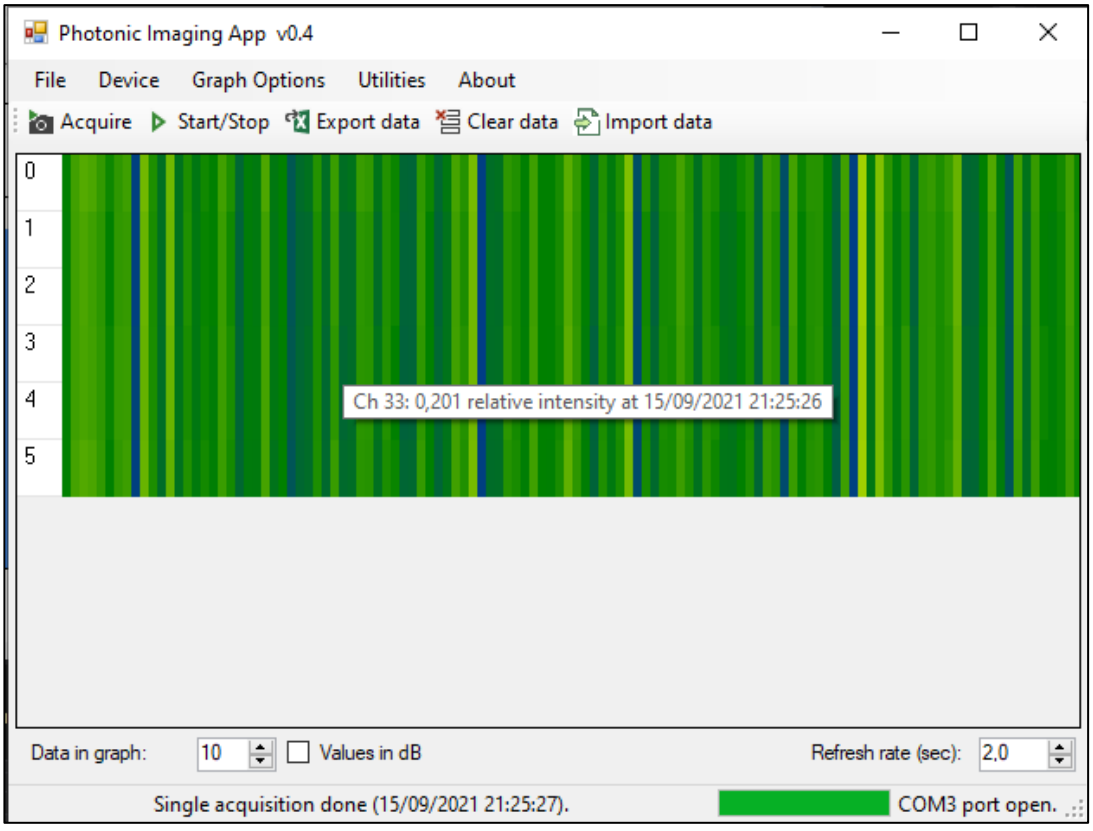


Figura 56 – Representação gráfica temporal.

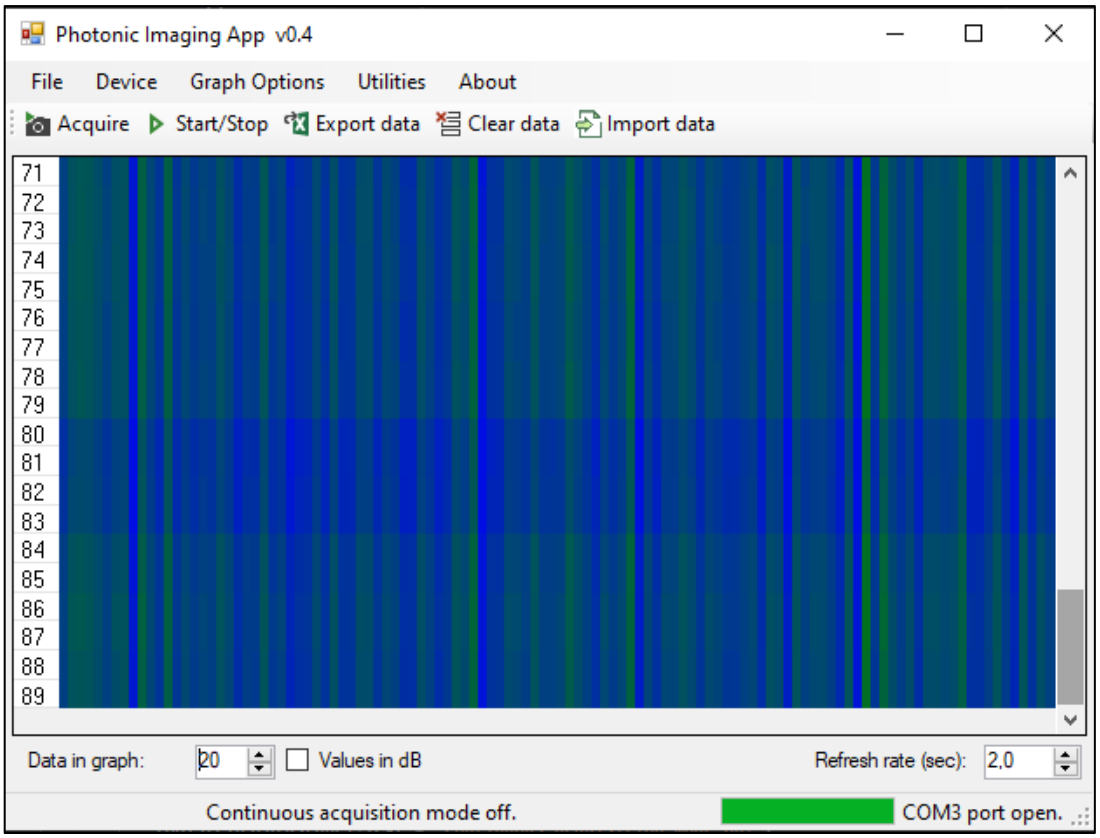


Figura 57 – Gráfico temporal mostrando aquisições em modo contínuo.

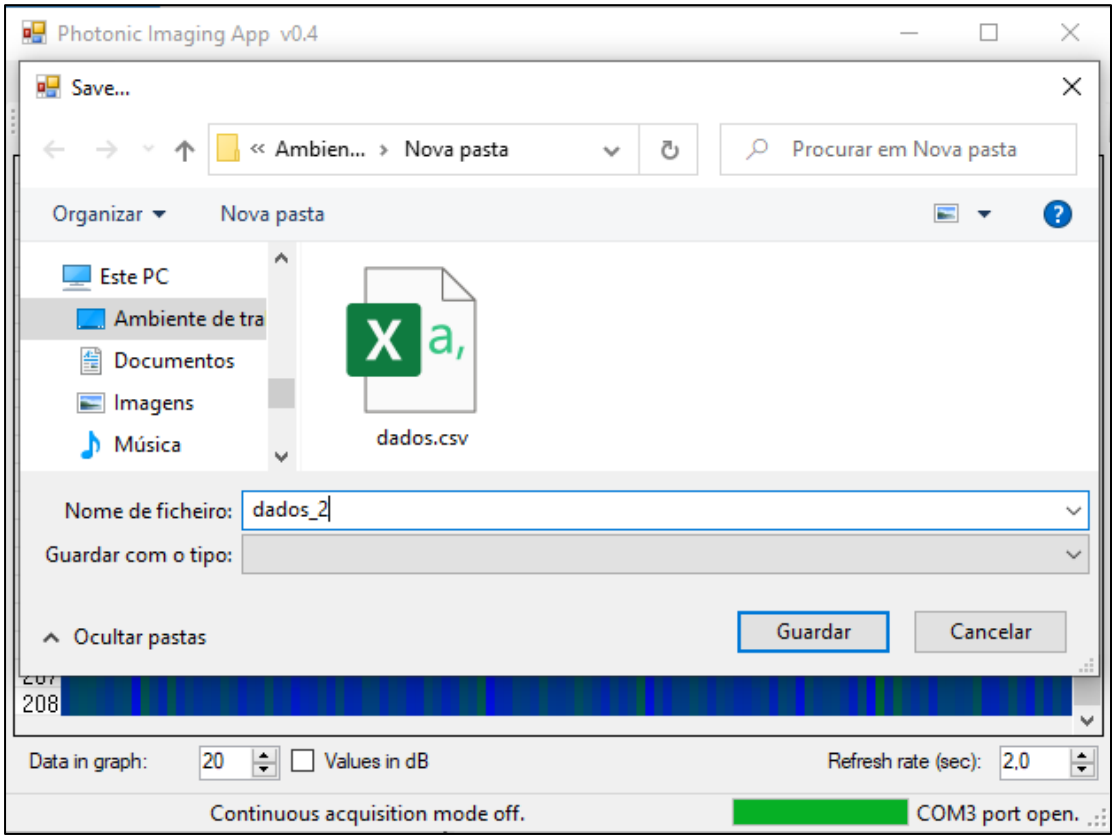


Figura 58 – Janela para guardar dados.

A operação de eliminação dos dados resulta numa janela de confirmação (Figura 59). Ao fechar a aplicação, também é apresentada uma janela semelhante para evitar perder dados importantes.

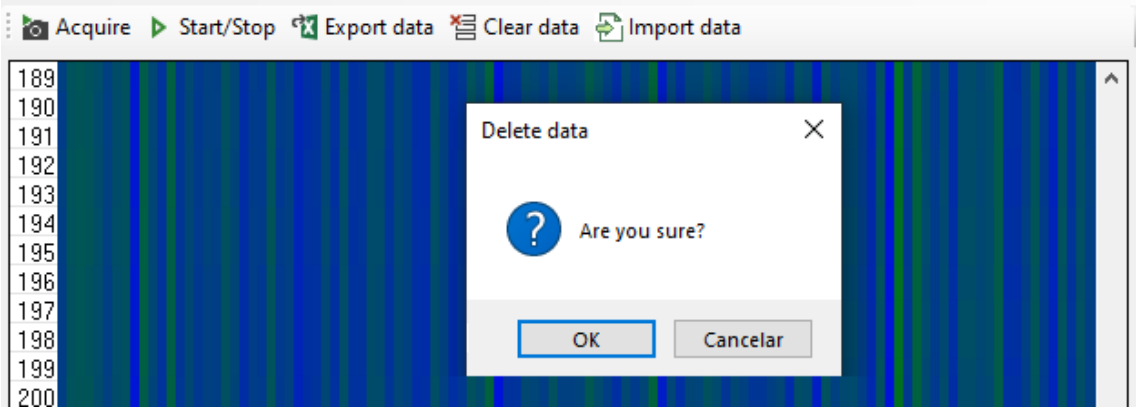


Figura 59 – Janela de confirmação para eliminação dos dados.

Na Figura 60 mostra-se a janela da comunicação série através da qual é possível enviar e receber mensagens diretamente ao dispositivo.

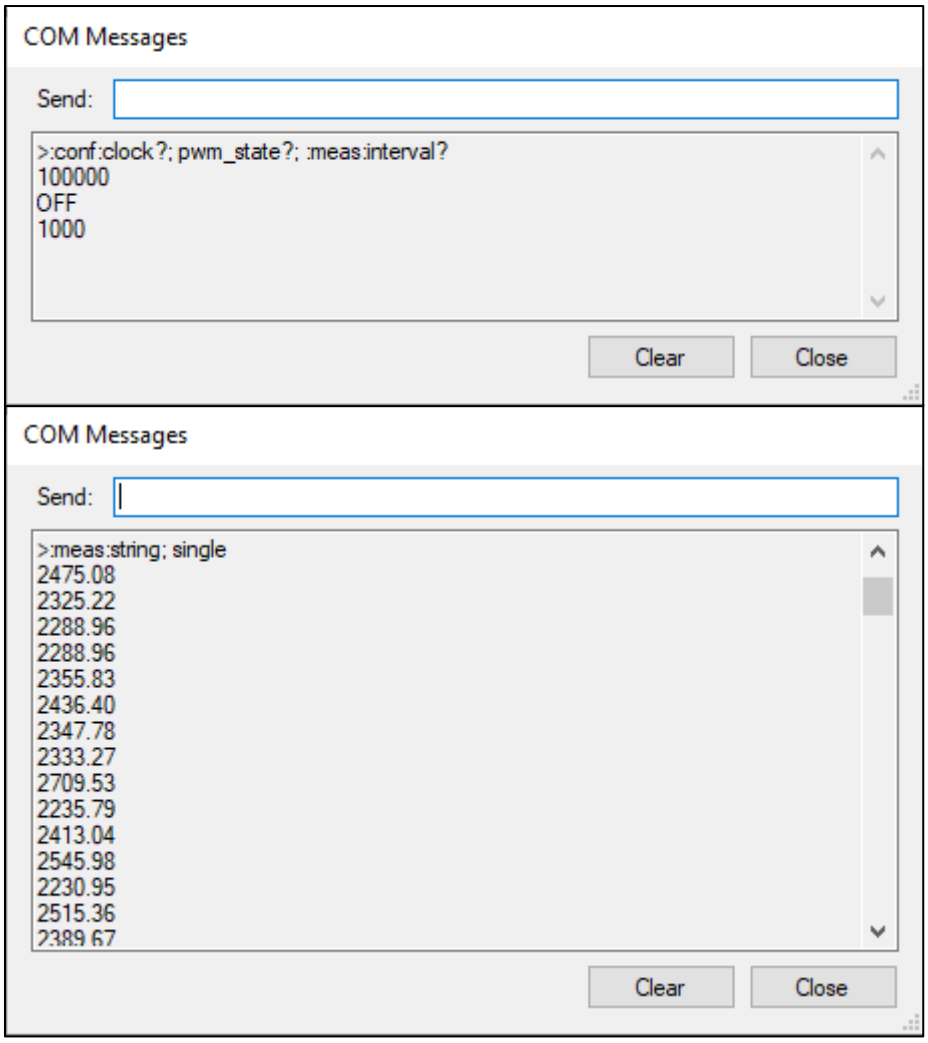


Figura 60 – Janela de mensagens COM. Em cima, foram enviados comandos SCPI ao microcontrolador, obtendo-se as respetivas respostas. Em baixo, o resultado de uma aquisição de sinal em formato de texto.

4.3. Visualização de sinais de teste

Nas figuras seguintes ilustram-se os resultados obtidos para alguns sinais de teste, mostrando o funcionamento correto do sistema completo e exemplificando algumas opções de representação gráfica.

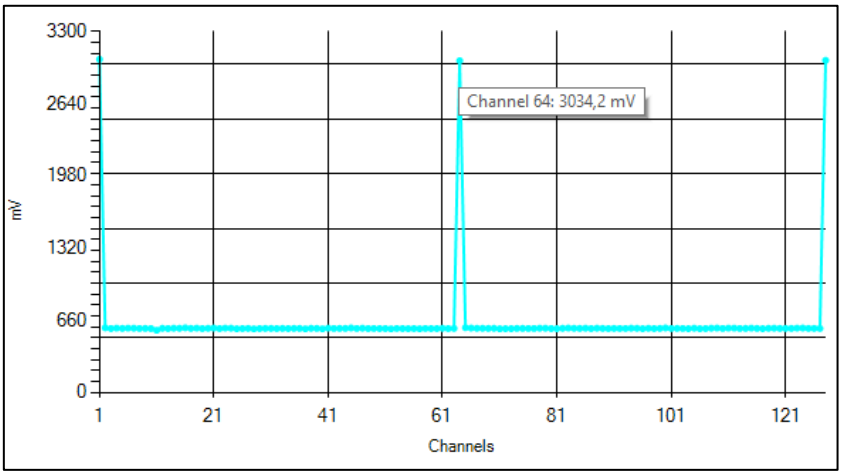


Figura 61 – Aquisição do sinal de teste de sincronismo.

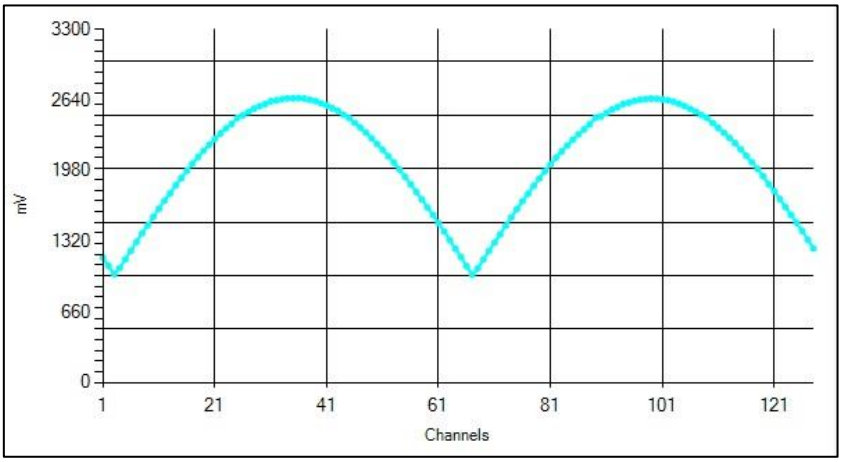


Figura 62 – Aquisição do sinal de teste sinusoidal.

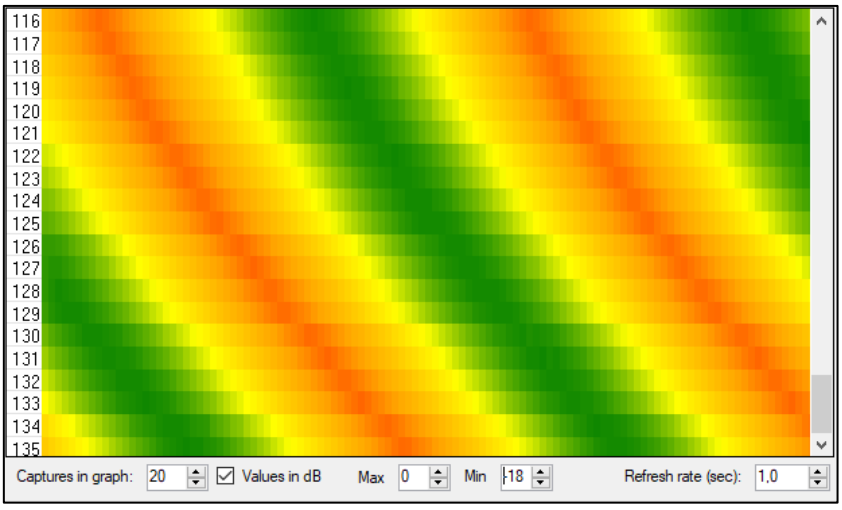


Figura 63 – Aquisição do sinal de teste sinusoidal ao longo do tempo.

5. Testes laboratoriais

Com o intuito de caracterizar o sistema de aquisição em termos da intensidade ótica que pode medir, da responsividade em diferentes pontos do espetro, da linearidade da resposta ao longo da linha de pixéis e da taxa de amostragem, foram efetuados alguns ensaios no laboratório recorrendo ao um laser IR e a um laser vermelho.

Sumário

Dos ensaios com os dois lasers foi possível determinar que a maior intensidade ótica que o módulo de aquisição consegue detetar sem saturar é inferior a 20,37 mW/cm^2 no comprimento de onda de 1550 nm. Apesar do sensor estar especificado para um funcionamento entre os comprimentos de onda desde 900 nm até 1700 nm, ainda foi possível obter sinal num comprimento de onda de 650 nm. Os varrimentos efetuados ao longo da linha de pixéis mostraram intensidades aparentemente constantes na observação dos gráficos gerados. Foi verificada uma taxa de amostragem até 100 Hz, com uma ligeira divergência face ao valor configurado.

Descrevem-se de seguida os procedimentos efetuados em laboratório, desde a preparação prévia até aos ensaios propriamente ditos e os seus resultados.

5.1. Preparação

A experimentação com o módulo de aquisição requer uma montagem em laboratório com um laser no comprimento de onda de 1550 nm. Numa primeira abordagem, procurou-se verificar o funcionamento correto do laser, fazendo uma montagem com um fotodíodo de InGaAs.

O laser foi controlado com um módulo Kinesis da ThorLabs, que dispõe de um programa de computador onde é possível aplicar a corrente ou a potência pretendida, assim como ligar e desligar o dispositivo e controlar outros aspetos relacionados com o seu funcionamento, como a polarização.

O acoplamento do fotodíodo com o feixe emitido pelo laser foi efetuado através de uma fibra ótica. Os componentes foram fixados em suportes ajustáveis em três eixos numa mesa de montagem, como mostra a Figura 64.

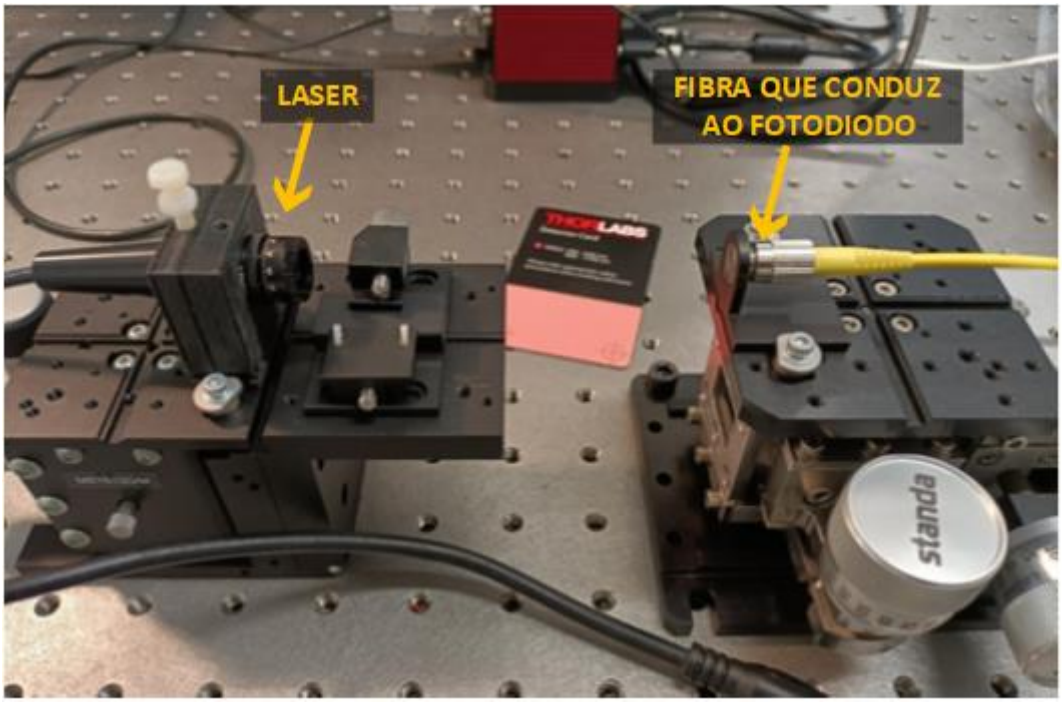


Figura 64 – Montagem do laser de 1550 nm com fotodíodo utilizando suportes ajustáveis.

O fotodíodo foi ligado a um amplificador de transimpedância que converte a fotocorrente gerada para uma tensão observável no osciloscópio. A Figura 65 mostra a caixa que contém o circuito de amplificação.

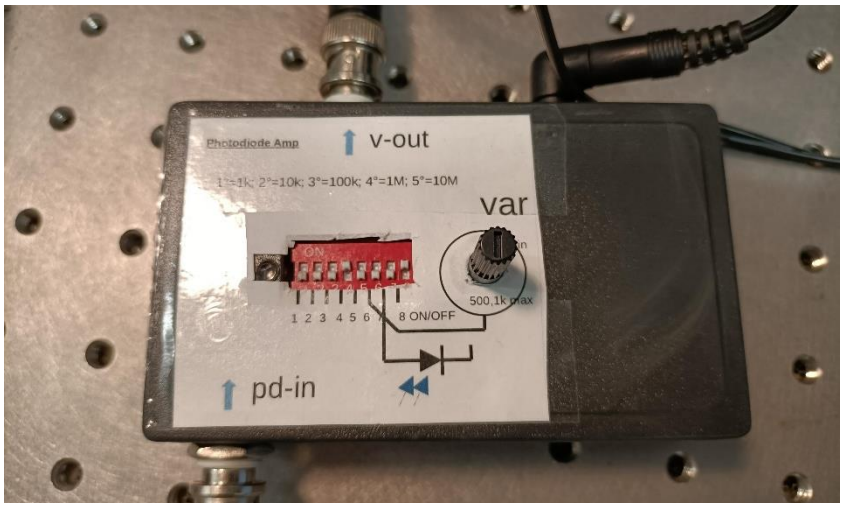


Figura 65 – Caixa do amplificador de transimpedância com várias opções de ganho.

Num primeiro alinhamento do feixe recorreu-se a um cartão de deteção, que numa superfície fotossensível emite luz visível a partir de radiação incidente no espetro infravermelho, para colocar o sinal visivelmente perto da entrada da fibra ótica. De seguida utilizou-se a macro, e por fim, a microafinação dos suportes da montagem, até se obter um máximo de tensão no osciloscópio. A caixa do laser possui uma lente ajustável de colimação, que foi também ajustada para melhorar o acoplamento do sinal.

Obtiveram-se as medidas de tensão para uma sequência de valores de corrente no laser, aplicadas através da interface gráfica. Estas resumem-se na seguinte tabela.

Tabela 9 – Valores medidos no osciloscópio para as correntes aplicadas no laser.

I_{laser} [mA]	V_o [mV]	I_{laser} [mA]	V_o [mV]	I_{laser} [mA]	V_o [mV]
0	102	10	560	16	1160
5	105	11	660	17	1260
6	155	12	766	18	1370
7	252	13	860	19	1480
8	352	14	960	20	1580
9	445	15	1060		

O laser apresentou uma corrente de limiar de cerca de 5 mA, abaixo da qual não se detetou qualquer potência ótica emitida. A corrente máxima estabelecida na aplicação de controlo do laser foi de 20 mA.

O módulo de controlo do laser permite a sua modulação por um sinal externo através da aplicação de tensões entre 0 e 10 V, correspondendo aos limites da corrente nula e da corrente máxima definida, respetivamente. A funcionalidade de modulação poderá ser aproveitada testar a taxa de aquisição de imagem do sistema com um sinal variável no tempo.

Uma modulação bem sucedida do laser foi observada no osciloscópio aplicando um sinal sinusoidal com mínimo de 0 V e máximo de 10 V, a uma frequência de 10 Hz. A Figura 66 mostra os sinais observados.

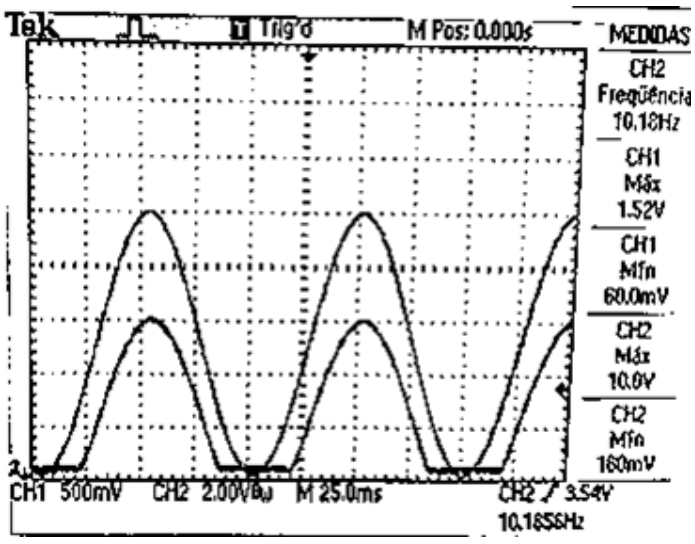


Figura 66 – Sinal modulador aplicado no módulo do laser e sinal detetado com o fotodíodo, observados no osciloscópio.

Note-se que a gama de tensões entre 0 e 10 V irá corresponder a uma gama de correntes no laser de 0 a 20 mA, resultando numa tensão constante no sinal detetado para correntes inferiores à corrente de limiar, 5 mA. A Figura 67 mostra a deteção de uma sinusoide completa com um nível de amplitude à escala da unidade de volt, para uma modulação com um sinal entre cerca de 2,5 V e 10 V.

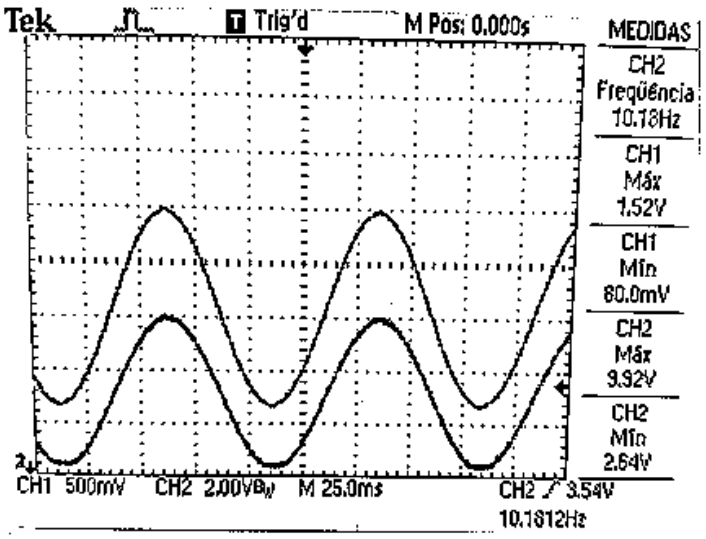


Figura 67 – Modulação do laser evitando correntes no laser inferiores à de limiar de emissão.

Aumentou-se a frequência de modulação para 100 Hz e depois 1 kHz, observando-se os sinais da Figura 68. Verifica-se que o sistema responde suficientemente depressa para uma modulação a 100 Hz, com um ligeiro desfasamento. À frequência de 1 kHz observou-se uma atenuação considerável no sinal detetado.

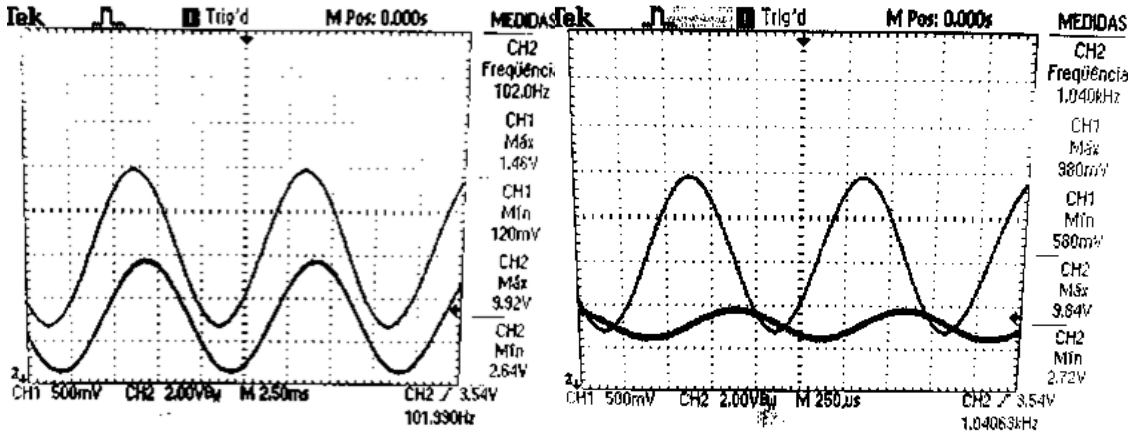


Figura 68 – Sinais observados no osciloscópio para uma frequência de modulação de 100 Hz à esquerda, e de 1 kHz à direita.

Os ensaios feitos mostraram o funcionamento correto de todos os componentes, podendo-se dar sequência aos ensaios para a caracterização do sistema de aquisição.

5.2. Caracterização do detetor ótico

Laser IR (1550 nm)

A incidência direta do feixe do laser sobre o sensor de imagem leva facilmente o sinal de vídeo à saturação, encontrando os limites da potência máxima que o sensor de imagem pode medir. O módulo de controlo do laser permite o ajuste da potência em passos de 10 µW, correspondendo a uma corrente mínima no laser de cerca de 5,6 mA.

Para minimizar a saturação, reduziu-se o tempo de integração ao mínimo. Na utilização do módulo observou-se a perda esporádica de informação com frequências de operação superiores a 250 kHz, começando pela perda de apenas um pixel e agravando com o aumento da frequência. Determinou-se então uma frequência de operação máxima de 250 kHz e consequentemente um tempo de integração efetivo mínimo de 4 µs. Com esta configuração, assim como a opção de sensibilidade mínima e o circuito de acondicionamento em ganho unitário, efetuaram-se as seguintes medições.

Foi efetuado um varrimento da potência do laser entre 10 µW e 100 µW, obtendo-se a sequência de imagem ilustrada na Figura 69, numa representação gráfica em MATLAB dos dados exportados da aplicação. Ao longo do varrimento nota-se a abertura da largura do feixe do laser, ocupando desde cerca de 5 pixéis até cerca de 11.

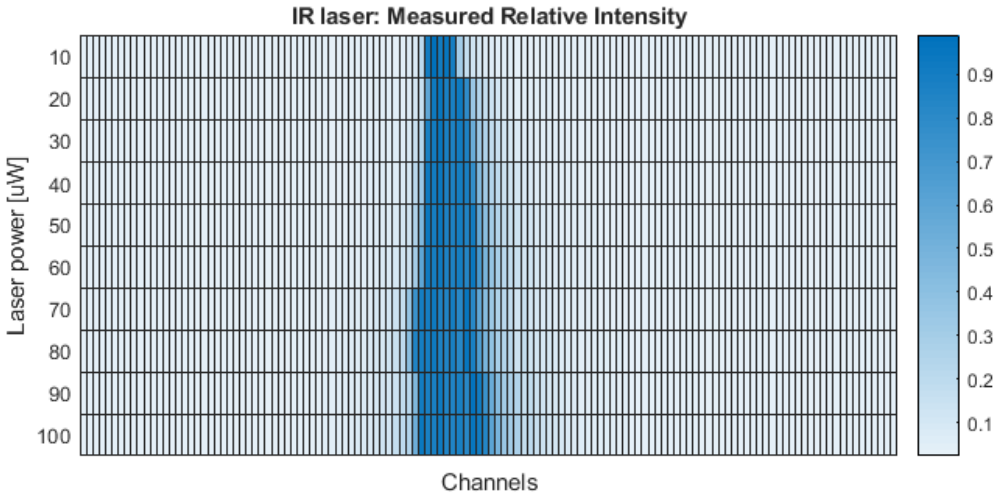


Figura 69 – Sequência de aquisições para um varrimento da potência do laser infravermelho em passos de 10 µW até 100 µW.

Os perfis do laser nos dois extremos do varrimento encontram-se nas Figura 70 e Figura 71, e é possível observar que o sinal se encontra em saturação, primeiramente, no centro, e depois, nas extremidades do perfil do feixe.

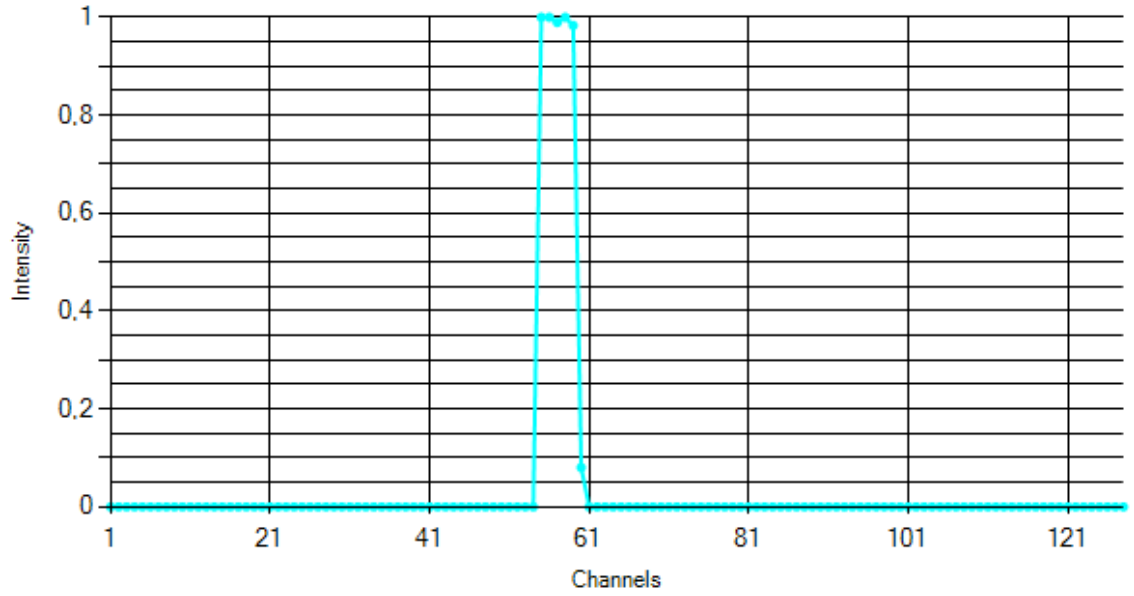


Figura 70 – Perfil do feixe do laser para uma intensidade de 10 µW, incidindo sobre 5 pixéis.

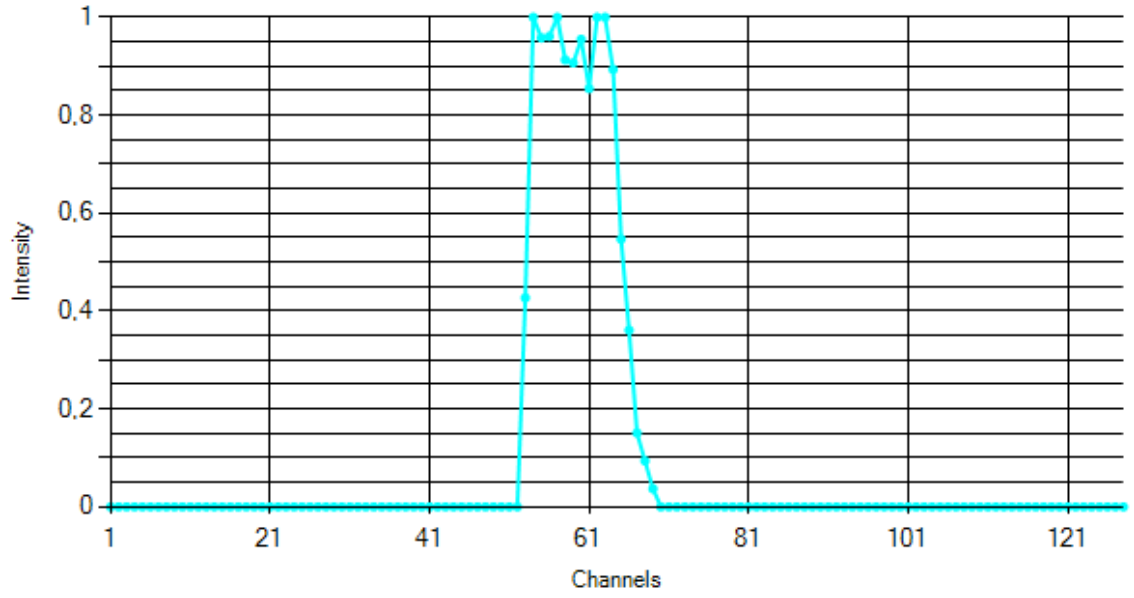


Figura 71 – Perfil do feixe do laser para uma intensidade de 100 µW, incidindo sobre cerca de 11 pixéis para uma intensidade relativa superior a 0,2.

O perfil inicial do feixe ocupa 5 pixéis, correspondendo a uma largura de 250 µm (50 µm por cada pixél). Considerando que o feixe do laser está centrado no sensor de imagem e que tem uma forma circular (com um raio de 0,0125 cm), e que os pixéis têm uma altura de 250 µm, pode-se considerar que todo o feixe incidiu sobre a janela do sensor, com uma intensidade de:

$$Intensidade\ ótica = \frac{Potência\ incidente}{Área\ do\ feixe} = \frac{0{,}01\ \mathrm{mW}}{\pi\ (0{,}0125\ cm)^2} = 20{,}37\ \mathrm{mW/cm^2}$$

Pode-se concluir que ao comprimento de onda de 1550 nm, a maior intensidade ótica que o sensor de imagem pode medir é inferior a 20,37 mW/cm^2.

Efetuou-se de seguida um varrimento da posição do sensor de imagem relativamente ao laser em passos de cerca de 0,1 mm, utilizando o ajuste do suporte da montagem. A Figura 72 mostra a sequência de imagem obtida.

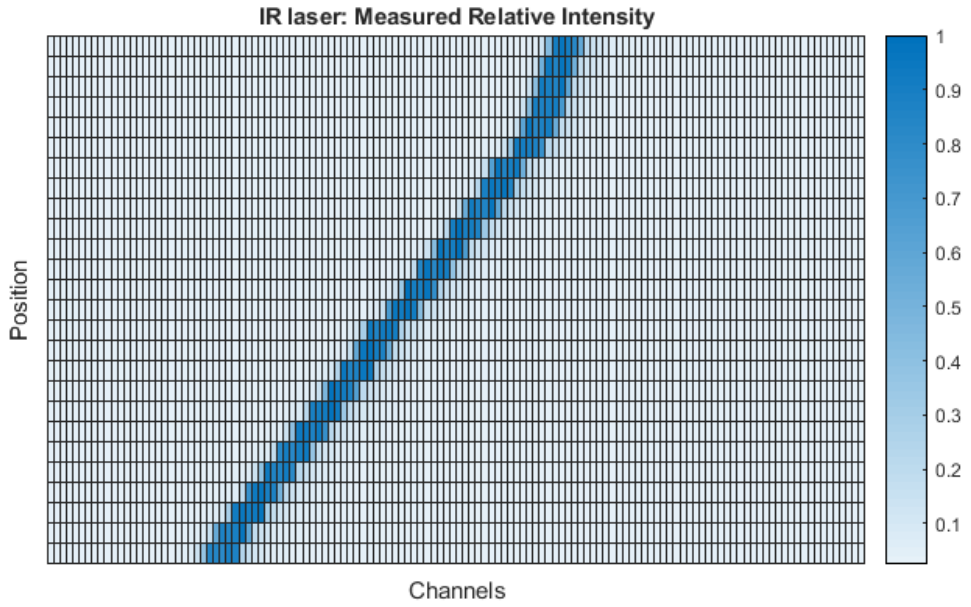


Figura 72 – Sequência de imagem obtida pelo varrimento da posição do sensor de imagem em passos de cerca de 0,1 mm relativamente ao laser infravermelho, com uma potência de 10 µW.

Os passos de 0,1 mm não se mostraram constantes nas extremidades permitidas pelo ajuste do suporte, variando de forma mais linear no centro do varrimento. Não se observaram variações consideráveis da sensibilidade do sensor nos pixéis percorridos pelo feixe do laser.

Por fim foi feita uma aquisição de imagem contínua a um feixe do laser modulado a 10 Hz, com uma taxa de amostragem de cerca de 100 amostras por segundo. A Figura 73 ilustra 30 amostras do conjunto de dados obtidos. Em cerca de 28 amostras observam-se 3 ciclos de modulação do sinal do laser, havendo um ligeiro desvio das 30 amostras esperadas para um sinal de 10 Hz amostrado à frequência de 100 Hz. Conclui-se que a taxa de amostragem efetiva para uma configuração a 100 Hz foi ligeiramente superior.

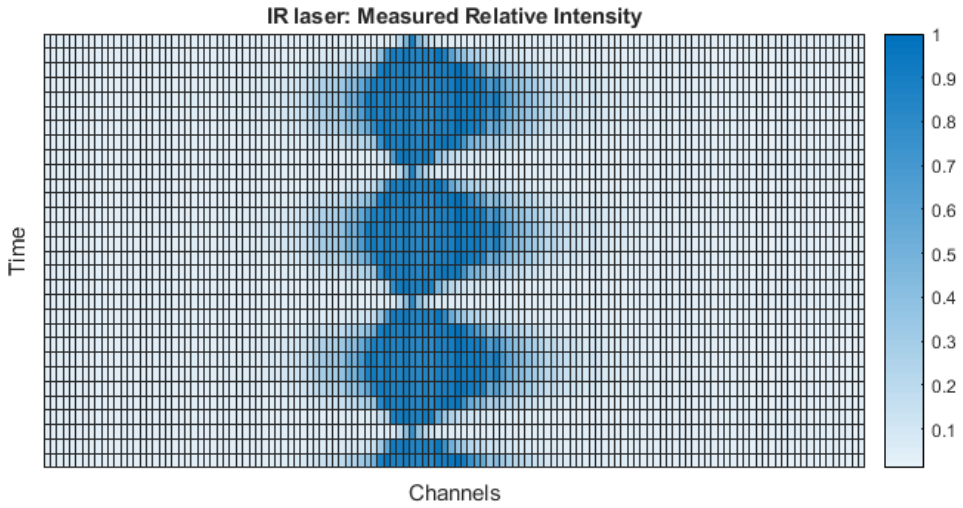


Figura 73 – Sequência temporal do laser modulado a 10 Hz com uma taxa de amostragem de cerca de 100 amostras por segundo.

Laser vermelho (650 nm)

O espectro de responsividade do sensor de imagem, segundo a Figura 21 que consta no capítulo 3.1.2, é mais relevante entre os comprimentos de onda de 900 nm e 1700 nm. No entanto, aumentando a sensibilidade e o tempo de integração do sensor, assim como tirando partido do circuito de acondicionamento, foi possível obter um sinal para um comprimento de onda de 650 nm. As configurações utilizadas encontram-se na Figura 74.

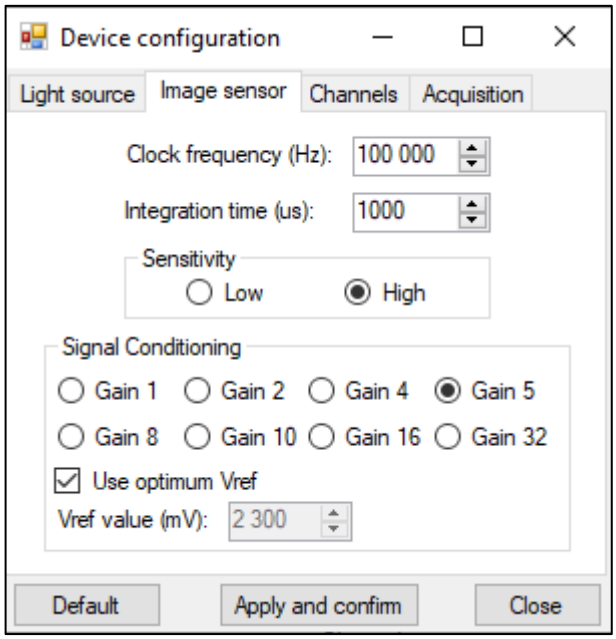


Figura 74 – Configuração do módulo de aquisição com uma maior sensibilidade e tempo de integração do que nos ensaios anteriores.

A Figura 76 mostra o perfil do centro do feixe do laser no plano horizontal para uma potência de 10 µW. O laser utilizado não possui uma lente de colimação ajustável, pelo que não foi possível otimizar a largura do feixe incidente no sensor de imagem. Não foi por isso possível fazer incidir toda a potência ótica sobre o sensor.

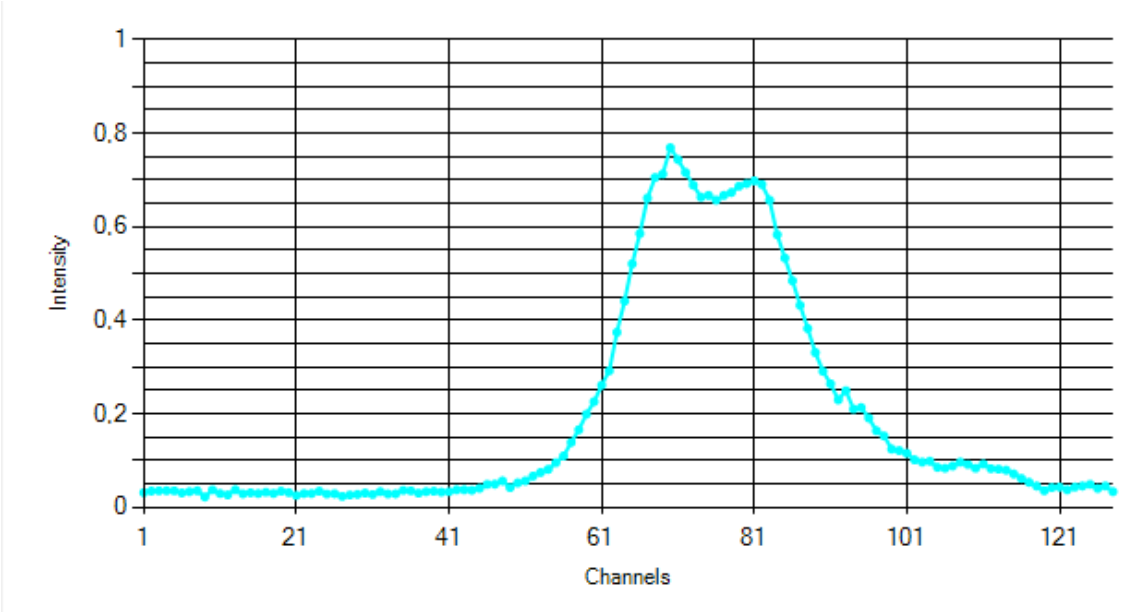


Figura 76 – Observação do perfil da intensidade do feixe emitido do laser vermelho no plano horizontal.

Através do módulo de controlo do laser, efetuou-se o varrimento decrescente da corrente a partir 20 mA, com passos de 1 mA. A Figura 75 mostra uma representação gráfica dos dados exportados pela aplicação. O sinal de vídeo não chegou a saturar.

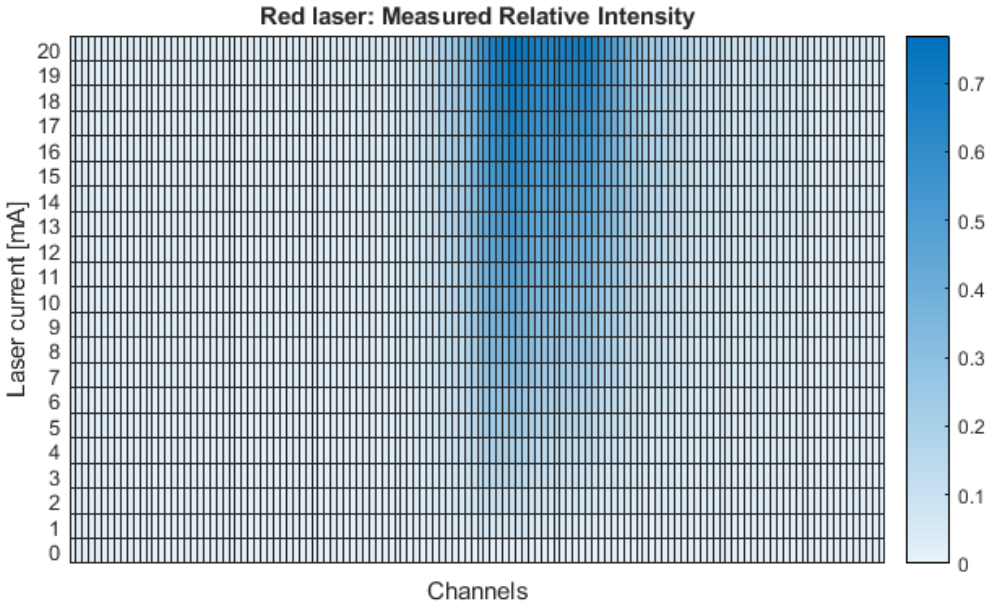


Figura 75 – Aquisição da sequência de imagem resultante do varrimento da corrente no laser vermelho.

Tal como anteriormente, foi realizado um varrimento da posição do sensor relativamente ao laser em passos de cerca de 0,1 mm. A Figura 78 mostra os dados obtidos. A intensidade relativa medida ao longo dos pixéis onde incidiu o feixe do laser novamente não apresentou desvios evidentes.

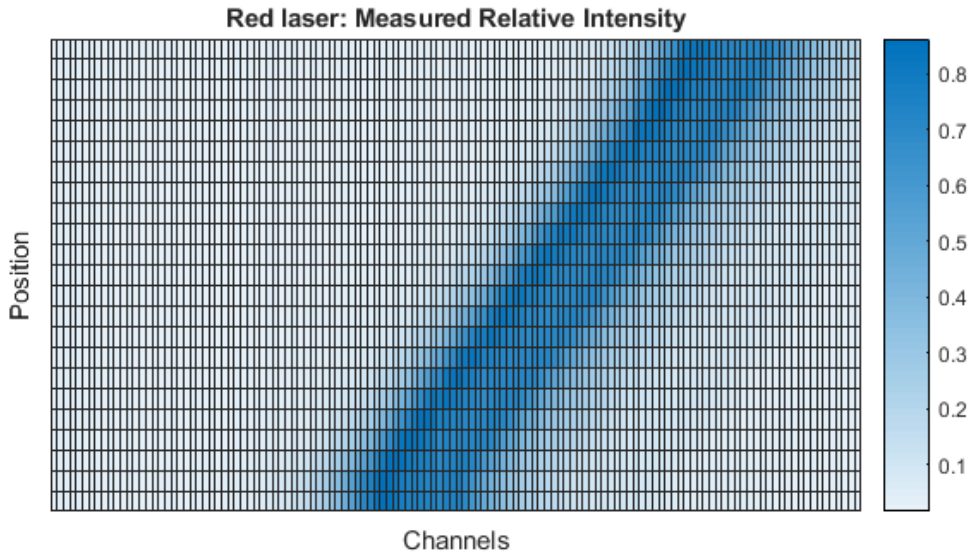


Figura 78 – Sequência de imagem resultante do varrimento da posição do sensor de imagem em relação ao laser vermelho.

Uma modulação idêntica do laser vermelho a 10 Hz resultou na sequência de imagem ilustrada na Figura 77. Tal como anteriormente, foi possível observar três ciclos do sinal modulado em cerca de 28 amostras, pelo que a frequência de amostragem foi ligeiramente superior aos 100 Hz configurados.

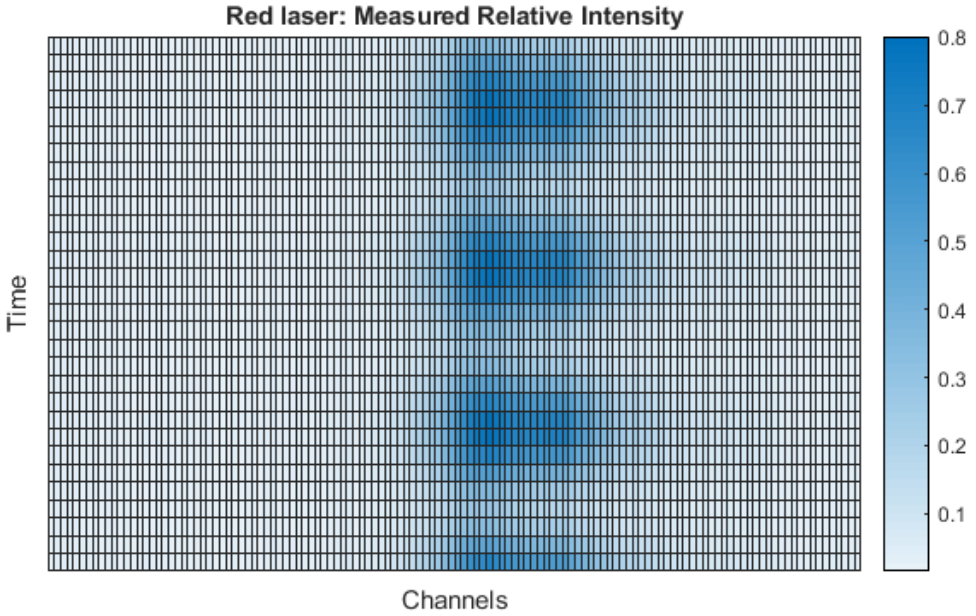


Figura 77 – Dados da aquisição continua a 100 amostras por segundo, do feixe do laser vermelho modulado a 10 Hz. O gráfico tem uma janela temporal de 30 amostras.

6. Conclusão

O sistema de aquisição de sinal desenvolvido cumpriu os objetivos estabelecidos, possibilitando a deteção de múltiplos sinais óticos no espetro infravermelho em simultâneo, e a sua observação e registo através de uma aplicação para PC. O sistema permite o mapeamento e agregação de vários pixéis do sensor de imagem num número arbitrário de canais, que pode ser utilizado para uma leitura direta da intensidade dos vários canais de um dispositivo fotónico.

A implementação do padrão SCPI foi bem sucedida, resultando num sistema de fácil configuração e utilização juntamente com a aplicação gráfica desenvolvida. Esta solução não só permitiu um acesso simplificado em tempo real às funções implementadas no microcontrolador através de uma porta USB, como torna simples a introdução de novas funcionalidades em caso de necessidade.

A aplicação gráfica auxilia na análise dos dados obtidos para uma sequência de várias aquisições de imagem singulares, ou para aquisições contínuas com uma taxa até cerca de 100 imagens por segundo. As opções de representação gráfica incluem um gráfico em linha e um gráfico em grelha temporal, em unidades de intensidade relativa, de escala logarítmica ou em milivolts. É também possível guardar e importar os dados em ficheiro com extensão .csv.

O detetor ótico mostrou uma boa versatilidade em termos de configuração para a deteção de diferentes intensidades óticas. Esta característica é proveniente das duas opções de sensibilidade do sensor de imagem, do tempo de integração variável entre 1 µs e 5 segundos, e do circuito de acondicionamento de sinal com oito opções de ganho entre 1 e 32.

A maior intensidade de radiação incidente que o sistema consegue detetar sem saturar no comprimento de onda de maior sensibilidade (~1550 nm) será inferior a 20,37 mW/cm^2. Este resultado foi limitado pela maior frequência de operação que foi obtida sem a ocorrência de artefactos na imagem, no valor de 250 kHz, impondo um tempo de integração mínimo de 4 µs. Este valor poderá ser posteriormente aumentado introduzindo uma configuração do ADC interno com recurso a DMA, sendo previsível uma possível diminuição do tempo de integração até 1 µs e consequentemente um aumento da intensidade ótica detetável sem a ocorrência de saturação.

A menor intensidade de radiação incidente detetável no comprimento de onda de 1550 nm não foi possível especificar com a configuração utilizada em laboratório, visto que a menor potência configurável para o laser levava o sistema à saturação. Outra

solução seria necessária para obter esta medida, recorrendo, por exemplo, a uma atenuação controlada do sinal ou a uma fonte emissora com uma potência menor.

O sistema desenvolvido poderá ser tornado compatível com o modelo de sensor de imagem G13913 de 256 pixéis com alterações pontuais apenas no *software*, e podendo assim aumentar a resolução do detetor caso seja necessário.

No futuro o módulo poderá ser utilizado na caracterização de amostras de circuitos fotónicos em silício amorfo, assim como na integração do sistema de diagnóstico portátil que está a ser desenvolvido no âmbito do projeto PhotoAKI.

7. Bibliografia

[1] Jalali, B. (2006). Fathpour,"*Silicon Photonics ", SJ Lightwave Technol*, *24*, 4600-4615.

[2] Fang, Z., & Zhao, C. Z. (2012). Recent progress in silicon photonics: a review. *ISRN Optics*, *2012*.

[3] Final Report Summary - SILAMPS (Silicon integrated lasers and optical amplifiers), (16 de Março de 2020), CORDIS, https://cordis.europa.eu/project/id/226470/reporting, acedido em Dezembro de 2020.

[4] Takei, R. (2016). Amorphous Silicon Photonics. *Crystalline and Non-crystalline Solids*, 21.

[5] Singh, J., & Shimakawa, K. (Eds.). (2003). *Advances in amorphous semiconductors*. CRC Press.

[6] Hamedani, Y., Macha, P., Bunning, T. J., Naik, R. R., & Vasudev, M. C. (2016). *Plasma-enhanced chemical vapor deposition: Where we are and the outlook for the future* (pp. 247-280). InTech.

[7] Moreno, M., Ambrosio, R., Torres, A., Torres, A., Rosales, P., Itzmoyotl, A., & Domínguez, M. (2016). Amorphous, Polymorphous, and Microcrystalline Silicon Thin Films Deposited by Plasma at Low Temperatures. *Crystalline and Non-crystalline Solids*, 147

[8] Zhu, S., Lo, G. Q., & Kwong, D. L. (2010). Low-loss amorphous silicon wire waveguide for integrated photonics: effect of fabrication process and the thermal stability. *Optics express*, *18*(24), 25283-25291.

[9] Selvaraja, S. K., Sleeckx, E., Schaekers, M., Bogaerts, W., Van Thourhout, D., Dumon, P., & Baets, R. (2009). Low-loss amorphous silicon-on-insulator technology for photonic integrated circuitry. *Optics Communications*, *282*(9), 1767-1770.

[10] Sun, R., McComber, K., Cheng, J., Sparacin, D. K., Beals, M., Michel, J., & Kimerling, L. C. (2009). Transparent amorphous silicon channel waveguides with silicon nitride intercladding layer. *Applied Physics Letters*, *94*(14), 141108.

[11] Evans, A. F., Hall, D. G., & Maszara, W. P. (1991). Propagation loss measurements in silicon-on-insulator optical waveguides formed by the bond-and-etchback process. Applied physics letters, 59(14), 1667-1669.

[12] Shoman, H., Jayatilleka, H., Jaeger, N. A., Shekhar, S., & Chrostowski, L. (2020). Measuring on-chip waveguide losses using a single, two-point coupled microring resonator. *Optics Express*, *28*(7), 10225-10238.

[13] Cardenas, J., Poitras, C. B., Robinson, J. T., Preston, K., Chen, L., & Lipson, M. (2009). Low loss etchless silicon photonic waveguides. *Optics express*, *17*(6), 4752-4757.

[14] Estevez, M. C., Alvarez, M., & Lechuga, L. M. (2012). Integrated optical devices for lab-on-a-chip biosensing applications. *Laser & Photonics Reviews*, *6*(4), 463-487.

[15] Soler, M., Calvo-Lozano, O., Estevez, M. C., & Lechuga, L. M. (2020). Nanophotonic Biosensors: Driving Personalized Medicine. *Optics and Photonics News*, *31*(4), 24-31.

[16] Passaro, V., Tullio, C. D., Troia, B., Notte, M. L., Giannoccaro, G., & Leonardis, F. D. (2012). Recent advances in integrated photonic sensors. *Sensors*, *12*(11), 15558-15598.

[17] González-Guerrero, A. B., Maldonado, J., Herranz, S., & Lechuga, L. M. (2016). Trends in photonic lab-on-chip interferometric biosensors for point-of-care diagnostics. *Analytical methods*, *8*(48), 8380-8394.

[18] Wang, J., Sanchez, M. M., Yin, Y., Herzer, R., Ma, L., & Schmidt, O. G. (2020). Silicon-Based Integrated Label-Free Optofluidic Biosensors: Latest Advances and Roadmap. *Advanced Materials Technologies*, *5*(6), 1901138.

[19] Molina-Fernández, Í., Leuermann, J., Ortega-Moñux, A., Wangüemert-Pérez, J. G., & Halir, R. (2019). Fundamental limit of detection of photonic biosensors with coherent phase read-out. *Optics express*, *27*(9), 12616-12629.

[20] Ho, A. H. P., Kim, D., & Somekh, M. G. (Eds.). (2017). *Handbook of photonics for biomedical engineering* (pp. 365-381). Amsterdam, The Netherlands: Springer.

[21] Debackere, P., Scheerlinck, S., Bienstman, P., & Baets, R. (2006). Surface plasmon interferometer in silicon-on-insulator: novel concept for an integrated biosensor. *Optics Express*, *14*(16), 7063-7072.

[22] Donzella, V., & Faralli, S. (2012). Recent Patents on Silicon Photonics Optical Amplifiers and Lasers. *Recent Patents on Electrical & Electronic Engineering (Formerly Recent Patents on Electrical Engineering)*, *5*(2), 120-133.

[23] Sun, R., Cheng, J., Michel, J., & Kimerling, L. (2009). Transparent amorphous silicon channel waveguides and high-Q resonators using a damascene process. *Optics letters*, *34*(15), 2378-2380.

[24] Vlasov, Y. A., & McNab, S. J. (2004). Losses in single-mode silicon-on-insulator strip waveguides and bends. *Optics express*, *12*(8), 1622-1631.

[25] Steglich, P., Hülsemann, M., Dietzel, B., & Mai, A. (2019). Optical Biosensors Based on Silicon-On-Insulator Ring Resonators: A Review. Molecules, 24(3), 519. https://doi.org/10.3390/molecules24030519

[26] Barshilia, D., Chau, L. K., & Chang, G. E. (2020). Low-cost planar waveguide-based optofluidic sensor for real-time refractive index sensing. Optics Express, 28(19), 27337. https://doi.org/10.1364/oe.400800

[27] Marchetti, R., Lacava, C., Carroll, L., Gradkowski, K., & Minzioni, P. (2019). Coupling strategies for silicon photonics integrated chips [Invited]. Photonics Research, 7(2), 201. https://doi.org/10.1364/prj.7.000201

[28] Paulsen, M., Jahns, S., & Gerken, M. (2017). Intensity-based readout of resonant-waveguide grating biosensors: Systems and nanostructures. Photonics and Nanostructures - Fundamentals and Applications, 26, 69–79. https://doi.org/10.1016/j.photonics.2017.07.003

[29] Leuermann, J., Stamenkovic, V., Ramirez-Priego, P., Sánchez-Postigo, A., Fernández-Gavela, A., Chapman, C. A., Bailey, R. C., Lechuga, L. M., Perez-Inestrosa, E., Collado, D., Halir, R., & Molina-Fernández, I. (2020). Coherent silicon photonic

interferometric biosensor with an inexpensive laser source for sensitive label-free immunoassays. Optics Letters, 45(24), 6595. https://doi.org/10.1364/ol.411635

[30] Densmore, A., Vachon, M., Xu, D. X., Janz, S., Ma, R., Li, Y. H., Lopinski, G., Delâge, A., Lapointe, J., Luebbert, C. C., Liu, Q. Y., Cheben, P., & Schmid, J. H. (2009). Silicon photonic wire biosensor array for multiplexed real-time and label-free molecular detection. Optics Letters, 34(23), 3598. https://doi.org/10.1364/ol.34.003598

[31] Mayeh, M., Viegas, J., Srinivasan, P., Marques, P., Santos, J. L., Johnson, E. G., & Farahi, F. (2009). Design and Fabrication of Slotted Multimode Interference Devices for Chemical and Biological Sensing. Journal of Sensors, 2009, 1–11. https://doi.org/10.1155/2009/470175

[32] Chatzipetrou, M., Gounaridis, L., Tsekenis, G., Dimadi, M., Vestering-Stenger, R., F. Schreuder, E., Trilling, A., Besselink, G., Scheres, L., van der Meer, A., Lindhout, E., G. Heideman, R., Leeuwis, H., Graf, S., Volden, T., Ningler, M., Kouloumentas, C., Strehle, C., Revol, V., . . . Zergioti, I. (2021). A Miniature Bio-Photonics Companion Diagnostics Platform for Reliable Cancer Treatment Monitoring in Blood Fluids. Sensors, 21(6), 2230. https://doi.org/10.3390/s21062230

[33] Hamamatsu, “InGaAs linear image sensors G13913 series: Near infrared image sensors for portable analytical instruments”, KMIR1033E03, October 2019.

[34] STMicroelectronics, “Reference Manual: STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs”, RM0008 Rev 20, December 2018.

[35] STMicroelectronics, “STM32F103x8 STM32F103xB Datasheet – production data”, DocID13587 Rev 17, August 2015.

[36] STMicroelectronics, “User Manual: Description of STM32F1 HAL and low-layer drivers”, UM1850 Rev 3, February 2020.

[37] Microchip, “MCP6S91/2/3: Single-Ended, Rail-to-Rail I/O, Low-Gain PGA”, DS2190A, 2004.

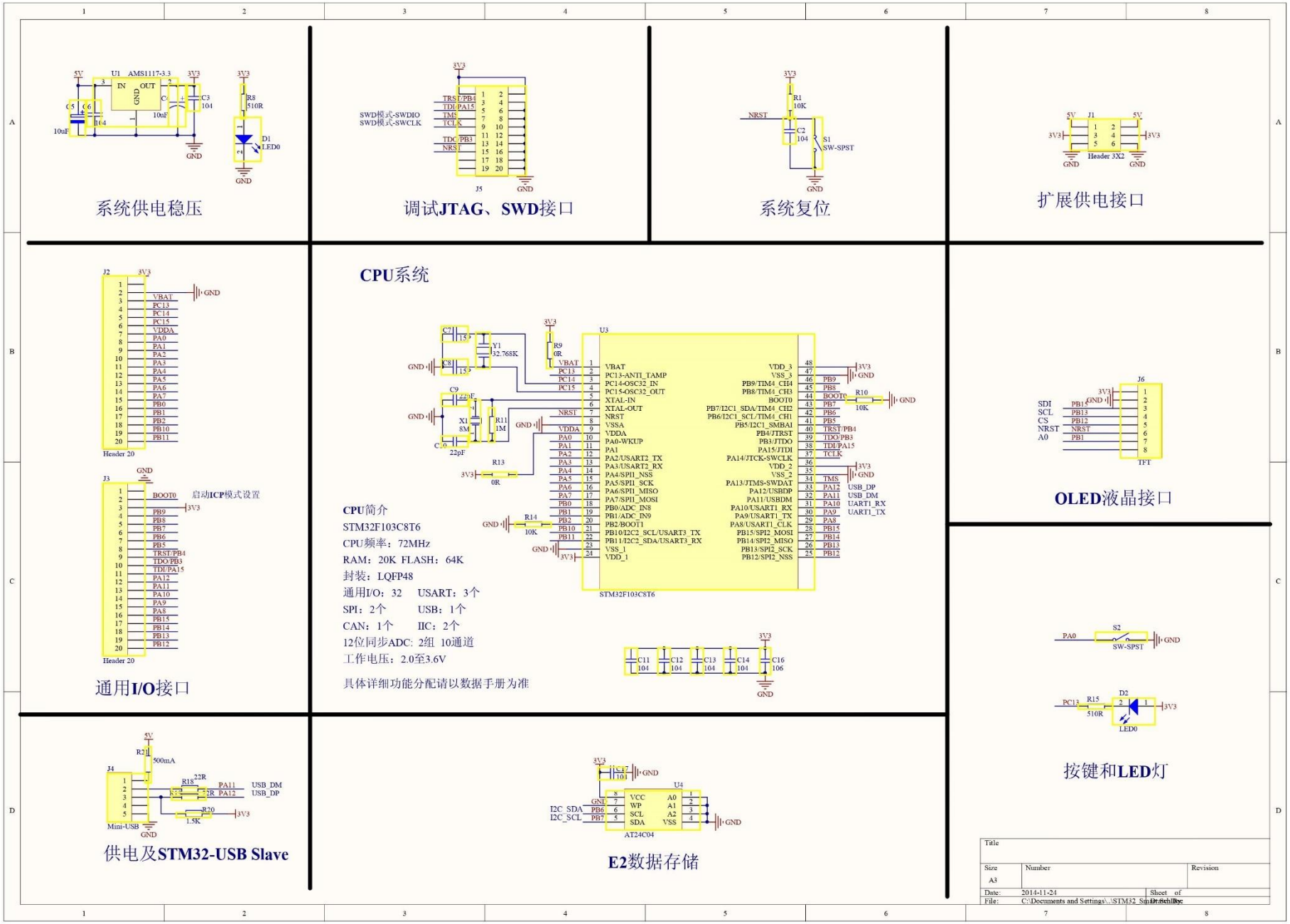
[38] Microchip, “MCP4725: 12-Bit Digital-to-Analog Converter with EEPROM Memory in SOT-23-6”, DS22039D, 2009.

[39] Microchip, “MCP33131D/21D/11D-10: 1 Msps 16/14/12-Bit Differential Input SAR ADC”, DS20005947A, 2018.

[40] Microchip, “MCP1700: Low Quiescent Current LDO”, DS20001826E, 2005 [Revised 2018].

[41] Analog Devices, “AD7801: +2.7 V to 5.5 V, Parallel Input, Voltage Output 8-Bit DAC”, AD7801 Datasheet rev .0, 1997.

Anexo A.1 – Esquemático da placa de desenvolvimento do MCU STM32F103C8



Anexo A.2 – Arquitetura do MCU da familia STM32F1

A Figura 79 ilustra o diagrama de blocos da arquitetura de microcontroladores da família STM32F1[4]. Além dos componentes mencionados, estão esquematizados os barramentos que fazem as suas interligações, nomeadamente os barramentos de instruções, de depuração, do sistema e do DMA, o barramento *Advanced High-performance Bus* (AHB) e as pontes para dois barramentos *Advanced Peripherial Bus* (APB) que ligam os periféricos. A matriz de barramento (*Bus matrix*) gere o acesso do CPU e do DMA ao AHB e às memórias. Às memórias estão também associados os seus dispositivos de interface, a interface de flash (FLITF) e o *Flexible Static Memory Controler* (FSMC). O módulo *Reset & Clock Control* (RCC) é responsável pelo controlo dos sinais de relógio e de reinício do sistema e dos periféricos. Vários dispositivos requerem a configuração e ativação do seu sinal de relógio, sendo útil a compreensão da sua posição nos barramentos e das frequências dos sinais de relógio que os alimentam.

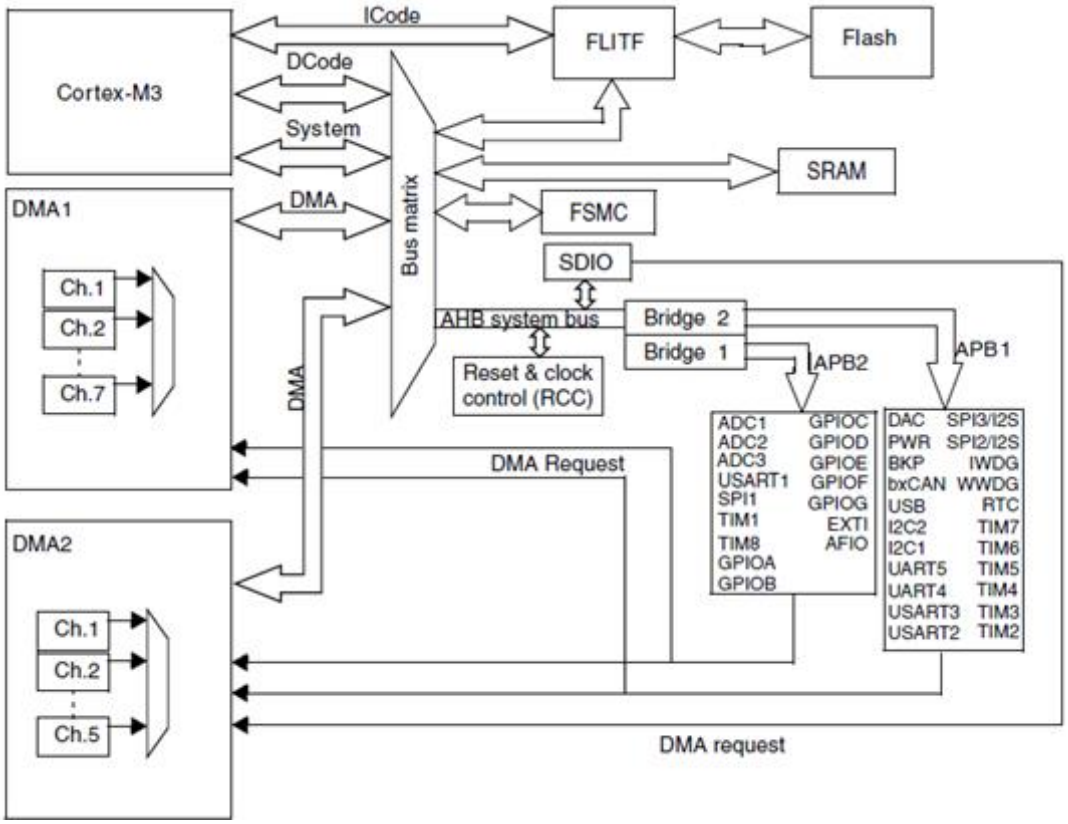


Figura 79 – Diagrama de blocos da arquitetura do sistema dos dispositivos da família STM32F1, retirado de [34].

[4] O núcleo STM32F103C8, de densidade média, não contém todos os periféricos representados.

Anexo B – Desenvolvimento do *firmware*

1. Controlador do sensor de imagem

Sinal de relógio

O sinal de relógio, tendo sido especificado entre 100 kHz e 2 MHz para o dispositivo G13913, é gerado utilizando um temporizador de *hardware*. Esta solução liberta o processador da tarefa de contabilizar o tempo e comutar o valor lógico de um pino de GPIO, ficando apenas encarregue da inicialização e configuração do dito. De outra forma, dadas as frequências pretendidas, não seria viável.

O controlador do periférico de temporização foi feito com a afetação direta dos registos, pois mostrou-se mais simples do que a utilização dos controladores fornecidos pelo fabricante. As funções escritas permitem inicializar o temporizador, ligar e desligar o sinal de relógio e alterar a sua frequência. No Anexo B.1 encontra-se o código do cabeçalho do módulo 'MyDrivers', contendo as definições dos pinos utilizados e os protótipos das funções dos controladores. No Anexo B.2 encontra-se o código referente ao controlador do sinal de relógio.

A função de inicialização começa por ativar o sinal de relógio do periférico. De seguida é feita a configuração do pino GPIO PB0, através da função de inicialização de GPIO da biblioteca HAL. O pino tem de ser configurado no modo de função alternativa, correspondendo ao canal 3 do temporizador TIM3, e com saída em *push-pull*. Os registos do temporizador são de seguida configurados.

```
/* Timer config */
TIM3->CR1 = 0;              // Configure register 1
TIM3->CR2 = 0;              // Configure register 2
TIM3->ARR = 1;              // Auto-Reload register
TIM3->CNT = 0;              // Counter
TIM3->PSC = (uint16_t)((18e6/freq)-1); // Pre-scaler register
TIM3->CCR3 = 0;             // Capture/Compare register
TIM3->CCMR2 = 0b011 << 4;   // Output compare mode ch 3
TIM3->CCER = 1 << 8;        // CC 3 output enable
```

Os registos de configuração TIM3_CR1 e TIM3_CR2 têm as configurações básicas do temporizador quando colocados a zero: contagem num único sentido e no sentido ascendente. O valor lógico '1' no primeiro bit de TIM3_CR1 ativa o funcionamento do temporizador.

O registo de auto-recarregamento, TIM3_ARR, reinicia o contador a zero quando este, TIM3_CNT, atinge o valor do primeiro. Com o valor 1 em TIM3_ARR, o contador é colocado a zero a cada dois ciclos de relógio.

O temporizador recebe um relógio à frequência de 72 MHz[5]. Este é depois dividido por um pré-divisor programável (registo TIM3_PSC) por qualquer factor entre 1 e 65536, definindo assim a frequência da contagem. O contador pode então ser configurado para funcionar entre as frequências de 1099 Hz e 72 MHz, reiniciando a cada dois ciclos entre 549 Hz ou 36 MHz, respectivamente.

O registo TIM3_CMRR2 define o modo de funcionamento para os canais 3 e 4, tendo neste caso sido introduzidos os bits para o modo de comutação da saída (*Output-Compare Mode*) no canal 3. Desta forma o pino é comutado sempre que o contador atinge o valor de comparação, guardado no registo TIM3_CCR3. Colocando este a zero, a saída é comutada cada vez que o contador reinicia. A cada duas comutações é feito um ciclo completo, pelo que a frequência do sinal de relógio do sensor de imagem pode finalmente ser programada entre cerca de 275 Hz e 18 MHz.

As frequências programáveis através do controlador desenvolvido estão dependentes das divisões inteiras aplicadas através do pré-divisor de relógio. Se a frequência obtida antes do pré-divisor fosse 2 MHz (o valor máximo especificado pelo sensor de imagem), os valores disponíveis na escala das centenas de kHz seriam escassos:

$$f_{op} \in \frac{2\ \mathrm{MHz}}{x} : \mathrm{x} \in \mathbb{N}^{+} = \{\ 2\ \mathrm{MHz}, 1\ \mathrm{MHz}, 666\ \mathrm{kHz}, 500\ \mathrm{kHz}, \dots\}$$

Por essa razão a configuração do temporizador foi pensada para obter uma frequência de 18 MHz antes do pré-divisor, ficando a frequência final limitada a 2 MHz apenas na função de configuração do temporizador. Assim haverá uma maior diversidade de valores disponíveis:

$$f_{op} \in \frac{18\ \mathrm{MHz}}{x} : x \in \mathbb{N}^{+} \wedge x > 8 = \{\dots, 1\ \mathrm{MHz}, 947\ \mathrm{kHz}, 900\ \mathrm{kHz}, 857\ \mathrm{kHz}, 818\ \mathrm{kHz}, \dots\}$$

Esta configuração também possibilita a programação de frequências mais baixas, mesmo inferiores ao limite mínimo de 100 kHz especificado para o sensor, o que teve utilidade em testes durante o desenvolvimento do sistema.

As frequências configuráveis através das funções `Clock_Setup()` e `Clock_Freq()` foram então limitadas entre 300 Hz e 2 MHz. A Figura 81 e a Figura 80 mostram os sinais gerados às frequências de 1 kHz e 1 MHz respectivamente.

[5] O relógio do sistema está configurado a 72 MHz, sendo dividido por 2 para alimentar o relógio do barramento APB1, a 36 MHz. Segundo a secção 8.3.2. Clock Configuration Register do manual de referência da família STM32F1, quando o relógio do ABP1 é dividido, os temporizadores neste barramento recebem um sinal de relógio com o dobro da frequência, pelo que o temporizador recebe o sinal a 72 MHz.

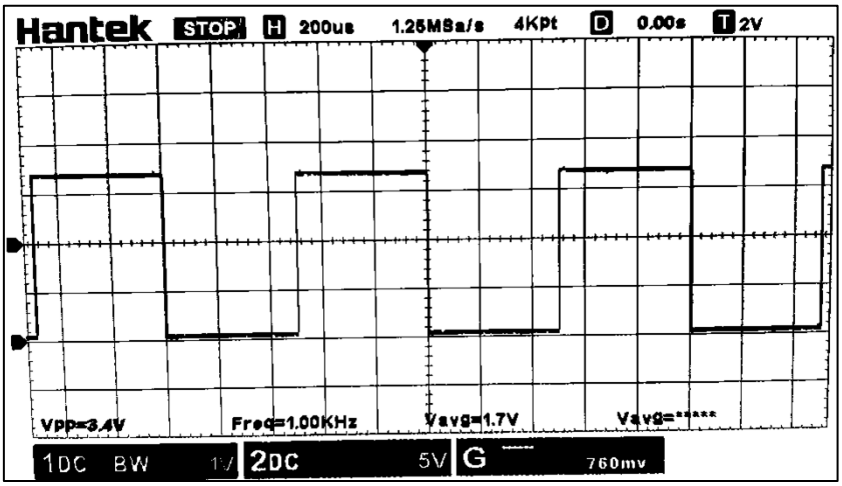


Figura 81 - Observação no osciloscópio do sinal de relógio gerado a 1 kHz.

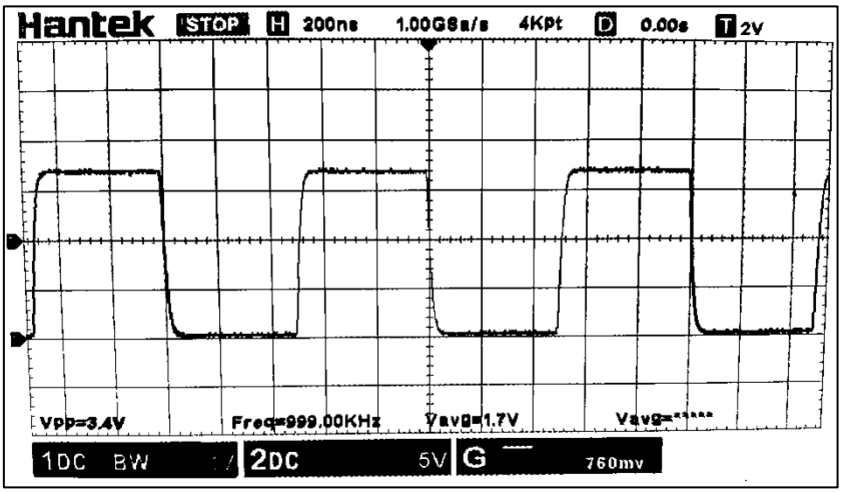


Figura 80 - Observação no osciloscópio do sinal de relógio gerado a 1 MHz.

Sinal de reset

O controlador para o sinal de *reset* foi escrito de forma semelhante, mas utilizando o canal 4 do temporizador TIM2. A utilização de *hardware* dedicado para gerar o sinal de pulso permite um controlo preciso da sua duração, mesmo em pulsos na ordem dos microssegundos. Tal precisão não seria possível se o controlo estivesse a cargo do processador. O código completo encontra-se no Anexo B.3. É disponibilizada uma função de inicialização, e outra função para a geração de pulsos entre 1 µs e 5 s, em passos de 1 µs ou 0,1 ms.

A função de inicialização começa, tal como a anterior, com a ativação do relógio do temporizador TIM2, seguindo-se a configuração do pino PA3 para a função alternativa com saída em *push-pull*. Os registos do temporizador são depois afetados como se exemplifica de seguida:

```
TIM2->CR1 = 1<<3; // Bit para modo OPM (one-pulse mode)
TIM2->CR2 = 0;
TIM2->CNT = 0;
TIM2->PSC = 71;    // base temporal de 1 microssegundo
TIM2->ARR = x;     // t_pulse de x microssegundos
TIM2->CCR4 = 1;    // t_delay de 1 microssegundo
TIM2->CCMR2 = 0b111 << 12 | 1<<10; // Bits para modo PWM2
TIM2->CCER = 1 << 12; // OUTPUT ENABLE CH4
```

O tempo de integração passado como argumento é avaliado entre duas gamas de valores. Se for inferior a 1 milissegundo, o pré-divisor, TIM2_PSC, é afetado com o valor 71 para gerar uma base temporal de 1 microssegundo (dividindo a frequência de 72 MHz por 72). Caso contrário, é afetado com o valor 7199 para gerar uma base temporal de 0,1 milissegundos (dividindo 72 MHz por 7200). Dado que o pré-divisor tem uma dimensão de 16 bits, sendo a maior divisão possível por 63556, para obter uma base temporal de 1 milissegundo seria necessário reconfigurar o relógio que alimenta o barramento APB1, o que teria consequências para os outros periféricos neste barramento. Optou-se por isso pelo valor de 0,1 ms para a base temporal maior.

No registo TIM2_CR1, é colocado a '1' o bit correspondente ao modo de pulso singular (ou *One-Pulse Mode*, OPM). Neste modo é possível gerar um sinal com uma determinada duração, após um atraso também configurável. A Figura 82 mostra um diagrama temporal dos sinais envolvidos.

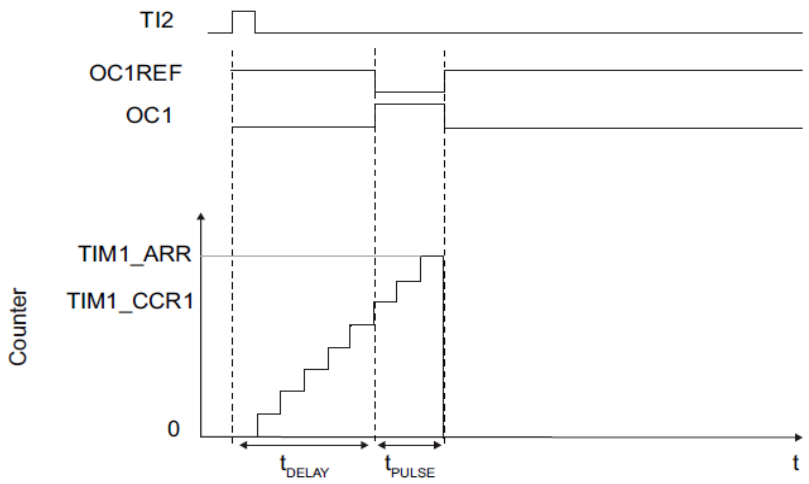


Figura 82 – Diagrama temporal que mostra o resultado da relação entre o registo de comparação TIMx_CRRx e o registo de auto-recarregamento TIMx_ARR, após um pulso do sinal de entrada. Retirado de [34].

O registo de comparação TIM2_CRR4 tem de ter um valor diferente do valor inicial do contador, obrigando a um atraso igual à base temporal utilizada. O registo TIM2_CMR2 é configurado para o modo PWM2 no canal 4[6]. Para produzir um pulso, o registo

[6] No modo PWM 2 a saída do comparador (OC ou Output-Compare) fica no valor lógico '1' quando o contador (TIM2_CNT) ultrapassa o valor do registo de comparação (TIM2_CRR4).

TIM2_ARR é carregado com a duração pretendida em microssegundos ou décimas de milissegundos, conforme a base temporal utilizada, sendo colocado o contador em funcionamento através do primeiro bit de TIM2_CR1.

A Figura 83 e a Figura 84 mostram os sinais gerados com duas durações distintas.

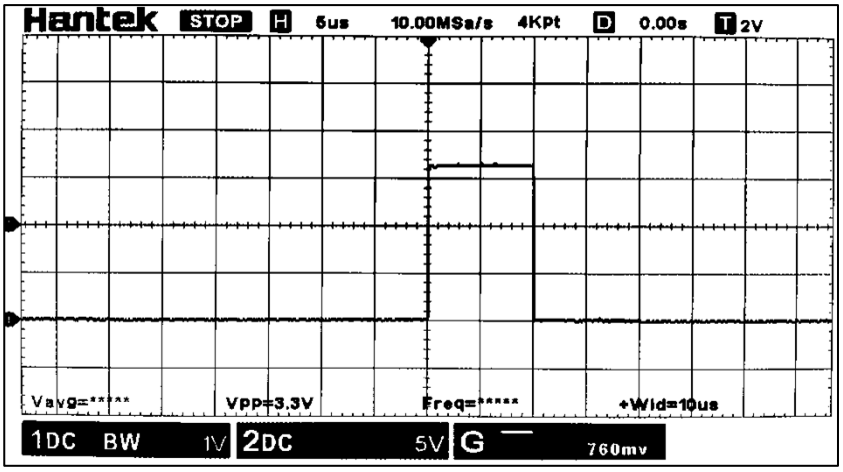


Figura 83 - Observação no osciloscópio do sinal de pulso de reset de 10 µs.

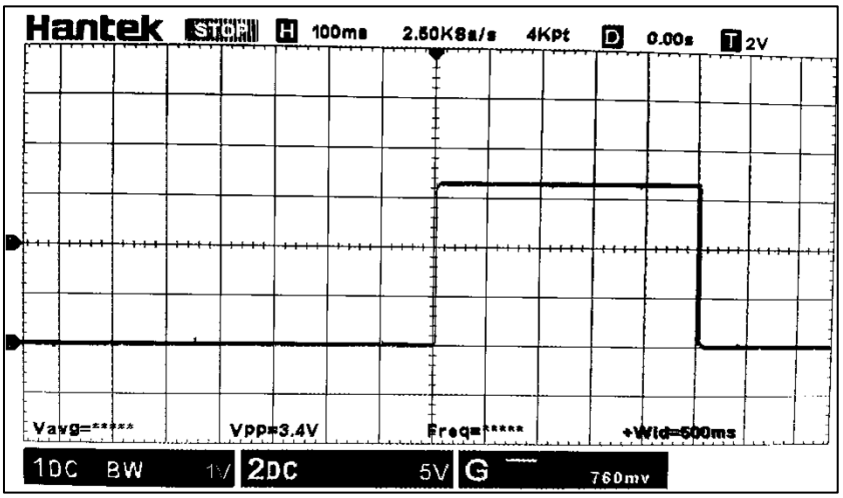


Figura 84 - Observação no osciloscópio do sinal de pulso de reset de 500 ms.

Sinal de seleção de ganho (cf_sel)

O sinal *cf_sel*, sendo apenas comutado no momento da sua configuração, é controlado com um pino de GPIO. As funções escritas para o seu controlador utilizaram simplesmente a biblioteca do Arduino para configurar e mudar o valor lógico na saída, estando o código disponível no Anexo B.4.

Com os controladores dos três sinais descritos é possível gerir os sinais de controlo do sensor de imagem. Para isso basta configurar os sinais de relógio e o sinal de seleção do ganho do transdutor, e aplicar um pulso do sinal de *reset* para efetuar uma aquisição

Assim, enquanto a contagem se encontra entre os valores de TIM2_CRR4 e TIM2_ARR um sinal de pulso é produzido.

de imagem. Quando o processo de integração termina, segue-se a transmissão por parte do sensor do sinal de vídeo e dos sinais auxiliares para a conversão.

2. Controlador do circuito de acondicionamento

Na entrada do circuito é esperado um sinal de vídeo com uma tensão máxima típica de 2,5 V, que corresponde à tensão no escuro, e uma tensão mínima que pode ir até ~0,3 V, em saturação. O propósito do circuito é o aumento da excursão do sinal de vídeo, para ocupar a gama de valores entre os 0 V e os 3,3 V e assim obter a melhor resolução na conversão para o domínio digital.

Cálculo dos parâmetros

O circuito integrado do PGA é o MCP6S91, cujo diagrama de blocos se encontra na Figura 85. No diagrama verifica-se que o integrado contem um amplificador operacional em montagem não-inversora, com uma escada de resistências na malha de realimentação. O terminal negativo do amplificador operacional é ligado a um ponto da escada através de uma porta de transmissão, estabelecendo o ganho configurado para o circuito.

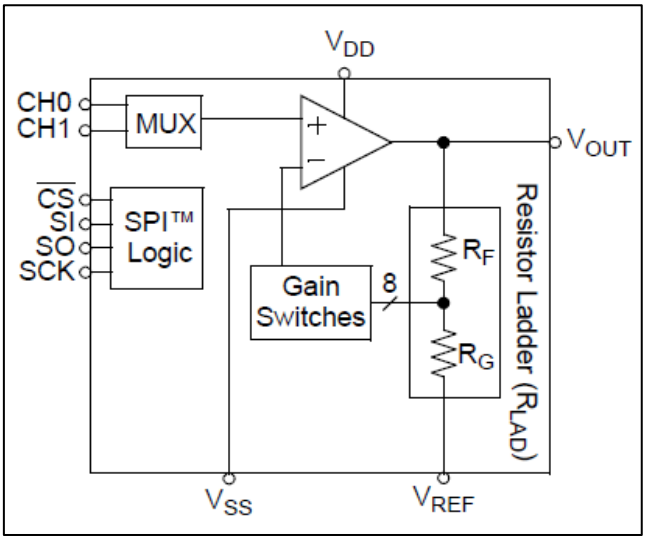


Figura 85 – Diagrama de blocos do PGA MCP6S91/2/3, retirado de [37].

Visto a partir do terminal *Vref*, o circuito funciona como uma montagem inversora. Recorrendo ao Teorema da Sobreposição, obteve-se a expressão para a tensão na saída a partir da tensão na entrada e da tensão *Vref*, que será:

$$V_o = V_i G - V_{ref}(G - 1)$$

O DAC utilizado para a programação do valor de referência foi o integrado MCP4725, com 12 bits de resolução. Este permite gerar tensões num intervalo de 0 a cerca de 3,3 V, com passos de aproximadamente 0,8 mV. A comunicação é feita através de um barramento I2C.

Foram obtidas as configurações de ganho e tensão *Vref* que otimizam a excursão do sinal para a conversão. Sabendo que a excursão máxima na saída será 3,3 V, a excursão máxima na entrada será 3,3/*G*, podendo G corresponder aos seguintes valores: 1, 2, 4, 5, 8, 10, 16 e 32. O valor de tensão máximo na entrada será o da tensão de referência do sensor de imagem, tipicamente 2,5 V. Os valores de tensão mínimos corresponderão à subtração da excursão na entrada a este valor. Na Tabela 10 encontram-se as configurações ideais de ganho para a gama de valores do sinal de vídeo.

Tabela 10 – Cálculo dos limites do sinal de vídeo para cada seleção de ganho do PGA.

Ganho	1	2	4	5	8	10	16	32
Excursão máxima na entrada [V]	3,3	1,65	0,825	0,66	0,4125	0,33	0,2062	0,1031
Vi_{MAX} [V]	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5
Vi_{Min} [V]	0,3	0,85	1,675	1,84	2,0875	2,17	2,2938	2,3969

Por fim, podem ser consideradas as seguintes expressões para o cálculo do valor de referência ideal para ganhos superiores ao unitário:

$$3{,}3 = V_{i_{MAX}} G - V_{ref}(G-1) \quad \text{v} \quad 0 = V_{i_{MIN}} G - V_{ref}(G-1)\,, \qquad \text{G} > 1$$

Disto resulta:

$$V_{ref} = \frac{V_{i_{MAX}} G - 3{,}3}{(G-1)} \quad \text{v} \quad V_{ref} = \frac{V_{i_{MIN}} G}{(G-1)}, \qquad \text{G} > 1$$

Calculando as tensões de referência para cada par de G e Vi máximo ou mínimo, obteve-se os resultados da Tabela 11. Estes resultam nas funções de transferência apresentadas na Figura 86.

Tabela 11 – Configurações ótimas de ganho e tensão de referência para a excursão do sinal de video.

Limites de Vi [V]	2,5 a 0,3	2,5 a 0,85	2,5 a 1,675	2,5 a 1,84	2,5 a 2,0875	2,5 a 2,17	2,5 a 2,2938	2,5 a 2,3969
Ganho	1	2	4	5	8	10	16	32
Vref [V]	-	1,7	2,233	2,3	2,386	2,411	2,447	2,474

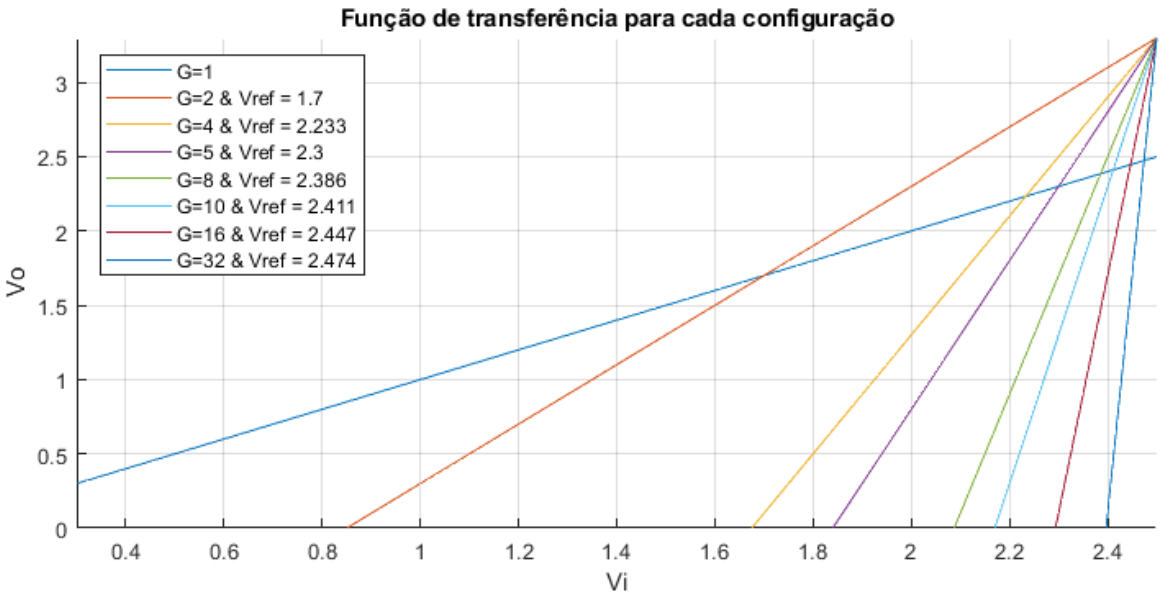


Figura 86 – Gráfico da função de transferência do circuito de acondicionamento para cada configuração de ganho e valor de referência.

Controlador do PGA

As funções para controlo do PGA foram escritas com base nas bibliotecas do Arduino para configuração dos pinos, assim como para transmitir os comandos ao circuito integrado através do barramento SPI. A leitura da ficha técnica do dispositivo indicou que a sua programação era possível enviando apenas dois bytes, sendo o primeiro o código da instrução do ganho e o segundo o código de seleção do valor, entre 0 e 7, correspondendo a cada um dos oito valores de ganho do dispositivo por ordem crescente. O código encontra-se no Anexo B.4.

Controlador de I2C por software

O controlo do DAC de 12 bits é feito através do barramento I2C. No momento em que o seu controlador foi escrito, tanto o código da biblioteca Arduino para o barramento I2C (‘Wire.h’) como o da biblioteca do controlador HAL revelaram-se pesados, fazendo o programa ultrapassar a dimensão da memória Flash. Dada a simplicidade da comunicação do único dispositivo existente no barramento, optou-se pelo desenvolvimento de um controlador de I2C por *software* apenas com as funções necessárias para o seu funcionamento.

Uma comunicação I2C tem um sinal de relógio e um sinal de dados, utilizando resistências de *pull-up* e controladores com transístores em dreno aberto para gerar os dois níveis lógicos. Sem atividade os sinais encontram-se no nível ativo, devendo o mestre, ou o escravo, puxá-los ao nível inativo nos tempos especificados pelo protocolo. A definição dos valores de tensão do DAC pode ser feita exclusivamente com

transmissões de alguns *bytes* a partir do microcontrolador. A Figura 87 exemplifica os sinais do barramento nesta situação.

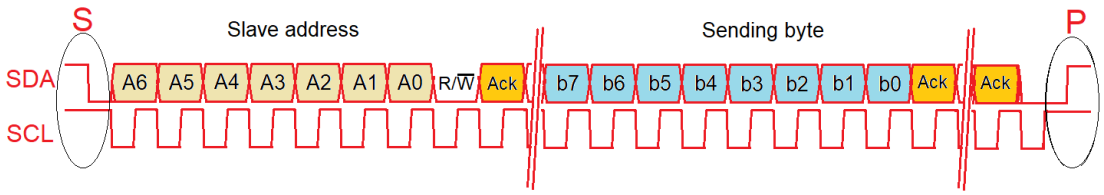


Figura 87 – Diagrama temporal de uma transmissão do mestre para o escravo no barramento I2C.

A comunicação começa com um sinal de início 'S', que consiste numa transição descendente do sinal de dados enquanto o sinal de relógio se mantém ativo. Segue-se a transmissão do endereço I2C do DAC e de um *bit* a zero indicando que se seguirá uma escrita no dispositivo. As transições do sinal de dados acontecem apenas quando o sinal de relógio está inativo, sendo o seu valor lido durante o período ativo. Caso o endereço esteja correto e o DAC disponível, este responderá com um sinal de *acknowledge,* que consiste em puxar o sinal de dados a zero, e a comunicação irá prosseguir. Se esta resposta alguma vez for negativa, ou seja, o sinal de dados manter-se ativo no período correspondente, a transmissão deve ser terminada com um sinal de paragem libertando o barramento. Continuando, um número indeterminado de *bytes* é transmitido, cada um seguido de uma resposta de *acknowledge*. Depois do último *byte*, o microcontrolador gera um sinal de paragem, 'P', que consiste numa transição ascendente do sinal de dados enquanto o sinal de relógio está ativo.

Para a implementação do controlador I2C em *software*, escreveram-se um conjunto de *macros* para fazer as transições dos pinos PB8 e PB9, e fazer a temporização em passos de ¼ de ciclo de relógio:

```
#define SET_SCL_HIGH      (GPIOB->ODR |= 1<<8)
#define SET_SCL_LOW       (GPIOB->ODR &= ~(1<<8))
#define SET_SDA_HIGH      (GPIOB->ODR |= 1<<9)
#define SET_SDA_LOW       (GPIOB->ODR &= ~(1<<9))
#define DELAY_QUARTER_I2C     (delayMicroseconds(2))
```

Com as macros fizeram-se funções para configurar os pinos, gerar os sinais 'S' e 'P', enviar um *byte* e ler uma resposta de *acknowledge.* Com estas funções escreveram-se uma função de transmissão, e uma função para verificar se o dispositivo responde. Os pinos são configurados como saída em dreno aberto[7]. O código é suficiente para

[7] Durante o desenvolvimento deste módulo, foi acrescentada uma solução para configurar o barramento I2C sem resistências externas, utilizando os pinos em configuração de saída *push-pull* e mudando a configuração para entrada com resistência de *pull-up* no momento anterior à leitura do sinal de *acknowledge*. Esta opção é determinada pela introdução da diretiva de pré-processador: `#define SOFTI2C_OD_FOR_ACK_ONLY`. Na última versão da PCB

estabelecer as comunicações I2C necessárias para esta aplicação, mas tem uma dimensão muito inferior à das bibliotecas. Este encontra-se no Anexo B.5.

Controlador do DAC para tensão Vref

Após a escrita do controlador I2C, o controlador para o DAC pode-se resumir à utilização da função de transmissão para definir o valor de tensão na sua saída. O código do encontra-se no Anexo B.6. Segundo a ficha técnica do dispositivo, é possível alterar o valor do DAC de forma volátil, ou de forma permanente recorrendo à sua memória EEPROM, como ilustrado na Figura 88. Optou-se por utilizar o comando de escrita volátil[8]. Este é constituído por 4 bytes, incluindo o endereço I2C no primeiro, o código da instrução e de controlo da resistência de saída do DAC no segundo, e por fim os 12 bits do valor e 4 bits de enchimento.

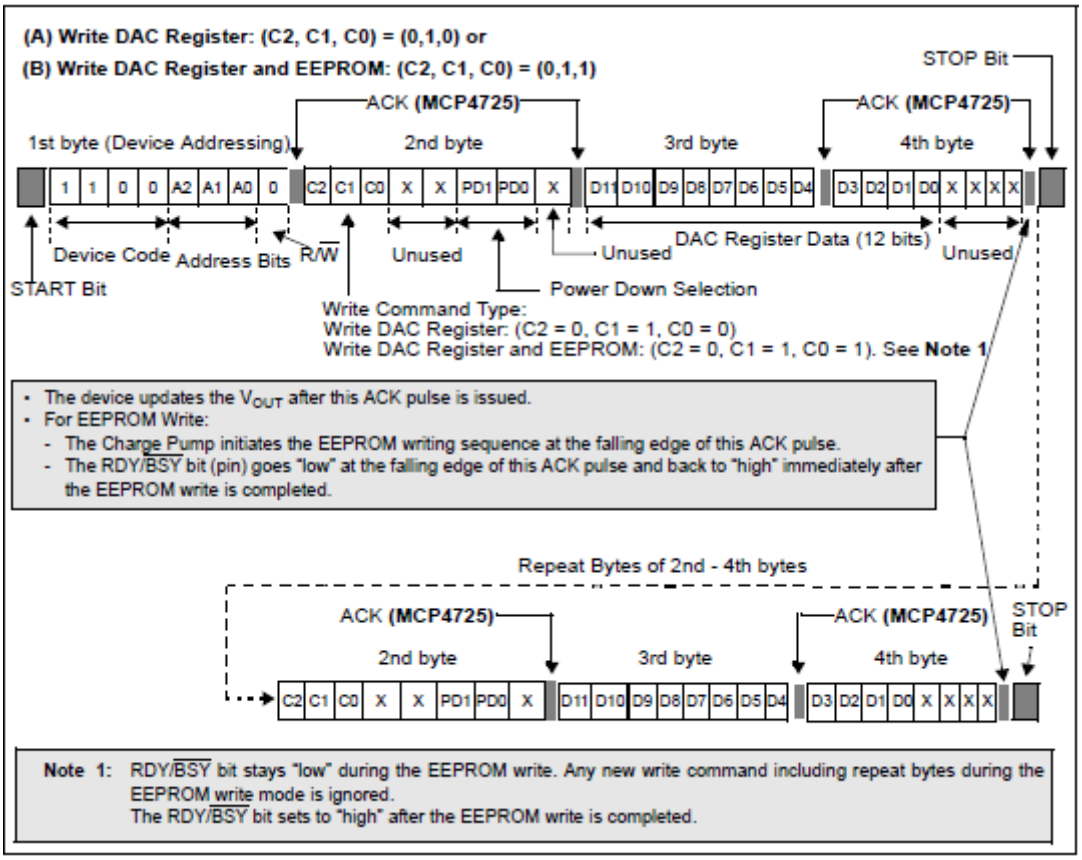


Figura 88 – Diagrama dos comandos de escrita do dispositivo MCP4725, retirado de [38].

A função de escrita limita inicialmente o valor de tensão a 3299 mV pois o DAC, com um passo de 0,8 mV, poderá atingir um valor máximo teórico de 3299,2 mV. É efetuado o cálculo do valor do DAC correspondente à tensão introduzida, no formato de virgula

produzida, foram colocadas resistências de *pull-up* no barramento I2C, alterando-se o programa final para utilizar a configuração dos pinos em dreno aberto.

[8] Por predefinição, o DAC será configurado juntamente com o PGA, pelo que não se considerou qualquer vantagem na utilização da EEPROM. Uma solução para o armazenamento da configuração completa do sistema será introduzida através da aplicação gráfica.

flutuante. A conversão para o tipo inteiro resulta sistematicamente num arredondamento por defeito, pelo que antes é somada meia unidade ao valor, resultando num arredondamento correto. A expressão seguinte é equivalente ao cálculo efetuado.

$$Valor\ DAC = Floor\left(\frac{V_{mV}}{3300} \times 4096 + 0{,}5\right), \qquad V_{mV} \in \mathbb{R}^+ : V_{mV} \leq 3299$$

Teste do circuito de acondicionamento

Com um osciloscópio no modo XY, observaram-se as funções de transferência do circuito de acondicionamento no intervalo de tensões entre 0 V e 3,3 V. O PGA e o DAC foram programados com os parâmetros calculados. A Figura 89 mostra as medições para os ganhos de 1 a 5, e a Figura 90 as medições para os ganhos de 8 a 32.

As funções de transferência mostraram o comportamento pretendido, servindo para amplificar sinais com várias excursões possíveis e um máximo de 2,5 V.

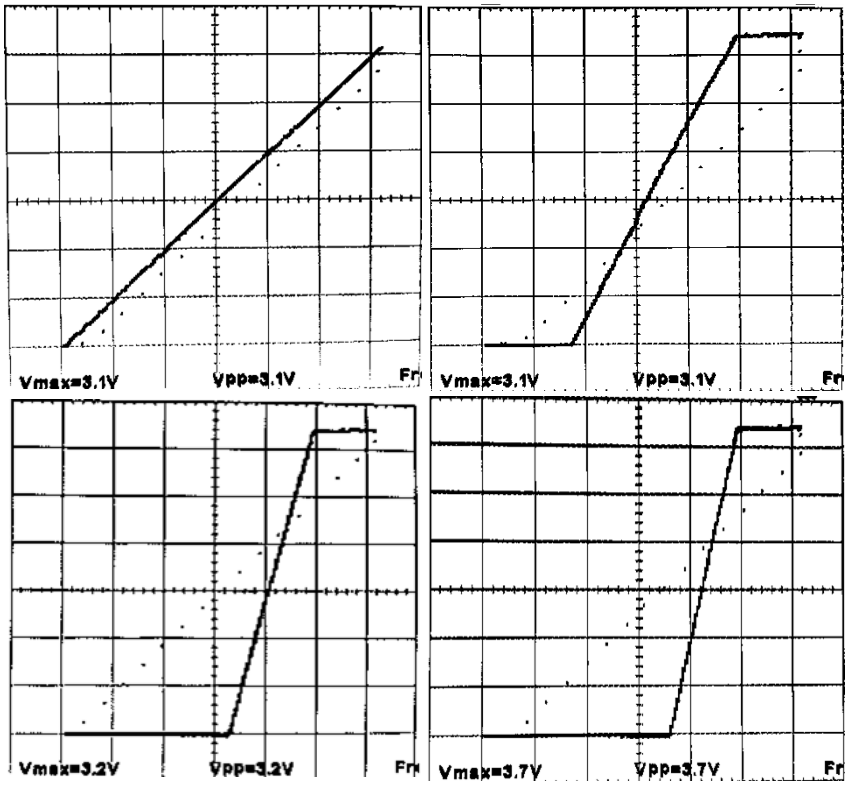


Figura 89 – Imagens do osciloscópio em modo XY, grelha com resolução de 500 mV. Medição da função de transferência do circuito de acondicionamento com os parâmetros calculados, com os ganhos: 1, 2, 4 e 5, da esquerda para a direita e de cima para baixo.

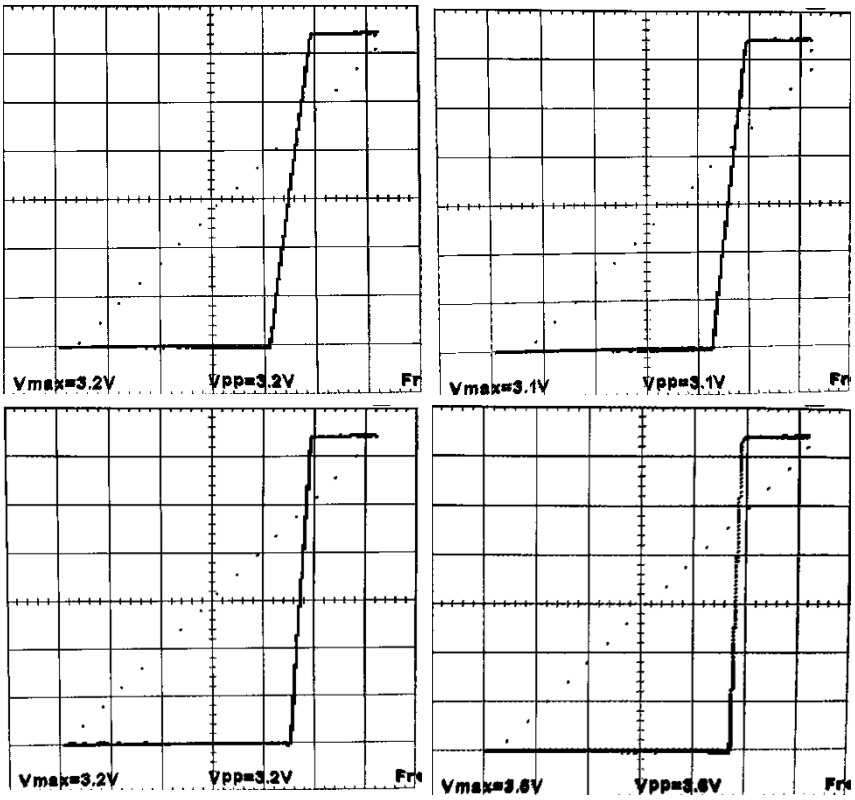


Figura 90 – Imagens do osciloscópio em modo XY, grelha com resolução de 500 mV. Medição da função de transferência do circuito de acondicionamento com os parâmetros calculados, com os ganhos: 8, 10, 16 e 32, da esquerda para a direita e de cima para baixo.

3. Controladores dos conversores analógico-digitais

Apesar da existência de um ADC no microcontrolador, optou-se pela inclusão de um conversor externo com uma resolução mais alta. Mantiveram-se os dois, podendo o utilizador escolher o conversor que deseja utilizar. O integrado escolhido para o ADC externo foi o MCP33131, tratando-se de um conversor de aproximações sucessivas de 16 bits capaz de frequências de amostragem até 1 Msps.

O controlador dos conversores apresenta um conjunto de funções que permitem a inicialização dos dois ADCs, dar o início a uma sequência de 128 conversões, esperar pelo fim da sequência, obter os resultados, e ativar ou desativar o conversor externo. Os protótipos das funções são as seguintes, estando o código completo no Anexo B.7.

```
void ADC_Setup();
void ADC_Prepare();
int  ADC_WaitSequence(uint32_t timeout);
void ADC_GetData(uint16_t newData[128]);
void ExternalADC_Setup();
void ExternalADC_Calibrate();
void ExternalADC_Enable();
void ExternalADC_Disable();
int16_t ADC_SingleConvertion();
```

A função `ADC_Setup()` inicializa o ADC interno, devendo a função `ExternalADC_Setup()` ser chamada para inicializar o ADC externo, e as funções `ExternalADC_Enable()` e `ExternalADC_Disable()` para ativar o ADC pretendido. A função `ADC_Prepare()` coloca o sistema pronto para receber o sinal 'AD_sp' do sensor de imagem e iniciar a sequência de conversões. A função `ADC_WaitSequence()` faz o programa aguardar pela conclusão da sequência de conversões até um tempo determinado e por fim a função `ADC_GetData()` obtém os dados. O resultado das conversões é dado num tipo inteiro com valores entre 0 e 32767, havendo bits de enchimento à direita no caso do ADC de menor resolução. Por fim, a função `ADC_SingleConvertion()` permite testar o funcionamento do conversor ativo, iniciando uma conversão e bloqueando até retornar o valor.

Controlador do ADC interno

O controlador do conversor analógico-digital do microcontrolador foi desenvolvido tendo por base o respectivo controlador HAL do fabricante. Este por sua vez disponibiliza funções de utilização do ADC em vários modos: bloqueante, com interrupções ou com DMA. O modo bloqueante coloca o processador a aguardar a conclusão de uma conversão. O modo com interrupções permite iniciar uma conversão ou sequência de conversões, gerando uma interrupção cada vez que o valor do ADC estiver pronto para ser lido. O modo com DMA permite realizar uma sequência de conversões, copiando o resultado de cada conversão para uma região de memória, sem intervenção do processador e gerando uma interrupção no fim da sequência.

A solução desenvolvida recorreu ao modo com interrupções por uma questão de simplicidade e de tempo. Um teste à frequência de amostragem obtida mostrou a capacidade de obter várias centenas de milhares de amostras por segundo, suficiente para a aplicação pretendida. Dispensou-se por isso o tempo de desenvolvimento necessário para introduzir o DMA.

O diagrama de blocos apresentado na Figura 91 ilustra os componentes e rotinas de interrupção envolvidas no processo de aquisição da sequência do sinal de vídeo. Note-se os três sinais: 'AD_sp', 'AD_trig', e 'EOC'. O sinal 'AD_sp' provoca uma interrupção para iniciar o processo de aquisição da sequência de vídeo, o sinal 'AD_trig' inicia uma conversão do ADC e o sinal 'EOC' provoca uma interrupção para ler uma amostra e verificar a condição de fim do processo.

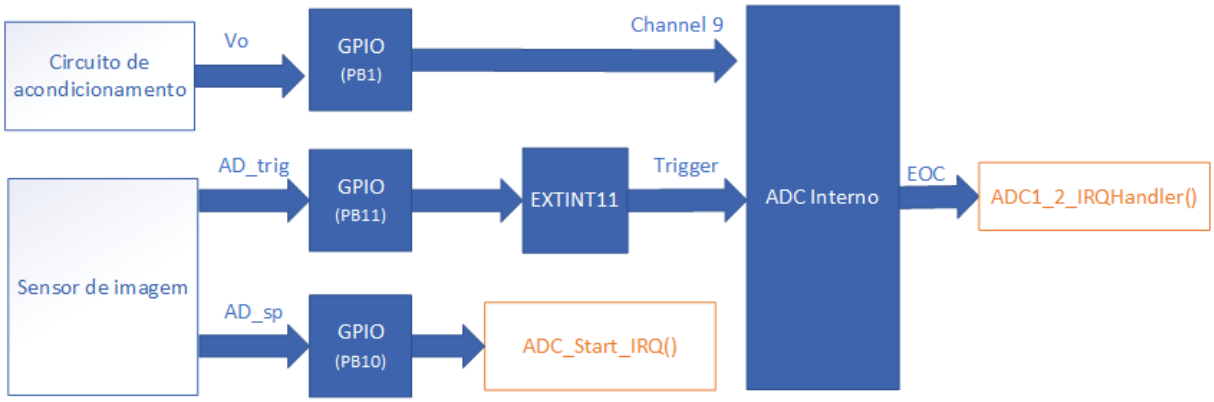


Figura 91 – Diagrama de blocos do ADC interno, mostrando ligações com o microcontrolador e restantes blocos.

A função de inicialização começa por configurar o relógio do ADC. São depois preenchidas as estruturas de inicialização para o ADC e para o canal pretendido. É selecionada a opção de início de conversão pela linha de interrupção externa número 11. O canal 9 do ADC é o que corresponde ao pino PB1, definindo-se um tempo de amostragem para o *sample&hold* de 1 ciclo e meio da frequência de relógio do ADC, o tempo mais curto.

Segue-se a configuração do pino PB1 em modo de entrada analógica, e da linha de interrupção externa número 11 ligando ao pino PB11 para gerar um evento a cada flanco ascendente do sinal na sua entrada. Este evento provoca o início de uma conversão do ADC sem intervenção do processador.

Por fim são inicializados os indicadores necessários para o funcionamento dos conversores e a função de interrupção gerada pelo sinal 'AD_sp', e são ativadas as interrupções do ADC no módulo NVIC. O código das duas funções de interrupção é o seguinte.

```
void ADC_Start_IRQ(){
  data_i = 0;
}
extern "C"
{
  void ADC1_2_IRQHandler(void){
    if( !sequence_finish && !externalADC){
      data[data_i++] = HAL_ADC_GetValue(&hadc);
      if(data_i >= 128){
         HAL_ADC_Stop_IT(&hadc);
         sequence_finish = true;
      }
    }else{
      HAL_ADC_GetValue(&hadc); // Clears EOC flag
    }
    HAL_NVIC_ClearPendingIRQ(ADC1_2_IRQn);
  }
}
```

A função de interrupção do sinal ‘AD_sp’ reinicia a contagem das amostras, sendo provocada pelo flanco descendente do pulso. O motivo desta opção pode ser compreendido através do diagrama temporal dos sinais do sensor, apresentado de novo e de forma resumida na Figura 92, sendo ilustrada a ocorrência de pulsos do sinal ‘AD_trig’ antes do início da sequência de vídeo e também após o flanco ascendente de ‘AD_sp’. A primeira amostra válida irá então ocorrer após o fim do pulso do sinal ‘AD_sp’, e a respetiva interrupção deverá ser o mais breve possível para permitir uma frequência de relógio do sensor mais elevada.

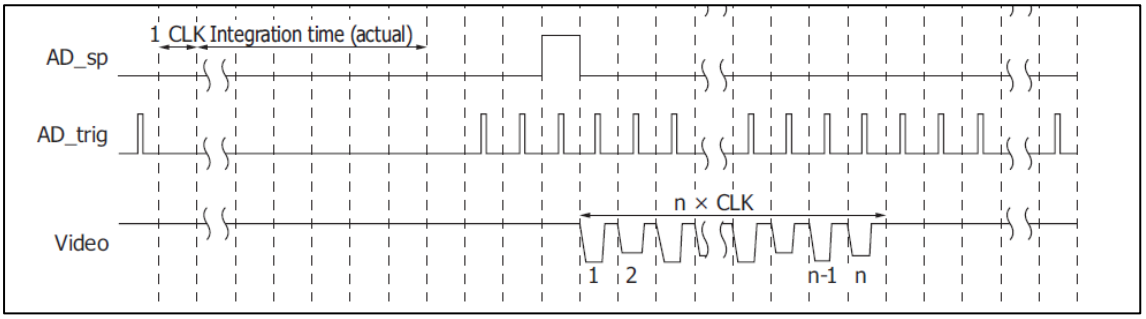


Figura 92 – Diagrama temporal dos sinais de saída do sensor de imagem, adaptado.

A função de interrupção do ADC, que é chamada no fim de cada conversão, obtém o resultado e guarda num *array*, incrementando o contador. Verifica também se as 128 amostras já foram obtidas, afetando o indicador de fim da sequência e parando o funcionamento do conversor.

Controlador do ADC externo

O integrado do ADC externo foi o MCP33131, tratando-se de um conversor de aproximações sucessivas de 16 bits capaz de frequências de amostragem até 1 Msps. Este foi montado numa configuração *single-ended* como ilustrado na Figura 93, seguindo as indicações da ficha técnica [39]. Nesta configuração a gama de valores da

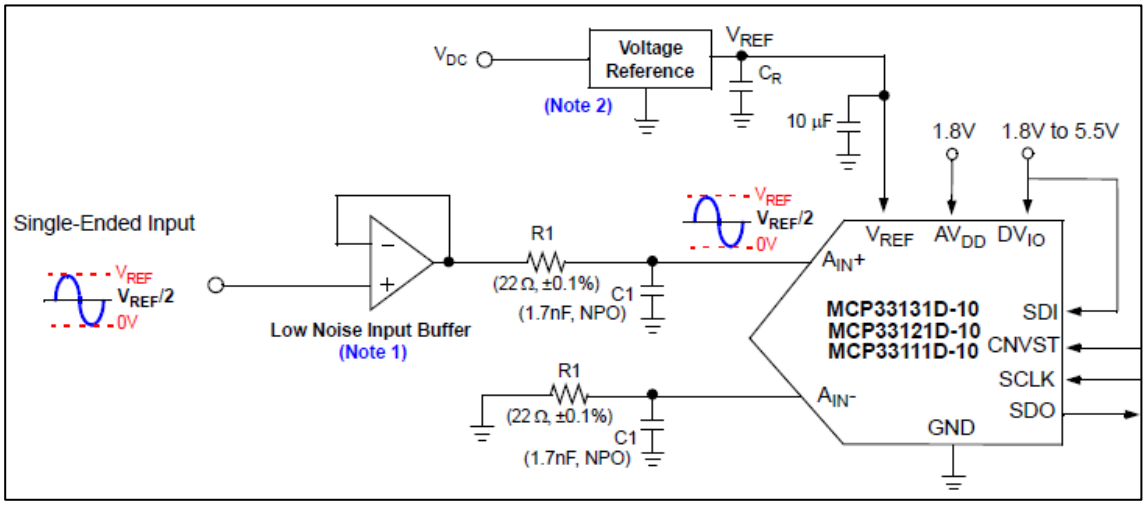


Figura 93 – Circuito da montagem single-ended do ADC, retirado de [39], figura 6-7.

conversão é exclusivamente positiva pelo que o bit de maior peso, utilizado para indicar o sinal, será sempre zero. A resolução do ADC nesta aplicação será então de 15 bits. Uma alimentação de 1,8 V é necessária, tendo sido utilizado o integrado MCP1700T-1802E/TT para esse efeito (ficha técnica em [40]).

O código do controlador encontra-se também no Anexo B.7.

A função de inicialização `ExternalADC_Setup()` trata apenas da configuração dos pinos GPIO para seleção do dispositivo no barramento SPI e para leitura do sinal de gatilho. A função `ExternalADC_Enable()` começa por desativar o ADC interno, parando qualquer conversão em curso e afetando os indicadores, configurando depois o pino PB11 para gerar a interrupção que controla o processo de cada conversão. A função `ExternalADC_Disable()` faz o processo inverso e volta a configurar o ADC interno.

A função `ADC_Prepare()`, no caso do conversor externo, inicia o barramento SPI no modo 1 (a amostragem dos dados é feita na transição descendente do sinal de relógio) e coloca o sinal MOSI ativo (ambos necessários segundo a ficha técnica, ver notas da Figura 94). Depois inativa o sinal de seleção do dispositivo, colocando o ADC no estado de aquisição.

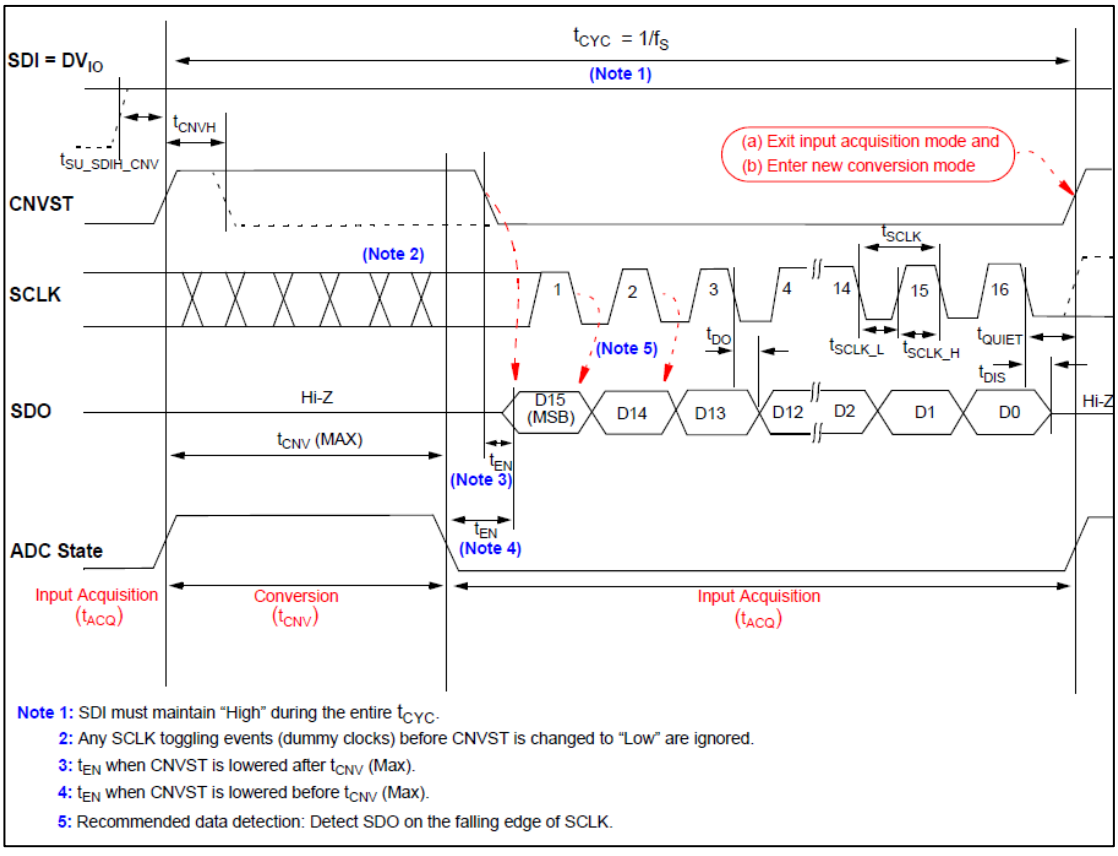


Figura 94 – Diagrama temporal dos sinais do ADC, retirado de [39].

A Figura 95 ilustra o diagrama de blocos do ADC externo, e o papel dos sinais auxiliares na ocorrência de rotinas de interrupção para desencadear o processo de conversão da sequência de vídeo. A função de interrupção do sinal 'AD_sp' continua apenas a reiniciar o contador das conversões

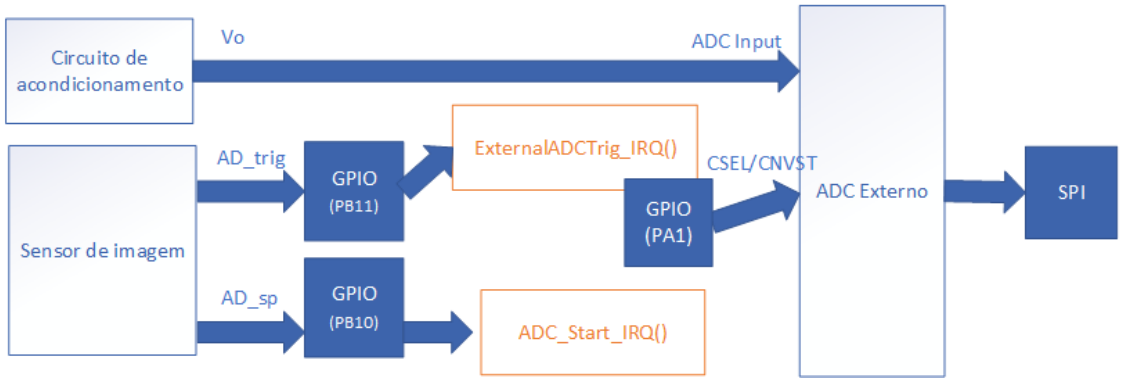


Figura 95 – Diagrama de blocos do ADC externo, mostrando ligações com o microcontrolador e restantes blocos.

A função que atende as interrupções geradas pelo sinal 'AD_trig', `ExternalADCTrig_IRQ()`, provoca o início de uma conversão elevando o sinal de seleção do dispositivo. O sinal é colocado novamente inativo, sendo realizada uma espera superior ao tempo máximo de conversão (ver Figura 96), seguindo-se a obtenção dos dados na saída. Isto repete-se as vezes necessárias, até que as 128 amostras sejam adquiridas, afetando-se o indicador de fim da sequência e libertando o barramento SPI.

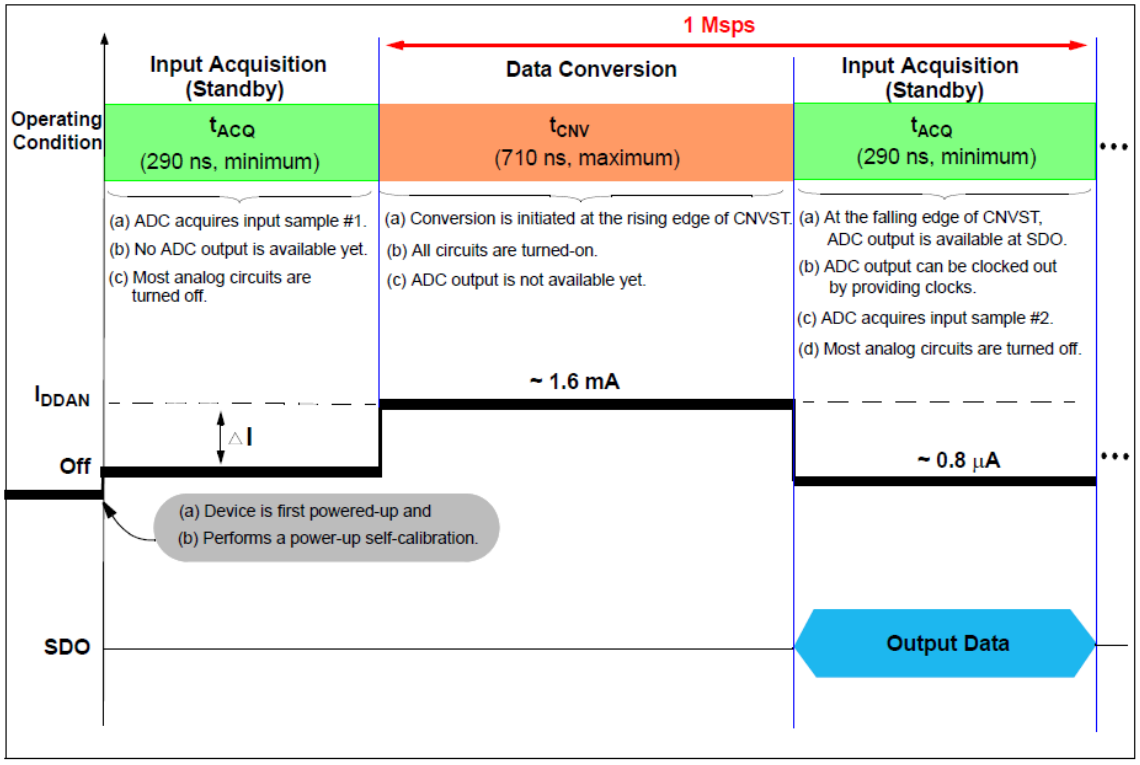


Figura 96 – Diagrama da sequência de operação do dispositivo, retirado de [39].

4. Controlador da saída analógica

A saída analógica foi introduzida como uma forma de leitura alternativa dos resultados da aquisição de imagem. Esta é alimentada por um DAC paralelo de 8 bits e funciona numa gama de tensões entre 0 e 3,3 V, com 256 níveis, através dos quais apresenta a sequência de informação dos canais processados.

A informação é antecedida por uma sequência de sincronismo de dois pulsos com *duty-cycle* de 50% e cujo período ativo é idêntico ao período do pulso para cada canal. A Figura 97 ilustra o diagrama de blocos deste subsistema e exemplifica o sinal gerado.

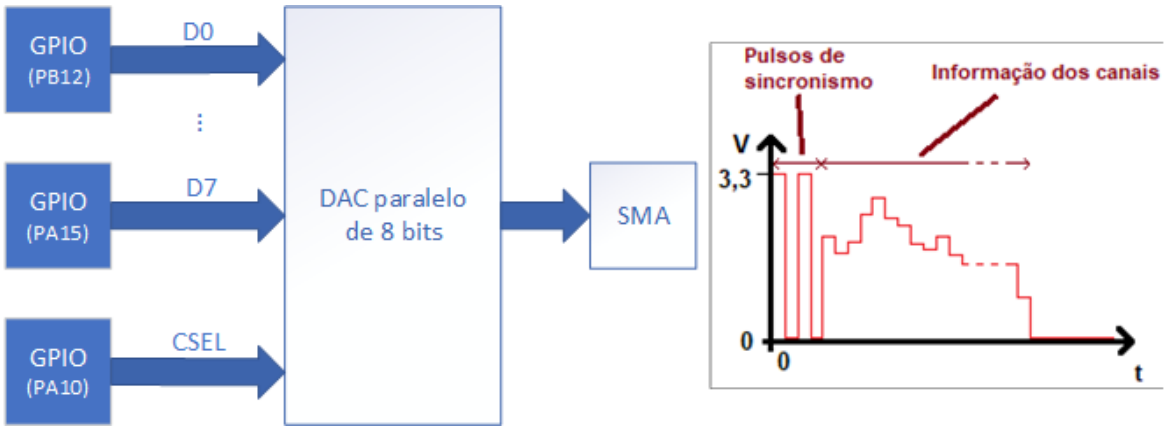


Figura 97 – Diagrama de blocos do DAC para a saída analógica e ilustração do sinal.

O DAC utilizado é o IC AD7801. O seu controlador foi escrito com base nas funções do Arduino para afetar os pinos de GPIO e fazer a temporização. O código encontra-se no Anexo B.8. A Figura 98 ilustra o diagrama de blocos do dispositivo.

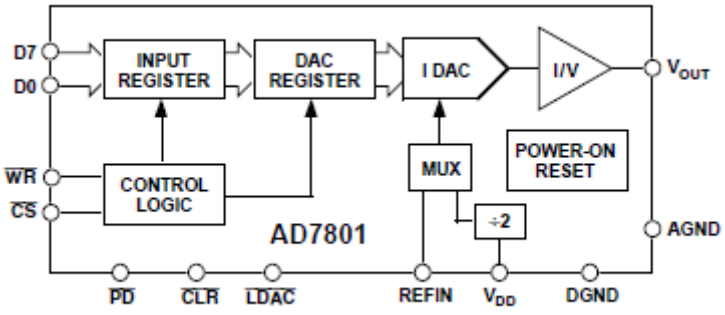


Figura 98 – Diagrama de blocos do dispositivo AD7801 [41].

Seguindo as indicações da ficha técnica [41], os sinais $\overline{\mathrm{CLR}}$ e $\overline{\mathrm{PD}}$ poderão ser ligados a V_{DD}, enquanto o sinal $\overline{\mathrm{LDAC}}$ poderá ser ligado à massa. Os sinais $\overline{\mathrm{CW}}$ e $\overline{\mathrm{WR}}$ poderão receber o sinal do mesmo pino de GPIO para a seleção do dispositivo, efetuando a conversão da palavra digital para um nível de tensão.

A função `AnalogOut_Setup()` configura os pinos de GPIO para a palavra de 8 bits e para o sinal de seleção do dispositivo. Esta afeta também a variável que guarda a frequência do sinal gerado. A função `AnalogOut_WriteWord()` permite escrever um valor entre 0 e 3,3 V a partir de uma palavra digital entre 0 e 255. A tensão na saída é alterada no flanco descendente do sinal de seleção do dispositivo. A função `AnalogOut_WriteArray()` permite escrever uma sequência de palavras, passada através de argumentos com um ponteiro e a dimensão de um *array*, à frequência configurada, após um sinal de sincronismo. Por fim a função `AnalogOut_SetFreq()` configura a frequência do sinal.

A Figura 99 mostra um sinal gerado pela saída analógica após aquisição de um sinal de teste. A solução desenvolvida para testes do módulo de aquisição será descrita no capítulo **Erro! A origem da referência não foi encontrada.**;

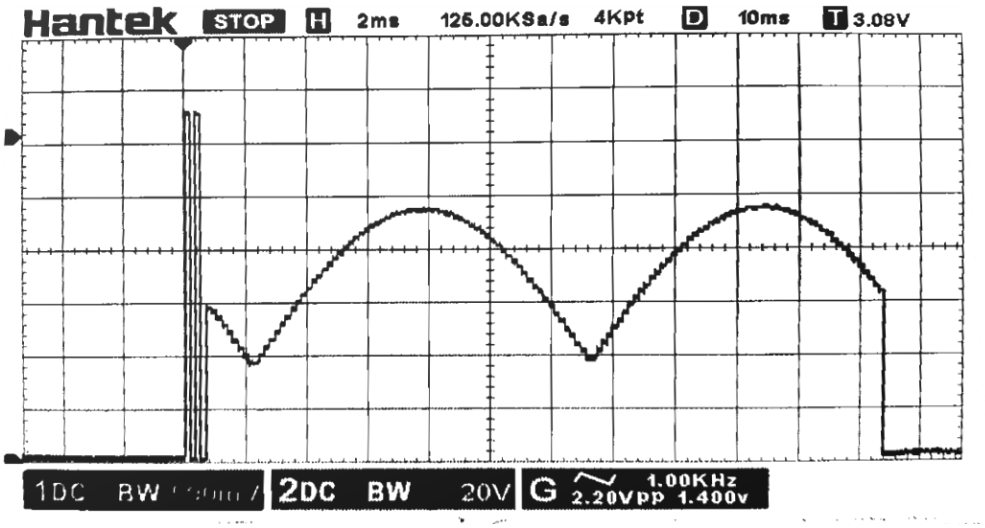


Figura 99 – Imagem do osciloscópio da saída analógica a 6400 Hz. Aquisição de sinal de teste.

5. Controlador do sinal PWM para controlo do laser

Prevendo a utilidade do controlo da intensidade do laser que alimenta o circuito de fotónica a partir do mesmo dispositivo que efetua a deteção do sinal, foi incluído no módulo de aquisição um segundo conector SMA para esse efeito. Durante o período de integração do sensor de imagem, é gerado um sinal PWM que liga e controla a intensidade da fonte. A Figura 100 ilustra a ligação do periférico do microcontrolador designado para esta função e o sinal gerado.

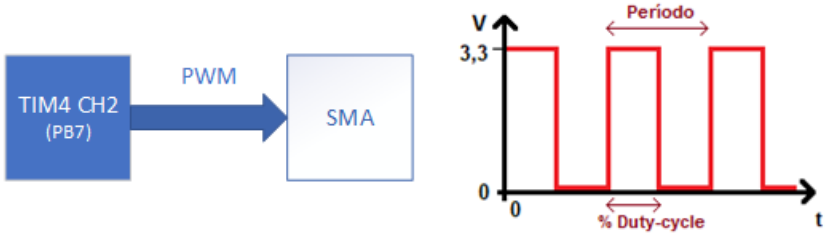


Figura 100 – Diagrama de blocos da saída PWM e ilustração do sinal.

O controlador para a fonte de luz utiliza o canal 2 do temporizador 4, com saída no pino PB7, para gerar um sinal PWM. A configuração é feita de forma semelhante à dos temporizadores dos sinais de relógio e de *reset* do sensor de imagem. O código completo encontra-se no Anexo B.9. Os registos do temporizador são afetados da seguinte forma.

```
/* Timer config */
TIM4->PSC = 71; // base temporal de 1 microssegundo
TIM4->CCMR1 = 0b110 << 12 | 1<<10;  // PWM MODE 1
TIM4->CCER = 0;
TIM4->CR2 = 0;
TIM4->CNT = 0;
TIM4->ARR = period_us - 1;
TIM4->CCR2 = period_us*dutyCycle;
TIM4->CR1 = 1; // Counter enable bit
```

Desta vez a base temporal utilizada é de 1 microssegundo, dividindo o relógio de 72 MHz por 72 colocando o valor 71 no registo TIM4_PSC. Isto permite a configuração de qualquer valor inteiro de *duty-cycle* entre 0 e 100 a uma frequência igual ou inferior a 10 kHz. O modo colocado no registo TIM4_CCMR1 corresponde ao modo PWM 1, em que a saída é ativa enquanto o contador tem um valor inferior ao registo de comparação do canal, TIM4_CCR2, e inativa em caso contrário. O período pretendido, em microssegundos, é colocado no registo TIM4_ARR. O ciclo de trabalho é definido pela percentagem do valor de TIM4_ARR colocado em TIM4_CCR2. A função `PWM_Config()` trata de configurar os registos com um novo período e ciclo de trabalho. As funções `PWM_Start()` E `PWM_Stop()` ligam e desligam o sinal de PWM.

A Figura 101, Figura 102 e Figura 103 mostram o sinal PWM no osciloscópio, com um período de 1 milissegundo e ciclos de trabalho de 50%, 10% e 90%, respectivamente.

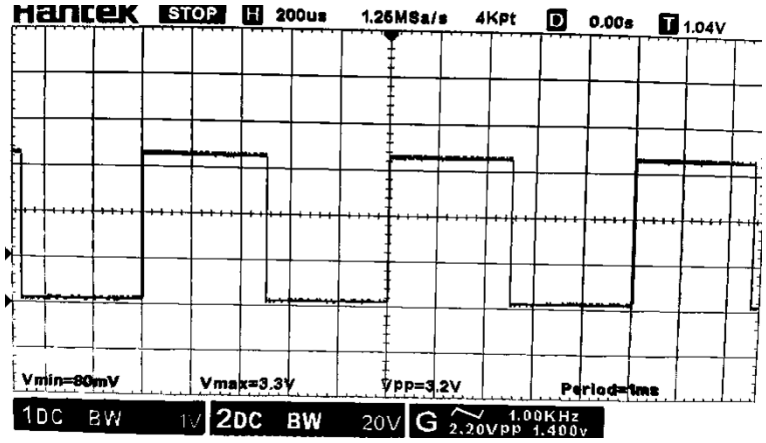


Figura 101 – Imagem do osciloscópio. Sinal PWM com período de 1 ms e ciclo de trabalho de 50%.

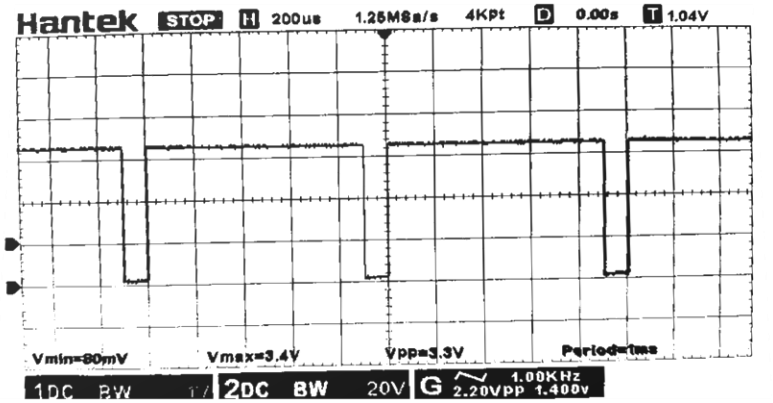


Figura 102 - Imagem do osciloscópio. Sinal PWM com período de 1 ms e ciclo de trabalho de 90%.

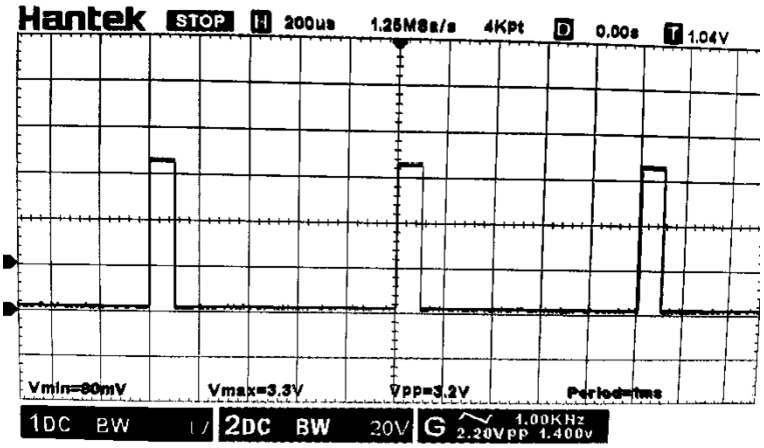


Figura 103 - Imagem do osciloscópio. Sinal PWM com período de 1 ms e ciclo de trabalho de 10%.

6. Funções e configurações do sistema

Nesta secção é explicado o módulo responsável pelo controlo do dispositivo (`DeviceFunctions.cpp`), mantendo uma estrutura de dados de configuração e disponibilizando funções de afetação e verificação dos parâmetros. As funções principais de aquisição, processamento e transmissão também são resolvidas neste módulo, assim como o mapeamento dos píxeis nos canais de saída e número de canais.

Estrutura de configuração

A estrutura de configuração do dispositivo, do tipo `SysCfgTypedef`, contém as seguintes variáveis (Tabela 12). Esta é definida no cabeçalho do módulo, disponível no Anexo B.10.

Tabela 12 – Variáveis da estrutura de configuração, tipos e valores predefinidos.

Variável	Tipo	Valor predefinido
clockFrequency	uint32_t	100000
integrationTime_us	uint32_t	10
acqInterval_ms	uint32_t	1000
acqTimeout_ms	uint32_t	1000
pwm_period_us	uint32_t	1000
pga_vref_mV	uint32_t	1700
SPIfrequency	uint32_t	10000000
analogOutFreq	uint32_t	10000
nChannels	uint8_t	128
channelLimits	uint8_t [256]	{0,0,1,1,…,127,127}
pwm_dutycycle	uint8_t	50
pga_gainSel	uint8_t	0
cselPin	bool	false
continuousModeOn	bool	false
equalChannels	bool	false
pwm_on	bool	false
analogOutOn	bool	true
externalADC	bool	false
strNotBytes	bool	true

As funções que configuram os parâmetros do dispositivo atualizam os membros desta estrutura. É também definida uma constante do tipo da estrutura de configuração com os parâmetros predefinidos, utilizada na inicialização automática do sistema.

Configuração do sistema

A função de inicialização do sistema, chama todas as funções de inicialização dos controladores e atualiza a estrutura de configuração. Após a inicialização do sistema, os parâmetros podem ser alterados ou verificados com um conjunto de funções, cujo código está disponível nos Anexos B.11 e B.12.

As funções de configuração limitam os valores passados como argumento na gama de valores válidos e retornam o valor atribuído. Para cada parâmetro há também uma função para obter o valor atual, sem o alterar. A maioria das funções utiliza os controladores desenvolvidos e descritos no subcapítulo anterior. A exceção são as funções que tratam parâmetros de nível superior, como o formato do resultado, em *bytes* ou em *string*, ou a configuração do mapeamento dos píxeis para os canais de saída.

A função `Device_Channels()` configura o número de canais apresentados após o processamento. A função `Device_ChannelLimits()` permite fazer o mapeamento dos píxeis para cada canal em blocos contíguos, definindo os limites inferior e superior. A função `Device_EqualChannels()` divide os 128 píxeis do sensor em grupos de igual dimensão, consoante o número total de canais.

A função `Device_OutputModeString()` afeta o parâmetro `strNotBytes` que indica o formato dos dados transmitidos.

As seguintes funções controlam parâmetros relacionados com a aquisição de sinal. A função `Device_ContinuousMode()` liga ou desliga o modo de aquisição continua. O intervalo entre aquisições neste modo é configurado com a função `Device_MeasureInterval()`. A função `Device_MeasureTimeout()` define o tempo que o programa espera pelo fim da aquisição da sequência de vídeo, sendo este passado por parâmetro à função do controlador do ADC que realiza a espera bloqueante, tanto no modo de aquisição contínua como no de aquisição única.

Funções de aquisição, processamento e transmissão

Foram definidas um conjunto de funções que efetuam as operações principais do módulo de aquisição. O código encontra-se no Anexo B.13.

A função `Device_Acquisition()` trata o processo de aquisição. Chama a função de preparação do ADC e gera o sinal de *reset* do sensor de imagem com o tempo de integração configurado. Se o sinal PWM não estiver já ativo, este é ligado durante o tempo de integração. De seguida a função bloqueia em espera do resultado das conversões. Em caso de erro retorna o código de erro correspondente. Caso contrário, obtém os dados da sequência de vídeo e guarda-os num *array* para processar, retornando zero.

A função `Device_Process()` realiza a média das amostras dos píxeis de cada canal, integrando cada conjunto de valores e dividindo pelo tamanho do conjunto. Esta operação é realizada com tipos inteiros, por ser mais rápida do que usando virgula flutuante[9]. No caso do ADC de 12 bits, o valor é deslocado 3 bits para a esquerda para ficar na mesma gama de valores que o ADC externo, com 15 bits. É também preenchido um *array* com os valores numa dimensão de 8 bits, para utilizar na saída analógica. No fim é feita uma conversão para o valor correspondente em milivolts, guardado num tipo de virgula flutuante, ficando o resultado disponível nos dois formatos.

A função de transmissão, `Device_Transmit()` coloca os resultados na porta série, em forma de caracteres ASCII ou valores em binário consoante o valor do parâmetro `strNotBytes`. A transmissão da sequência num formato binário é mais eficiente e não perde resolução, podendo o valor de cada canal ser enviado em 2 bytes. A representação de valores entre 0,00 e 3300,00 em *string,* com um caracter de fim de

[9] O processador Cortex-M3 não possui uma unidade de calculo com virgula flutuante, podendo essa funcionalidade estar presente em alguns microcontroladores com um CPU Cortex-M4.

linha, obriga à utilização de 5 a 8 bytes para transmitir o valor de cada canal. Com uma porta série configurada à frequência de 115200 Hz, com o formato de *string* a transmissão dos valores de 128 canais poderia levar entre 5,5 ms e 8,9 ms. No formato de sequência binária a mesma informação demoraria cerca de 2,3 ms a ser transmitida. Por isso, a opção binária permite uma aquisição continua com um intervalo menor, especialmente para um número de canais elevado. A função termina com a geração do sinal da saída analógica, caso esta opção esteja ativa.

A função `Device_ContAcquisition()` chama as funções anteriores em sequência, se verificar que já passou o intervalo programado desde a última aquisição e que o modo de aquisição continua está ativo. Para isso armazena o momento da última aquisição numa variável estática. Esta função irá ser chamada no laço principal.

7. Biblioteca SCPI

A interface de programação desenvolvida disponibiliza uma função de inicialização, uma função para adicionar comandos a um nó da árvore ou ao nó comum, passando em argumento as funções a executar em situação de SET ou QUERY, e uma função que recebe uma linha de texto e executa os comandos contidos de acordo com a sintaxe do padrão e a estrutura configurada.

O código do cabeçalho do módulo `SCPI` encontra-se no Anexo B.14.

O módulo define um tipo `NodeTypedef`, que contém uma estrutura cujos membros incluem: o nome do nó, completo e abreviado, um *array* de ponteiros para nós filhos com dimensão fixa, o número de filhos presente no *array*, e ponteiros para uma função de SET e para outra de QUERY.

A função de inicialização aloca memória para a instanciação de dois nós, o nó comum e o nó de raiz da arvore. Estes nós não executam quaisquer funções quando são chamados.

Três variáveis estáticas armazenam ponteiros para o nó comum, o nó de raiz e o nó atual, que guarda o último nó chamado, inicializado com a raiz da árvore. As três funções `SCPI_GetRootNode()`, `SCPI_GetCommonNode()` e `SCPI_GetCurrentNode()` retornam os valores destas variáveis. A função `GetNodeIn()` permite obter o ponteiro para o nó filho de um nó conhecido, através do ponteiro para o nó pai e de um nome do nó filho.

A função `SCPI_AddNode`() permite criar e adicionar um nó filho em qualquer nó (desde que este não exceda o limite da dimensão do seu *array* de ponteiros), definindo o nome completo e abreviado, e as funções chamadas nas situações de SET ou de QUERY.

A fim de exemplificar, o seguinte bloco de código poderia ser utilizado para construir a estrutura de comandos apresentada anteriormente na Figura 27 no capítulo **Erro! A origem da referência não foi encontrada.**.

```
void RST_Func(*arg){(…)} // Definição das funções a chamar
(…)
int main{
  SCPI_Initialize();

  NodeTypedef *common = SCPI_GetCommonNode();
  SCPI_AddNode(common, "RST",  NULL, RST_Func, DoNothing);
  SCPI_AddNode(common, "IDN",  NULL, IDN_Func, DoNothing);

  NodeTypedef *root = SCPI_GetRootNode();
  SCPI_AddNode(root, "configure", "conf", DoNothing, DoNothing);
  SCPI_AddNode(root, "measure",   "meas", DoNothing, DoNothing);

  NodeTypedef *confNode, *measNode;
  confNode = GetNodeIn(root, "conf");
  measNode = GetNodeIn(root, "meas");

  SCPI_AddNode(confNode, "clock",   NULL, ClockSet, ClockQuery);
  SCPI_AddNode(confNode, "int_time, NULL, IntTSet,  IntTQuery);
  SCPI_AddNode(measNode, "datatype",NULL, DataTSet, DataTQuery);
(…)
}
```

Após a inicialização, a função `SCPI_RunCommand()` executa um comando SCPI. O código desta encontra-se no Anexo B.15.

Trata-se de uma função recursiva que lê a *string* passada por argumento, passo a passo. A função procura caracteres que fazem parte da sintaxe do padrão SCPI, identificando o tipo de comando (comum ou da árvore), procura a palavra-chave seguinte e chama a função auxiliar `DoFunctionAndMove`. A última verifica a existência da palavra-chave na estrutura do nó passado como argumento e, caso positivo, obtém os argumentos da função do nó (SET ou QUERY), chama-a, atualiza o nó atual e retorna o resto do comando.

8. Integração da biblioteca SCPI no sistema

O módulo `DeviceSCPI.cpp` constrói a estrutura de comandos necessária para tornar as funções do dispositivo (`DeviceFunctions.cpp`) acessíveis através da comunicação série, recorrendo ao módulo SCPI que foi desenvolvido (`scpi.c`). O

cabeçalho do módulo define os nomes dos comandos SCPI implementados e encontra-se no Anexo B.16. A função `SCPI_Setup()` trata todo o processo de inicialização do SCPI e encontra-se no Anexo B.17. A tabela seguinte resume todos os comandos.

Tabela 13 – Tabela completa dos comandos SCPI implementados.

Posição	**Comando**	**Comentário**
Comum (*)	IDN	Devolve a identificação do dispositivo.
	RST	Reinicia o dispositivo.
	ECHO	Define/Pergunta a opção de eco.
	pThis	Devolve o nome do nó atual (p/ debug).
	pAll	Devolve a estrutura da árvore de comandos SCPI (p/ debug).
Raíz (:)	CONFigure	Nó de configuração. Não faz nenhuma ação.
	MEASure	Nó de medição. Não faz nenhuma ação.
	TEST	Nó de teste. Não faz nenhuma ação.
CONFigure	DEFault	Coloca as definições predefinidas do módulo.
	clock	Define/Pergunta a frequência do sinal de relógio.
	intg	Define/Pergunta a duração do pulso do sinal de *reset*.
	csel	Define/Pergunta a opção de ganho do sensor.
	n_ch	Define/Pergunta o número de canais de deteção.
	ch_lmt	Define/Pergunta os limites de um canal.
	ch_equal	Define/Pergunta a opção de distribuição dos píxeis nos canais.
	pwm_period	Define/Pergunta o período do sinal PWM.
	pwm_duty	Define/Pergunta o ciclo de trabalho do sinal PWM.
	pwm_state	Define/Pergunta o estado do sinal PWM.
	pga_gain	Define/Pergunta o ganho do PGA.
	pga_vref	Define/Pergunta a tensão Vref do PGA.
	pga_conf	Coloca uma configuração combinada de ganho e tensão Vref.
	adc_ext	Define/Pergunta a opção de seleção do ADC externo.
	spi_freq	Define/Pergunta a frequência do barramento SPI.
MEASure	single	Executa uma aquisição única.
	cont	Define/Pergunta o modo de aquisição contínua.
	interval	Define/Pergunta o intervalo entre aquisições.
	timeout	Define/Pergunta o '*timeout*' para uma sequência de amostras.
	anlg_out	Define/Pergunta a ativação da saída analógica.
	anlg_freq	Define/Pergunta a frequência da saída analógica.
	string	Define/Pergunta o formato de saída em caracteres ASCII.
	bytes	Define/Pergunta o formato de saída em valores binários.
TEST	resetpulse	Provoca um pulso do sinal de *reset*.
	pwm_pulse	Provoca um pulso do sinal de PWM.
	anlg_val	Coloca um valor de tensão na saída analógica.
	adc_conv	Faz uma conversão com o ADC ativo.
	adc_cal	Faz a calibração do ADC ativo.
	i2c_dac	Verifica se o DAC do circuito de acondicionamento responde.

O código do programa principal encontra-se no Anexo B.18. Este tem apenas duas funções: `setup()` e `loop()`.

A função `setup()` é chamada uma vez e faz a inicialização da comunicação série à frequência de 115200 Hz, dos controladores do sensor de imagem e do modulo do protocolo SCPI.

A função `loop()` é chamada repetidamente após a inicialização. Verifica a existência de caracteres na porta série a uma frequência de 20 Hz, lendo e correndo os comandos que nesta forem introduzidos. No restante tempo, a cada 1 ms (no mínimo) chama a função de aquisição continua.

Anexo B.1 – Cabeçalho do módulo 'MyDrivers'

```
#ifndef _MYDRIVERS_H_
#define _MYDRIVERS_H_

#define HAL_EXTI_MODULE_ENABLED

#include <Arduino.h>
#include "SPI.h"

#define PIN_MOSI       PA7
#define PIN_MISO       PA6
#define PIN_SCK        PA5
#define PIN_CSEL       PA4
#define PIN_CLOCK      PB0
#define PIN_ADC_IN     PB1
#define PIN_ADC_START  PB10
#define PIN_ADC_TRIG   PB11
#define PIN_RESET      PA3
#define PIN_PWM        PB7
#define PIN_CS_DAC     PA10
#define PIN_CS_PGA     PA2
#define PIN_CS_ADC     PA1

#define PIN_DAC_0      PB12
#define PIN_DAC_1      PB13
#define PIN_DAC_2      PB14
#define PIN_DAC_3      PB15
#define PIN_DAC_4      PB5
#define PIN_DAC_5      PB4
#define PIN_DAC_6      PB3
#define PIN_DAC_7      PA15

#define ADC_MAX_VAL       4095
#define EXT_ADC_MAX_VAL  32767    // 15 bits are used in single-
ended mode of operation
#define ADC_REF_MILIVOLT  3300

extern volatile uint16_t data[128];

/* Clock pin functions */
void Clock_Setup(float freq);
void Clock_Start();
void Clock_Stop();
void Clock_Freq(float freq);

/* Reset pin functions */
void Reset_Setup();
void Reset_Pulse_us(uint32_t us);

/* Cf Sel pin functions */
void CfSel_Setup();
void CfSel_On();
void CfSel_Off();
```

```
/* PWM pin functions */
void PWM_Setup();
void PWM_Config(uint32_t period_us, float dutyCycle);
void PWM_Start();
void PWM_Stop();

/* Internal and external ADC functions */
void ADC_Setup();
void ADC_Prepare();
int ADC_WaitSequence(uint32_t timeout);
void ADC_GetData(uint16_t newData[128]);
void ExternalADC_Setup();
void ExternalADC_Calibrate();
void ExternalADC_Enable();
void ExternalADC_Disable();
int16_t ADC_SingleConvertion();

void SetSPIFreq(uint32_t freq);

/* Conditioning circuit functions  */
void PGA_Setup();
void PGA_SetGain(const uint8_t g);
void Vref_Setup();
bool Vref_Set(uint32_t val);
bool Vref_TestDevice();
bool SoftI2C_DeviceReady(uint8_t addr);

/* Analog output functions */
void AnalogOut_Setup(uint32_t freq);
void AnalogOut_SetFreq(uint32_t freq);
void AnalogOut_WriteWord(uint8_t value);
void AnalogOut_WriteArray(uint8_t *data, uint32_t n);

#endif
```

Anexo B.2 – Controlador para sinal de relógio

```
void Clock_Setup(float freq){

  if(freq < 300)
    freq = 300;
  else if(freq > 2e6)
    freq = 2e6;

  /* Clock enable */
  __HAL_RCC_TIM3_CLK_ENABLE();

  /* Pin GPIO Config*/
  GPIO_InitTypeDef gpio_init;
  gpio_init.Pin = GPIO_PIN_0;
  gpio_init.Mode   = GPIO_MODE_AF_PP;
  gpio_init.Pull   = GPIO_NOPULL;
  gpio_init.Speed  = GPIO_SPEED_FREQ_MEDIUM;
  HAL_GPIO_Init(GPIOB, &gpio_init);

  /* Timer config */
  TIM3->CR1 = 0;
  TIM3->CR2 = 0;
  TIM3->CCR3 = 0;
  TIM3->CCMR2 = 0b011 << 4; // 011 - OC1REF toggles when
TIMx_CNT=TIMx_CCR1
  TIM3->CCER = 1 << 8; // Capture/Compare 3 output enable
  TIM3->CNT = 0;
  TIM3->PSC = (uint16_t)((18e6/freq)-1);
  TIM3->ARR = 1;

  AFIO->MAPR |= (0b10 << 10); // REMAP TIM3 CH2 to PB5, avoid
conflict with SPI1 MISO
}

void Clock_Start(){
  TIM3->CR1 |= 1;
}
void Clock_Stop(){
  TIM3->CR1 &= ~1;
}
void Clock_Freq(float freq){
  if(freq < 300)
    freq = 300;
  else if(freq > 2e6)
    freq = 2e6;
  TIM3->PSC = (uint16_t)((18e6/freq)-1);
}
```

Anexo B.3 – Controlador para sinal de *reset*

```
void Reset_Setup(){
  __HAL_RCC_TIM2_CLK_ENABLE();

  /* Pin GPIO Config*/
  GPIO_InitTypeDef gpio_init;
  gpio_init.Pin = GPIO_PIN_3;
  gpio_init.Mode   = GPIO_MODE_AF_PP;
  gpio_init.Pull   = GPIO_NOPULL;
  gpio_init.Speed  = GPIO_SPEED_FREQ_MEDIUM;
  HAL_GPIO_Init(GPIOA, &gpio_init);

  TIM2->CCMR2 = 0b111 << 12 | 1<<10;   // PWM MODE 2, ch active
while TIMx_CNT > TIMx_CCRx in upcounting, fast enable
  TIM2->CCER = 1 << 12; // OUTPUT ENABLE CH4
  TIM2->CR2 = 0;
  TIM2->CR1 = 1<<3; // OPM
}

void Reset_Pulse_us(uint32_t us){
  if(us > 5000000)
    us = 5000000;

  if(us > 1000)
  {
    TIM2->PSC = 7199; // base temporal de 100 microssegundos
    TIM2->ARR = us/100;
  }
  else
  {
    TIM2->PSC = 71; // base temporal de 1 microssegundo
    TIM2->ARR = us;
  }
  TIM2->CNT = 0;
  TIM2->CCR4 = 1; //
  TIM2->CR1 |= 1; // CEN
}
```

Anexo B.4 – Controlador para sinal CfSel e PGA

```
/* Controlador para sinal CfSel*/
void CfSel_Setup(){
  pinMode(PIN_CSEL, OUTPUT);
}

void CfSel_On(){
  digitalWrite(PIN_CSEL, HIGH);
}

void CfSel_Off(){
 digitalWrite(PIN_CSEL, LOW);
}



/* Controlador para PGA */

// PGA pode usar modo SPI 0,0 ou 1,1
const uint8_t PGA_INSTR_SHUTDOWN = 0b00100000;
const uint8_t PGA_INSTR_GAIN     = 0b01000000;

void PGA_Setup(){
  pinMode(PIN_CS_PGA, OUTPUT);
  digitalWrite(PIN_CS_PGA, HIGH);
  PGA_SetGain(0);
}

void PGA_SetGain(const uint8_t gainSel){
  digitalWrite(PIN_CS_PGA, LOW);

  SPI.beginTransaction(SPISettings(spiFreq,             MSBFIRST,
SPI_MODE0));
  SPI.transfer( PGA_INSTR_GAIN );
  SPI.transfer( gainSel & 0b111 );
  SPI.endTransaction();

  digitalWrite(PIN_CS_PGA,HIGH);
}
```

Anexo B.5 – Controlador de I2C por software

```
#define GPOD_10MHZ      0b0101
#define GPIO_OUT_2MHz   0b0001
#define GPIO_IN_PULLUP  0b1000
#define SET_SCL_HIGH     (GPIOB->ODR |= 1<<8)
#define SET_SCL_LOW      (GPIOB->ODR &= ~(1<<8))
#define SET_SDA_HIGH     (GPIOB->ODR |= 1<<9)
#define SET_SDA_LOW      (GPIOB->ODR &= ~(1<<9))
#define DELAY_QUARTER_I2C    (delayMicroseconds(3))
//#define SOFTI2C_OD_FOR_ACK_ONLY

void SoftI2C_Setup(){
  GPIOB->CRH &= ~(0xFF);
#ifdef SOFTI2C_OD_FOR_ACK_ONLY
  GPIOB->CRH |= GPIO_OUT_2MHz | GPIO_OUT_2MHz << 4 ; // PB8 PB9
#else
  GPIOB->CRH |= GPOD_10MHZ | GPOD_10MHZ << 4 ; // PB8 PB9
#endif
  SET_SDA_HIGH;
  SET_SCL_HIGH;
}

void SoftI2C_Start(){
  //SDA E SCL HIGH
  DELAY_QUARTER_I2C;
  DELAY_QUARTER_I2C;
  SET_SDA_LOW;
  DELAY_QUARTER_I2C;
  DELAY_QUARTER_I2C;
}

void SoftI2C_Stop(){
  SET_SCL_LOW;
  DELAY_QUARTER_I2C;
  DELAY_QUARTER_I2C;
  SET_SDA_LOW;
  DELAY_QUARTER_I2C;
  DELAY_QUARTER_I2C;
  SET_SCL_HIGH;
  DELAY_QUARTER_I2C;
  DELAY_QUARTER_I2C;
  SET_SDA_HIGH;
}

bool SoftI2C_ReadAck(){
#ifdef SOFTI2C_OD_FOR_ACK_ONLY
  GPIOB->CRH &= ~(0xF<<4);
  GPIOB->CRH |= GPIO_IN_PULLUP << 4 ; // PB9
#else
  SET_SDA_HIGH;
#endif
  SET_SCL_LOW;
  DELAY_QUARTER_I2C;
  DELAY_QUARTER_I2C;
```

```
  SET_SCL_HIGH;
  DELAY_QUARTER_I2C;
  uint8_t ack = (GPIOB->IDR & (1<<9)) >> 9; // read PB9
  DELAY_QUARTER_I2C;
  SET_SCL_LOW;
#ifdef SOFTI2C_OD_FOR_ACK_ONLY
  GPIOB->CRH &= ~(0xF<<4);
  GPIOB->CRH |=  GPIO_OUT_2MHz << 4 ; // PB9
#endif
  return ack == 0;
}

void SoftI2C_SendByte(uint8_t data){
  for(int8_t i=7; i >= 0; --i){
    SET_SCL_LOW;
    DELAY_QUARTER_I2C;
    ((data & 1<<i) >> i)?SET_SDA_HIGH:SET_SDA_LOW;
    DELAY_QUARTER_I2C;
    SET_SCL_HIGH;
    DELAY_QUARTER_I2C;
    DELAY_QUARTER_I2C;
  }
}

bool SoftI2C_Send(uint8_t addr, uint8_t *pData, uint8_t n_bytes){
  SoftI2C_Start();
  SoftI2C_SendByte(addr << 1);
  if(!SoftI2C_ReadAck()){
    SoftI2C_Stop();
    return false;
  }
  for(int i = 0; i< n_bytes; ++i){
    SoftI2C_SendByte(pData[i]);
    if(!SoftI2C_ReadAck()){
      SoftI2C_Stop();
      return false;
    }
  }
  SoftI2C_Stop();
  return true;
}

bool SoftI2C_DeviceReady(uint8_t addr){
  SoftI2C_Start();
  SoftI2C_SendByte(addr << 1);
  bool ack = SoftI2C_ReadAck();
  SoftI2C_Stop();
  return ack;
}
bool SoftI2C_DeviceReady(uint8_t addr){
  SoftI2C_Start();
  SoftI2C_SendByte(addr << 1);
  bool ack = SoftI2C_ReadAck();
  SoftI2C_Stop();
  return ack;
}
```

Anexo B.6 – Controlador do DAC do circuito de acondicionamento

```
/** MCP4725 - DAC 12 bits para deslocamento de nivel **/
#define DAC_7BIT_ADD  0b1100010    //0b1100110 chip anterior
#define DAC_WRITE_CMD 0b01000000   //WRITE, PD MODE 0
#define DAC_MAX_MV 3300.0
#define DAC_MAX_BIN 4096.0


void Vref_Setup()
{
  SoftI2C_Setup();
}

bool Vref_Set(uint32_t milivolts)
{
  if(milivolts > 3299){
    milivolts = 3299;
  }
  uint16_t value = (uint16_t)(DAC_MAX_BIN/DAC_MAX_MV*milivolts +
0.5) & 0x0fff;
  uint8_t data[3];
  data[0] = DAC_WRITE_CMD;
  data[1] = (uint8_t)(value >> 4);
  data[2] = (uint8_t)(value << 4);

  return SoftI2C_Send(DAC_7BIT_ADD, data, 3);
}

bool Vref_TestDevice()
{
  return SoftI2C_DeviceReady(DAC_7BIT_ADD);
}
```

Anexo B.7 – Controlador dos conversores analógico-digitais

```
volatile uint16_t data[128];
volatile uint16_t data_i = 0;
volatile bool sequence_finish;
volatile bool sequence_error;
static bool externalADC = false;
static ADC_HandleTypeDef hadc;
static EXTI_HandleTypeDef hexti;

void ADC_Start_IRQ(){
  data_i = 0;
}

void Setup_ADC_StartPin(){
  pinMode(PIN_ADC_START, INPUT_PULLDOWN);
  attachInterrupt(digitalPinToInterrupt(PIN_ADC_START),
ADC_Start_IRQ, FALLING);
}

void ADC_Setup(){

  externalADC = false;
  sequence_finish = true;
  sequence_error = false;

  /* Config. Clock ADC */
  RCC_PeriphCLKInitTypeDef PeriphClkInit;
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
  HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
  RCC->APB2ENR |= 1 << 9;

  /* Config. ADC */
  hadc.Instance = ADC1;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc.Init.ContinuousConvMode = DISABLE;
  hadc.Init.NbrOfConversion = 1;
  hadc.Init.DiscontinuousConvMode = DISABLE; // discarded
  hadc.Init.NbrOfDiscConversion = 1; // discarded
  hadc.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_EXT_IT11; //
ADC_SOFTWARE_START;

  HAL_NVIC_SetPriority(ADC1_2_IRQn, 2, 1);
  HAL_NVIC_EnableIRQ(ADC1_2_IRQn);

  /* Config. canal ADC */
  ADC_ChannelConfTypeDef ADC_channel_config;
  ADC_channel_config.Channel = ADC_CHANNEL_9;
  ADC_channel_config.Rank = ADC_REGULAR_RANK_1;
  ADC_channel_config.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;

  HAL_ADC_Init(&hadc);
  HAL_ADC_ConfigChannel(&hadc, &ADC_channel_config);
```

```
  /* Config Pin Input */
  GPIO_InitTypeDef gpio_init;
  gpio_init.Pin    = GPIO_PIN_1;
  gpio_init.Mode   = GPIO_MODE_ANALOG;
  gpio_init.Pull   = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &gpio_init);

  /* Config EXTI pin PB11*/
  GPIO_InitTypeDef pin_b11_init;
  pin_b11_init.Pin    = GPIO_PIN_11;
  pin_b11_init.Mode   = GPIO_MODE_INPUT;
  pin_b11_init.Pull   = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &pin_b11_init);

  /* Config EXTI */
  EXTI_ConfigTypeDef extiConfig;
  extiConfig.Line = EXTI_LINE_11;
  extiConfig.Mode = EXTI_MODE_EVENT;// | EXTI_MODE_INTERRUPT;
  extiConfig.Trigger = EXTI_TRIGGER_RISING;
  extiConfig.GPIOSel = EXTI_GPIOB; //EXTI_GPIOA
  HAL_EXTI_SetConfigLine(&hexti, &extiConfig );

  /* Pin com interrupção que provoca inicio da captura */
  Setup_ADC_StartPin();
}

extern "C"
{
  void ADC1_2_IRQHandler(void){
    if( !sequence_finish && !externalADC){
      data[data_i++] = HAL_ADC_GetValue(&hadc);
      if(data_i >= 128){
         HAL_ADC_Stop_IT(&hadc);
         sequence_finish = true;
      }
    }else{
      HAL_ADC_GetValue(&hadc);
    }
    HAL_NVIC_ClearPendingIRQ(ADC1_2_IRQn);
  }
}

void ADC_Prepare(){
  sequence_finish = false;
  sequence_error = false;
  data_i = 0;
  if(externalADC){
    SPI.beginTransaction(SPISettings(spiFreq,          MSBFIRST,
SPI_MODE1));
    digitalWrite(PIN_MOSI, HIGH); // SDI signal high during whole
cycle
    digitalWrite(PIN_CS_ADC, LOW); // Chip select on.
  }else{
    HAL_ADC_Start_IT(&hadc);
  }
```

```
}
int ADC_WaitSequence(uint32_t timeout){
  uint32_t ini = millis();
  while( (millis()-ini < timeout))
  {
    if(sequence_error == true){
      return 1;
    }
    if(sequence_finish == true){
      return 0;
    }
  }
  return 2; // timeout
}

void ADC_GetData(uint16_t newData[128]){
  for(int i=0; i<128; ++i){
    newData[i] = data[i];
  }
}

int16_t ADC_SingleConvertion(){
  if(externalADC){
    uint16_t sample = 0;

    digitalWrite(PIN_CS_ADC, LOW); // Chip select on. First sample
will be taken on rising edge

    SPI.beginTransaction(SPISettings(spiFreq,          MSBFIRST,
SPI_MODE1));
    digitalWrite(PIN_MOSI, HIGH);
    delayMicroseconds(10);
    SPI.transfer(0xff); // Read garbage
    SPI.transfer(0xff);

    digitalWrite(PIN_CS_ADC, HIGH); // CONVST signal, sample is
hold and converstion starts
    digitalWrite(PIN_CS_ADC, LOW); // chip select, data transfer
    delayMicroseconds(10);
    sample = SPI.transfer(0xff) << 8;
    sample += SPI.transfer(0xff);

    SPI.endTransaction();
    digitalWrite(PIN_CS_ADC, HIGH); // last convertion, CS HIGH
to free SPI

    return sample;
  }
  else{
    HAL_ADC_Start(&hadc);
    if(HAL_ADC_PollForConversion(&hadc, 1000) == HAL_OK){
      return HAL_ADC_GetValue(&hadc) << 3;
    }else{
      return -1;
    }
  }
```

```
}
void ExternalADCTrig_IRQ(){
    if( !sequence_finish && externalADC){
      digitalWrite(PIN_CS_ADC, HIGH); // CONVST signal, sample is
hold and conversion starts
      digitalWrite(PIN_CS_ADC,  LOW);  //  chip  select,  data
transfer
      delayMicroseconds(2);//delayMicroseconds(extADCdelay);
      data[data_i] = SPI.transfer(0xff) << 8;
      data[data_i] += SPI.transfer(0xff);
      if(++data_i >= 128){
         sequence_finish = true;
         SPI.endTransaction();
         digitalWrite(PIN_CS_ADC, HIGH); // last convertion, CS
HIGH to free SPI
      }
   }
}

void ExternalADC_Setup(){
  pinMode(PIN_ADC_TRIG, INPUT_PULLUP);
  pinMode(PIN_CS_ADC, OUTPUT);
}

void ExternalADC_Calibrate(){
  SPI.beginTransaction(SPISettings(spiFreq,            MSBFIRST,
SPI_MODE1));
  digitalWrite(PIN_MOSI, HIGH); // SDI high during whole process
  digitalWrite(PIN_CS_ADC,LOW); // force low to high transistion
for conversion
  digitalWrite(PIN_CS_ADC,HIGH);
  digitalWrite(PIN_CS_ADC,LOW); // Data reading
  delayMicroseconds(5);
  SPI.transfer(0xff);
  SPI.transfer(0xff);
  for(uint16_t i = 0; i < 128; ++i){ // 1024 clocks during CNVST
= LOW for calibration command
    SPI.transfer(0xff);
  }
  delay(550); // tCAL ~ 500ms
  digitalWrite(PIN_CS_ADC, HIGH);
  SPI.endTransaction();
}
void ExternalADC_Enable(){
  sequence_finish = true;
  HAL_ADC_Stop_IT(&hadc);
  HAL_NVIC_ClearPendingIRQ(ADC1_2_IRQn);
  externalADC = true;
  attachInterrupt(digitalPinToInterrupt(PIN_ADC_TRIG),
ExternalADCTrig_IRQ, RISING);
}
void ExternalADC_Disable(){
  sequence_finish = true; externalADC = false;
  detachInterrupt(digitalPinToInterrupt(PIN_ADC_TRIG));
  ADC_Setup();
}
```

Anexo B.8 – Controlador da saída analógica

```
static uint32_t AnalogOutFrequency = 1000;

void AnalogOut_Setup(uint32_t freq){
  pinMode(PIN_DAC_0, OUTPUT);
  pinMode(PIN_DAC_1, OUTPUT);
  pinMode(PIN_DAC_2, OUTPUT);
  pinMode(PIN_DAC_3, OUTPUT);
  pinMode(PIN_DAC_4, OUTPUT);
  pinMode(PIN_DAC_5, OUTPUT);
  pinMode(PIN_DAC_6, OUTPUT);
  pinMode(PIN_DAC_7, OUTPUT);
  pinMode(PIN_CS_DAC, OUTPUT);
  AnalogOutFrequency = freq;
  AnalogOut_WriteWord(0);
}

void AnalogOut_SetFreq(uint32_t freq){
  if(freq > 100000)
    freq = 100000;
  else if(freq < 1000)
    freq = 1000;
  AnalogOutFrequency = freq;
}

void AnalogOut_WriteWord(uint8_t value){
  digitalWrite(PIN_DAC_7, value & (1<<7));
  digitalWrite(PIN_DAC_6, value & (1<<6));
  digitalWrite(PIN_DAC_5, value & (1<<5));
  digitalWrite(PIN_DAC_4, value & (1<<4));
  digitalWrite(PIN_DAC_3, value & (1<<3));
  digitalWrite(PIN_DAC_2, value & (1<<2));
  digitalWrite(PIN_DAC_1, value & (1<<1));
  digitalWrite(PIN_DAC_0, value & (1));
  digitalWrite(PIN_CS_DAC, LOW);
  delayMicroseconds(1);
  digitalWrite(PIN_CS_DAC, HIGH);
}

void AnalogOut_WriteArray(uint8_t *data, uint32_t n){
  AnalogOut_WriteWord(255);
  delayMicroseconds(1*1e6/AnalogOutFrequency);
  AnalogOut_WriteWord(0);
  delayMicroseconds(1*1e6/AnalogOutFrequency);
  AnalogOut_WriteWord(255);
  delayMicroseconds(1*1e6/AnalogOutFrequency);
  AnalogOut_WriteWord(0);
  delayMicroseconds(1*1e6/AnalogOutFrequency);
  for(uint32_t i = 0; i<n; ++i){
    AnalogOut_WriteWord(data[i]);
    delayMicroseconds(1e6/AnalogOutFrequency);
  }
  AnalogOut_WriteWord(0);
}
```

Anexo B.9 – Controlador do sinal PWM

```
void PWM_Setup(){
  __HAL_RCC_TIM4_CLK_ENABLE();

  /* Pin GPIO Config*/
  GPIO_InitTypeDef gpio_init;
  gpio_init.Pin = GPIO_PIN_7;
  gpio_init.Mode   = GPIO_MODE_AF_PP;
  gpio_init.Pull   = GPIO_NOPULL;
  gpio_init.Speed  = GPIO_SPEED_FREQ_MEDIUM;
  HAL_GPIO_Init(GPIOB, &gpio_init);

  TIM4->PSC  =  (uint16_t)(72e0-1);  //  base  temporal  de  1
microssegundo
  TIM4->CNT = 0;
  TIM4->ARR = 1000;
  TIM4->CCR2 = 1;
  TIM4->EGR |= 1; // UPDATE GENERATION
  TIM4->CCMR1 = 0b110 << 12 | 1<<10;  // PWM MODE 1, ch active
while TIMx_CNT < TIMx_CCRx in upcounting, fast enable
  TIM4->CCER = 0;
  TIM4->CR2 = 0;

}

void PWM_Config(uint32_t period_us, float dutyCycle){
  TIM4->CNT = 0;
  TIM4->ARR = period_us - 1;
  TIM4->CCR2 = period_us*dutyCycle;
}

void PWM_Start(){
  TIM4->CCER = 1 << 4;
  TIM4->CR1 = 1; // CEN
}

void PWM_Stop(){
  TIM4->CCER = 0;
  TIM4->CR1 = 0; // CEN
}
```

Anexo B.10 – Cabeçalho do módulo DeviceFunctions

```
#ifndef DEVICE_FUNCTIONS_H
#define DEVICE_FUNCTIONS_H

#include "Arduino.h"
#include "myDrivers.h"

extern "C"
{
  #include "myUtils.h"
}

/* Device configuration struct*/
typedef struct{
  uint32_t clockFrequency;
  uint32_t integrationTime_us;
  uint32_t acqInterval_ms;
  uint32_t acqTimeout_ms;
  uint32_t pwm_period_us;
  uint32_t pga_vref_mV;
  uint32_t SPIfrequency;
  uint32_t analogOutFreq;
  uint8_t nChannels;
  uint8_t channelLimits[256];
  uint8_t pwm_dutycycle;
  uint8_t pga_gainSel;
  bool cselPin;
  bool continuousModeOn;
  bool equalChannels;
  bool pwm_on;
  bool analogOutOn;
  bool externalADC;
  bool strNotBytes;
}SysCfgTypedef;

#define DEFAULT_CHANNEL_LIMITS              {0, 0, 1, 1, 2, 2, 3,
3,   4,   4,   5,   5,   6,   6,   7,   7,   8,   8,   9,   9,   \
10,10,11,11,12,12,13,13,14,14,15,15,16,16,17,17,18,18,19,19,    \
20,20,21,21,22,22,23,23,24,24,25,25,26,26,27,27,28,28,29,29,    \
30,30,31,31,32,32,33,33,34,34,35,35,36,36,37,37,38,38,39,39,    \
40,40,41,41,42,42,43,43,44,44,45,45,46,46,47,47,48,48,49,49,    \
50,50,51,51,52,52,53,53,54,54,55,55,56,56,57,57,58,58,59,59,    \
60,60,61,61,62,62,63,63,64,64,65,65,66,66,67,67,68,68,69,69,    \
70,70,71,71,72,72,73,73,74,74,75,75,76,76,77,77,78,78,79,79,    \
80,80,81,81,82,82,83,83,84,84,85,85,86,86,87,87,88,88,89,89,    \
90,90,91,91,92,92,93,93,94,94,95,95,96,96,97,97,98,98,99,99,    \
100,100,101,101,102,102,103,103,104,104,105,105,106,106,107,107,
108,108,109,109,                                                 \
110,110,111,111,112,112,113,113,114,114,115,115,116,116,117,117,
118,118,119,119,                                                 \
120,120,121,121,122,122,123,123,124,124,125,125,126,126,127,127}
```

```
const SysCfgTypedef defaultConfig = {100000, 10, 1000, 1000, 1000,
1700, 10000000, 10000, 128, DEFAULT_CHANNEL_LIMITS, 50, 0, false,
false, false, false, true, false, true};
/* Device Functions */
void Device_InitialSetup();

uint32_t Device_Clock(uint32_t val_hz);
uint32_t Device_IntgTime(uint32_t val_ms);
bool Device_CfSel(bool val_bool);
uint8_t Device_Channels(uint8_t val);
int  Device_ChannelLimits(uint8_t  ch,  uint8_t  inf_lmt,  uint8_t
up_lmt);
bool Device_EqualChannels(bool val);
uint32_t Device_PWMPeriod(uint32_t val_us);
uint8_t Device_PWMDutyCycle(uint8_t val);
bool Device_PWMState(bool val);
uint8_t Device_PGAGain(uint8_t val);
uint32_t Device_PGAVref(uint32_t val);
void Device_PGAConfig(uint32_t gain);
bool Device_ExternalADC(bool val);
uint32_t Device_SPIFrequency(uint32_t val);
bool Device_ContinuousMode(bool val);
uint32_t Device_MeasureInterval(uint32_t val_ms);
uint32_t Device_MeasureTimeout(uint32_t val_ms);
bool Device_AnalogOutput(bool val);
uint32_t Device_AnalogOutFrequency(uint32_t val_hz);
bool Device_OutputModeString(bool val);

uint32_t Device_GetClock();
uint32_t Device_GetIntgTime();
bool Device_GetCfSel();
uint8_t Device_GetChannels();
uint8_t Device_GetChannelUpperLimit(uint8_t ch);
uint8_t Device_GetChannelLowerLimit(uint8_t ch);
bool Device_GetEqualChannels();
uint32_t Device_GetPWMPeriod();
uint8_t Device_GetPWMDutyCycle();
bool Device_GetPWMState();
uint8_t Device_GetPGAGain();
uint32_t Device_GetPGAVref();
bool Device_GetExternalADC();
uint32_t Device_GetSPIFrequency();
bool Device_GetContinuousMode();
uint32_t Device_GetMeasureInterval();
uint32_t Device_GetMeasureTimeout();
bool Device_GetAnalogOutput();
uint32_t Device_GetAnalogOutFrequency();
bool Device_GetOutputModeString();

void Device_SetConfig(const SysCfgTypedef *sConfig);
int Device_Acquisition();
void Device_Process();
void Device_Transmit();
void Device_ContAcquisition();

#endif
```

Anexo B.11 – Funções de configuração de parâmetros

```
#include "DeviceFunctions.h"

static SysCfgTypedef currentConfig = defaultConfig;

void Device_InitialSetup(){
  SPI.begin();
  PWM_Setup();
  Reset_Setup();
  CfSel_Setup();
  Clock_Setup(defaultConfig.clockFrequency);
  PGA_Setup();
  Vref_Setup();
  ADC_Setup();
  ExternalADC_Setup();
  AnalogOut_Setup(defaultConfig.analogOutFreq);
  Device_SetConfig(&defaultConfig);
  Clock_Start();
}

void Device_SetConfig(const SysCfgTypedef *sConfig){
  currentConfig = *sConfig;
  SetSPIFreq(currentConfig.SPIfrequency);
  Clock_Freq((float)(currentConfig.clockFrequency));
  if(currentConfig.cselPin){
    CfSel_On();
  }else{
    CfSel_Off();
  }
  PWM_Config(currentConfig.pwm_period_us,
currentConfig.pwm_dutycycle/100.0);
  if(currentConfig.pwm_on)
    PWM_Start();
  else
    PWM_Stop();
  PGA_SetGain(currentConfig.pga_gainSel);
  Vref_Set(currentConfig.pga_vref_mV);
  AnalogOut_SetFreq(currentConfig.analogOutFreq);
  if(currentConfig.externalADC)
    ExternalADC_Enable();
  else
    ExternalADC_Disable();
}

uint32_t Device_Clock(uint32_t val_hz){
  if(val_hz > 2000000)
    val_hz = 2000000;
  else if (val_hz < 300)
    val_hz = 300;
  currentConfig.clockFrequency = val_hz;
  Clock_Freq(currentConfig.clockFrequency);
  return currentConfig.clockFrequency;
}
```

```
uint32_t Device_IntgTime(uint32_t val_us){
  if(val_us > 5000000)
    val_us = 5000000;
  else if (val_us < 1)
    val_us = 1;
  currentConfig.integrationTime_us = val_us;
  return currentConfig.integrationTime_us;
}

bool Device_CfSel(bool val_bool){
  if(val_bool){
    CfSel_On();
    currentConfig.cselPin = true;
  }else{
    CfSel_Off();
    currentConfig.cselPin = false;
  }
  return currentConfig.cselPin;
}

uint8_t Device_Channels(uint8_t val){
  if(val > 128){
    val = 128;
  }else if(val < 1){
    val = 1;
  }
  currentConfig.nChannels = val;
  if(currentConfig.equalChannels){
    Device_EqualChannels(true);
  }
  return currentConfig.nChannels;
}

int  Device_ChannelLimits(uint8_t  ch,  uint8_t  inf_lmt,  uint8_t
up_lmt){
  if(ch > 127 || inf_lmt > 127 || up_lmt > 127 || inf_lmt > up_lmt)
  {
    return -1;
  }
  currentConfig.channelLimits[ch*2] = inf_lmt;
  currentConfig.channelLimits[ch*2+1] = up_lmt;
  currentConfig.equalChannels = false;
  return 0;
}

bool Device_EqualChannels(bool val){
  if(val){
    uint32_t ch_width = 128/currentConfig.nChannels;

    for(uint32_t ch=0; ch<currentConfig.nChannels; ++ch)
    {
      currentConfig.channelLimits[ch*2] = ch*ch_width;
      currentConfig.channelLimits[ch*2+1] = (ch+1)*ch_width -1;
    }
    currentConfig.equalChannels = true;
```

```
  }
  else{
    currentConfig.equalChannels = false;
  }

  return currentConfig.equalChannels;
}

uint32_t Device_PWMPeriod(uint32_t val_us){
  if(val_us > 200000){
    val_us = 200000;
  }else if(val_us < 2){
    val_us = 2;
  }
  currentConfig.pwm_period_us = val_us;
  PWM_Config(currentConfig.pwm_period_us,
currentConfig.pwm_dutycycle/100.0);

  return currentConfig.pwm_period_us;
}

uint8_t Device_PWMDutyCycle(uint8_t val){
  if(val > 100){
    val = 100;
  }
  currentConfig.pwm_dutycycle = val;
  PWM_Config(currentConfig.pwm_period_us,
currentConfig.pwm_dutycycle/100.0);

  return currentConfig.pwm_dutycycle;
}

bool Device_PWMState(bool val){
  if(val){
    PWM_Start();
    currentConfig.pwm_on = true;}
  else{
    PWM_Stop();
    currentConfig.pwm_on = false;
  }
  return  currentConfig.pwm_on;
}

uint8_t Device_PGAGain(uint8_t val){
  switch(val){
    default:
    case 1:
       currentConfig.pga_gainSel = 0;
       PGA_SetGain(currentConfig.pga_gainSel);
       return 1;
    case 2:
       currentConfig.pga_gainSel = 1;
       PGA_SetGain(currentConfig.pga_gainSel);
       return 2;
    case 4:
       currentConfig.pga_gainSel = 2;
```

```
        PGA_SetGain(currentConfig.pga_gainSel);
        return 4;
    case 5:
        currentConfig.pga_gainSel = 3;
        PGA_SetGain(currentConfig.pga_gainSel);
        return 5;
    case 8:
        currentConfig.pga_gainSel = 4;
        PGA_SetGain(currentConfig.pga_gainSel);
        return 8;
    case 10:
        currentConfig.pga_gainSel = 5;
        PGA_SetGain(currentConfig.pga_gainSel);
        return 10;
    case 16:
        currentConfig.pga_gainSel = 6;
        PGA_SetGain(currentConfig.pga_gainSel);
        return 16;
    case 32:
        currentConfig.pga_gainSel = 7;
        PGA_SetGain(currentConfig.pga_gainSel);
        return 32;
  }
}

uint32_t Device_PGAVref(uint32_t val){
   if(val > 3300)
     val = 3300;
   currentConfig.pga_vref_mV = val;
   Vref_Set(currentConfig.pga_vref_mV);
   return currentConfig.pga_vref_mV;
}

void Device_PGAConfig(uint32_t gain){
  uint32_t optVref[] = {1700, 1700, 2233, 2300, 2386, 2411, 2447,
2474};
  Device_PGAGain(gain);
  Device_PGAVref(optVref[currentConfig.pga_gainSel]);
}

bool Device_ExternalADC(bool val){
  if(val){
    ExternalADC_Enable();
    currentConfig.externalADC = true;
  }else{
    ExternalADC_Disable();
    currentConfig.externalADC = false;
  }
  return currentConfig.externalADC;
}

uint32_t Device_SPIFrequency(uint32_t val){
  if(val > 50000000){
    val = 50000000;
  }else if( val < 100000){
    val = 100000;
```

```
  }

  currentConfig.SPIfrequency = val;
  SetSPIFreq(currentConfig.SPIfrequency);

  return currentConfig.SPIfrequency;
}

bool Device_ContinuousMode(bool val){
  currentConfig.continuousModeOn = val;

  return currentConfig.continuousModeOn;
}

uint32_t Device_MeasureInterval(uint32_t val_ms){
  if(val_ms > 300000){
    val_ms = 300000;
  }else if(val_ms < 10){
    val_ms = 10;
  }
  currentConfig.acqInterval_ms = val_ms;

  return currentConfig.acqInterval_ms;
}

uint32_t Device_MeasureTimeout(uint32_t val_ms){
  currentConfig.acqTimeout_ms = val_ms;

  return currentConfig.acqTimeout_ms;
}

bool Device_AnalogOutput(bool val){
  currentConfig.analogOutOn = val;

  return currentConfig.analogOutOn;
}

uint32_t Device_AnalogOutFrequency(uint32_t val_hz)
{
  if(val_hz > 100000){
    val_hz = 100000;
  }else if (val_hz < 1000){
    val_hz = 1000;
  }

  currentConfig.analogOutFreq = val_hz;
  AnalogOut_SetFreq(currentConfig.analogOutFreq);

  return currentConfig.analogOutFreq;
}

bool Device_OutputModeString(bool val){
  currentConfig.strNotBytes = val;

  return currentConfig.strNotBytes;
}
```

Anexo B.12 – Funções para obter parâmetros

```
uint32_t Device_GetClock()
{
  return currentConfig.clockFrequency;
}

uint32_t Device_GetIntgTime()
{
  return currentConfig.integrationTime_us;
}

bool Device_GetCfSel()
{
  return currentConfig.cselPin;
}

uint8_t Device_GetChannels()
{
  return currentConfig.nChannels;
}

uint8_t Device_GetChannelUpperLimit(uint8_t ch)
{
  return currentConfig.channelLimits[ch*2+1];
}

uint8_t Device_GetChannelLowerLimit(uint8_t ch)
{
   return currentConfig.channelLimits[ch*2];
}

bool Device_GetEqualChannels()
{
  return currentConfig.equalChannels;
}

uint32_t Device_GetPWMPeriod()
{
  return currentConfig.pwm_period_us;
  }

uint8_t Device_GetPWMDutyCycle()
{
  return currentConfig.pwm_dutycycle;
}

bool Device_GetPWMState()
{
  return currentConfig.pwm_on;
}

uint8_t Device_GetPGAGain(){
  uint8_t gains[] = {1,2,4,5,8,10,16,32};
  return gains[currentConfig.pga_gainSel];
```

```
}

uint32_t Device_GetPGAVref()
{
  return currentConfig.pga_vref_mV;
}

bool Device_GetExternalADC()
{
  return currentConfig.externalADC;
}

uint32_t Device_GetSPIFrequency()
{
  return currentConfig.SPIfrequency;
}

bool Device_GetContinuousMode()
{
  return currentConfig.continuousModeOn;
}

uint32_t Device_GetMeasureInterval()
{
  return currentConfig.acqInterval_ms;
}

uint32_t Device_GetMeasureTimeout()
{
  return currentConfig.acqTimeout_ms;
}

bool Device_GetAnalogOutput()
{
  return currentConfig.analogOutOn;
}

uint32_t Device_GetAnalogOutFrequency()
{
  return currentConfig.analogOutFreq;
}

bool Device_GetOutputModeString()
{
  return currentConfig.strNotBytes;
}
```

Anexo B.13 – Funções de aquisição, processamento e transmissão

```
static uint16_t acquiredData[128];
static float fProcessedData[128];
static uint16_t processedData[128];
static uint8_t processedDataDAC[128];
static uint32_t time_i, time_f;

int Device_Acquisition(){
  ADC_Prepare();
  Reset_Pulse_us(currentConfig.integrationTime_us);
  uint32_t time_i = millis();
  if(!currentConfig.pwm_on){ // If PWM active remains active
    PWM_Start();
    while(millis()-time_i  <  (currentConfig.integrationTime_us  /
1000));
    PWM_Stop();
  }
  int ret = ADC_WaitSequence(currentConfig.acqTimeout_ms);
  switch(ret){
    case 0:
      delay(2);  //  Fix  bug  where  data  was  from  previous
acquisition
      ADC_GetData(acquiredData);
      captureCount++;
      return 0;
    case 1:
    default:
      if(currentConfig.strNotBytes)
        Serial.println("Error");
      else
        Serial.write(0xff); //
      return 1;
    case 2:
      if(currentConfig.strNotBytes)
        Serial.println("Timeout");
      else
        Serial.write(0xfe); //
      return 2;
  }
}

void Device_Process(){
    for(uint8_t ch = 0; ch < currentConfig.nChannels; ++ch){
      uint32_t sum = 0;
      uint8_t start_pixel = currentConfig.channelLimits[2*ch   ];
      uint8_t  end_pixel    = currentConfig.channelLimits[2*ch  +
1];
      for(uint8_t j = start_pixel; j <= end_pixel; ++j){
        sum += acquiredData[j];
      }
      if(currentConfig.externalADC){
        if(currentConfig.analogOutOn){
          processedDataDAC[ch] = (uint8_t) ((sum/(end_pixel + 1 -
start_pixel)) >> 7); // tensao media em 8 bits para DAC
```

```
        }
        processedData[ch] = sum/(end_pixel + 1 - start_pixel); //
tensao mÃ©dia
      }else{
        if(currentConfig.analogOutOn){
          processedDataDAC[ch] = (uint8_t) ((sum/(end_pixel + 1 -
start_pixel)) >> 4); // tensao media em 8 bits para DAC
        }
        processedData[ch] = (sum/(end_pixel + 1 - start_pixel))
<< 3; // media normalizada para 15 bits
      }
      fProcessedData[ch]                                          =
processedData[ch]*ADC_REF_MILIVOLT/(float)EXT_ADC_MAX_VAL;
    }
}

void Device_Transmit(){
  if(currentConfig.strNotBytes)
  {
   for(int i=0; i<currentConfig.nChannels; ++i){
    Serial.println(fProcessedData[i]);
  }
  Serial.println("End");
  }
  else
  {
    Serial.write((uint8_t)currentConfig.nChannels);
    for(int i=0; i<currentConfig.nChannels; ++i)
    {
      Serial.write((uint8_t)(processedData[i]>>8));
      Serial.write((uint8_t)processedData[i]);
    }
  }

  if(currentConfig.analogOutOn){
    AnalogOut_WriteArray(processedDataDAC,
currentConfig.nChannels);
  }
}

void Device_ContAcquisition(){
  static uint32_t acqTime;
  if(currentConfig.continuousModeOn)
  {
    if(millis()-acqTime > (int)(currentConfig.acqInterval_ms) )
    {
      acqTime = millis();
      if(Device_Acquisition() == 0)
      {
        Device_Process();
        Device_Transmit();
      }
    }
  }
}
```

Anexo B.14 – Cabeçalho do módulo SCPI

```
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "myUtils.h"

#define MAX_SUBNODES      20
#define MAX_ARG_SIZE      128

typedef void (*funcType)(char *) ;

typedef struct node_s{
  char *nodeName;
  char *shortName;
  struct node_s *childNodes[MAX_SUBNODES];
  uint8_t nChilds;
  funcType setFunction;
  funcType queryFunction;
}NodeTypedef;

void SCPI_Initialize();
void  SCPI_AddNode(NodeTypedef  *parent,  const  char  *nodeName,
const char *shortName, funcType setFunc, funcType queryFunc);
NodeTypedef *SCPI_GetRootNode();
NodeTypedef *SCPI_GetCommonNode();
NodeTypedef *SCPI_GetCurrentNode();
NodeTypedef   *   GetNodeIn(NodeTypedef   *parent,   const   char
*nodeName);
int SCPI_RunCommand(char *cmd);
```

Anexo B.15 – Corpo do modulo SCPI.c

```
#include "scpi.h"

static NodeTypedef *currentNode;
static NodeTypedef *rootNode;
static NodeTypedef *commonNode;

void UndefinedFunc(char *arg){}

NodeTypedef *SCPI_GetRootNode(){
  return rootNode;
}

NodeTypedef *SCPI_GetCommonNode(){
  return commonNode;
}

NodeTypedef *SCPI_GetCurrentNode(){
  return currentNode;
}

void SCPI_Initialize(){
  /* Common commands */
  commonNode = (NodeTypedef*)malloc(sizeof(NodeTypedef));
  commonNode->nodeName = (char*)malloc(sizeof(char)*7);
  strcpy(commonNode->nodeName, "common");
  commonNode->nodeName;
  strcpy(commonNode->nodeName, "common");
  commonNode->nChilds = 0;
  //commonNode->changePath = true;
  commonNode->setFunction = &UndefinedFunc;
  commonNode->queryFunction = &UndefinedFunc;

  /* Root node init */
  rootNode = (NodeTypedef*)malloc(sizeof(NodeTypedef));

  rootNode->nodeName = (char*)malloc(sizeof(char)*5);
  strcpy(rootNode->nodeName, "root");

  rootNode->nodeName;
  strcpy(rootNode->nodeName, "root");

  rootNode->nChilds = 0;
  rootNode->setFunction = &UndefinedFunc;
  rootNode->queryFunction = &UndefinedFunc;

  currentNode = rootNode;
}

void SCPI_AddNode(NodeTypedef *parent, const char *nodeName,
const char *shortName, funcType setfunc, funcType queryfunc){

  NodeTypedef                    *newNode                    =
(NodeTypedef*)malloc(sizeof(NodeTypedef));
```

```
  newNode->nodeName                                                    =
(char*)malloc(sizeof(char)*(strlen(nodeName)+1));
  strcpy(newNode->nodeName, nodeName);

  if(shortName != NULL){
    newNode->shortName                                                 =
(char*)malloc(sizeof(char)*(strlen(shortName)+1));
    strcpy(newNode->shortName, shortName);
  }else{
    newNode->shortName = (char*)malloc(sizeof(char)*1);
    *(newNode->shortName) = '\0';
  }
  newNode->nChilds = 0;
  newNode->setFunction = setfunc;
  newNode->queryFunction = queryfunc;

  if(parent != NULL){
    parent->childNodes[parent->nChilds++] = newNode;
  }
}

NodeTypedef   *   GetNodeIn(NodeTypedef   *parent,   const   char
*nodeName){
  if(parent == NULL){
    parent = rootNode;
  }
  for(int i = 0; i < parent->nChilds; ++i){
    if(WordIs(parent->childNodes[i]->nodeName,    nodeName)    ||
WordIs(parent->childNodes[i]->shortName, nodeName)){
      return parent->childNodes[i];
    }
  }
  return NULL;
}

char * DoFunctionAndMove(NodeTypedef *node, char *cmd)
{
  for(int i = 0; i < node->nChilds; ++i){
    if(WordIs(node->childNodes[i]->nodeName, cmd) || WordIs(node-
>childNodes[i]->shortName, cmd)){
      cmd += WordSizeIs(cmd); // AvanÃ§a para depois da keyword
      cmd = FindNextNonWhitespace(cmd);
      // Verifica se o comando é de set ou query
      bool flagQuery = false;
      if(cmd[0] == '?'){
        flagQuery = true;
        cmd += 1;
      }
      // Obtem argumentos
      char arg[MAX_ARG_SIZE] = {'\0'};
      char *lastChr = FindLastCharOfArgument(cmd);
      if(lastChr != NULL){
        uint32_t arg_size = lastChr-cmd +1;
        strncpy(arg, cmd, arg_size );
        arg[arg_size] = '\0';
```

```
      }
      if(flagQuery){
        node->childNodes[i]->queryFunction(arg);      //  executa
funcao de query
      }else{
        node->childNodes[i]->setFunction(arg);        //  executa
funcao de set
      }
      if(node != commonNode){
        if(currentNode->childNodes[i]->nChilds > 0){  // Verifica
se deve actualizar o currentNode
          currentNode = currentNode->childNodes[i];
        }
        cmd = FindNextPunctuation(cmd);
        if(cmd[0] == ':'){ // Evita leitura de ':' na função
SCPI_RunCommand, reiniciando o currentNode
          cmd += 1;
        }
      }else{
        cmd = strchr(cmd, ';'); // Node comum apenas tem um nivel.
Procura proximo comando
      }
      return cmd;
    }
  }
  cmd = strchr(cmd, ';'); // Não encontrou palavra, procura o
proximo comando
  return cmd;
}


int SCPI_RunCommand(char *cmd){
  if(cmd == NULL || cmd[0] == '\0')
    return 0;
  cmd = FindNextNonWhitespace(cmd);
  char * aux;

  switch(cmd[0]){
    case '*':
      cmd = FindNextWord(cmd);
      aux = FindNextPunctuation(cmd);
      if(aux != NULL && cmd != NULL && aux < cmd){ // Caso encontre
pontuacao antes da proxima palavra
        return SCPI_RunCommand(aux);
      }
      cmd = DoFunctionAndMove(commonNode, cmd);
      return SCPI_RunCommand(cmd);
    case ';':
      return SCPI_RunCommand(cmd+1);
    case ':':
      currentNode = rootNode;
      cmd = FindNextWord(cmd);
      aux = FindNextPunctuation(cmd);
      if(aux != NULL && cmd != NULL && aux < cmd){ // Caso encontre
pontuacao antes da proxima palavra
        cmd = aux;
```

```
      }else if(cmd == NULL){ // Caso a proxima palavra nao exista
        return 0;
      }
    default:
      cmd = DoFunctionAndMove(currentNode, cmd);
      return SCPI_RunCommand(cmd);
  }
}
```

Anexo B.16 – Cabeçalho do módulo DeviceSCPI

```
#ifndef DEVICE_SCPI_H
#define DEVICE_SCPI_H

#include "Arduino.h"
#include "myDrivers.h"
#include "DeviceFunctions.h"

#define DEBUG__

extern "C"
{
  #include "myUtils.h"
  #include "scpi.h"
}

#define GBL_IDN "IDN"
#define GBL_RST "RST"
#define GBL_ECHO "ECHO"

#ifdef DEBUG__
#define GBL_PRINT_THIS "printThisNode"
#define GBL_PRINT_ALL "printAllNodes"
#define GBL_PRINT_THIS_SHORT "pThis"
#define GBL_PRINT_ALL_SHORT "pAll"
#endif

#define CMD_CONF_SHORT     "CONF"
#define CMD_CONF           "CONFigure"
#define SUB_CMD_DEFAULT    "default"
#define SUB_CMD_CLK        "clock"
#define SUB_CMD_INT_T      "intg"
#define SUB_CMD_CSEL       "csel"
#define SUB_CMD_N_CH       "n_ch"
#define SUB_CMD_CH_LMT     "ch_lmt"
#define SUB_CMD_CH_EQUAL   "ch_equal"
#define SUB_CMD_PWM_PERIOD "PWM_PERIOD"
#define SUB_CMD_PWM_DUTY   "PWM_DUTY"
#define SUB_CMD_PWM_STATE  "PWM_STATE"

#define SUB_CMD_PGA_GAIN   "pga_gain"
#define SUB_CMD_PGA_VREF   "pga_vref"
#define SUB_CMD_PGA_CONF   "pga_conf"
#define SUB_CMD_ADC_EXT    "adc_ext"
#define SUB_CMD_SPI_FREQ   "spi_freq"

#define CMD_MEAS_SHORT     "MEAS"
#define CMD_MEAS           "MEASure"
#define SUB_CMD_SINGLE     "single"
#define SUB_CMD_CONTINUOUS "cont"
#define SUB_CMD_INTERVAL   "interval"
#define SUB_CMD_TIMEOUT    "timeout"
#define SUB_CMD_ANALOG_OUT "anlg_out"
#define SUB_CMD_ANALOG_FREQ "anlg_freq"
```

```
#define SUB_CMD_STRING      "string"
#define SUB_CMD_BYTES       "bytes"

#ifdef DEBUG__
#define CMD_TEST            "TEST"
#define SUB_CMD_RST_PULSE   "resetPulse"
#define SUB_CMD_PWM_PULSE   "PWM_PULSE"
#define SUB_CMD_ANALOG_SET  "anlg_val"
#define SUB_CMD_ADC_CONV    "adc_conv"
#define SUB_CMD_ADC_CAL     "adc_cal"
#define SUB_CMD_I2C_TEST    "i2c_dac"
#endif


void SCPI_Setup();
bool SCPI_IsEchoOn();


#endif
```

Anexo B.17 – Corpo do módulo DeviceSCPI

```
#include "DeviceSCPI.h"

static bool scpi_echo;

void RST_Func(char *arg){
  Serial.println("RESET...");
  HAL_NVIC_SystemReset();
}

void IDN_Func(char *arg){
  Serial.println("Image sensor controller");
}

void EchoEnDis(char *arg){
  scpi_echo = ReadTrueOrFalse(arg);
}

void EchoState(char *arg){
  Serial.println(scpi_echo?"ON":"OFF");
}

void DoNothing(char * arg){}

/* Measure commands (measurement parameters, data acquisition) */

void Meas_IntervalSet(char *arg){
  Device_MeasureInterval(ReadPositiveNumber(arg));
}

void Meas_IntervalQuery(char *arg){
  Serial.println(Device_GetMeasureInterval());
}

void Measure_Single(char * arg){
  if(Device_Acquisition() == 0){ // se != 0 -> erro
    Device_Process();
    Device_Transmit();
  }
}

void Measure_ContinuousSet(char * arg){
  Device_ContinuousMode(ReadTrueOrFalse(arg));
}

void Measure_ContinuousQuery(char *arg){
   Serial.println(Device_GetContinuousMode()?"ON":"OFF");
}

void Meas_TimeoutSet(char *arg){
  Device_MeasureTimeout(ReadPositiveNumber(arg));
}
```

```
void Meas_TimeoutQuery(char *arg){
  Serial.println(Device_GetMeasureTimeout());
}

void Meas_AnalogOutSet(char *arg){
  Device_AnalogOutput(ReadTrueOrFalse(arg));
}

void Meas_AnalogOutQuery(char *arg){
  Serial.println(Device_GetAnalogOutput()?"ON":"OFF");
}

void Meas_AnalogOutFrequencySet(char *arg){
  Device_AnalogOutFrequency(ReadPositiveNumber(arg));
}

void Meas_AnalogOutFrequencyQuery(char *arg){
  Serial.println(Device_GetAnalogOutFrequency());
}

void Meas_SetString(char *arg)
{
  Device_OutputModeString(true);
}

void Meas_QueryString(char *arg)
{
  Serial.println(Device_GetOutputModeString());
}

void Meas_SetBytes(char *arg)
{
   Device_OutputModeString(false);
}

void Meas_QueryBytes(char *arg)
{
  Serial.println(!Device_GetOutputModeString());
}


/*  Configuration  commands  (image  sensor  clock  frequency,
integration time... PWM parameters, signal conditioning, etc...)
*/

void Conf_Default(char * arg){
  Device_SetConfig(&defaultConfig);
}

void Conf_ClockSet(char *arg){
  Device_Clock(ReadPositiveNumber(arg));
}

void Conf_ClockQuery(char *arg){
  Serial.println(Device_GetClock());
}
```

```
void Conf_IntTimeSet(char *arg){
  Device_IntgTime(ReadPositiveNumber(arg));
}

void Conf_IntTimeQuery(char *arg){
  Serial.println(Device_GetIntgTime());
}

void Conf_PWM_PeriodSet(char *arg){
  Device_PWMPeriod(ReadPositiveNumber(arg));
}

void Conf_PWM_DutySet(char *arg){
  Device_PWMDutyCycle(ReadPositiveNumber(arg));
}

void Conf_PWM_PeriodQuery(char *arg){
  Serial.println(Device_GetPWMPeriod());
}

void Conf_PWM_DutyQuery(char *arg){
  Serial.println(Device_GetPWMDutyCycle());
}

void Conf_PWM_State(char *arg){
  Device_PWMState(ReadTrueOrFalse(arg));
}

void Query_PWM_State(char *arg){
  Serial.println(Device_GetPWMState()?"ON":"OFF");
}

void Conf_CSelSet(char *arg){
  Device_CfSel(ReadTrueOrFalse(arg));
}

void Conf_CSelQuery(char *arg){
  Serial.println(Device_GetCfSel()?"HIGH":"LOW");
}

void Conf_NChannelsSet(char *arg){
  Device_Channels(ReadPositiveNumber(arg));
}

void Conf_NChannelsQuery(char *arg){
  Serial.println(Device_GetChannels());
}

void Conf_ChLimitsSet(char *arg){
  uint32_t ch, inf_lmt, up_lmt;

  ch = ReadPositiveNumber(arg) & 0x7f;
  arg = FindNextComma(arg) + 1;
  inf_lmt = ReadPositiveNumber(arg) & 0x7f;
  arg = FindNextComma(arg) + 1;
```

```
  up_lmt = ReadPositiveNumber(arg) & 0x7f;

  Device_ChannelLimits(ch, inf_lmt, up_lmt);

}

void Conf_ChLimitsQuery(char *arg){
  uint32_t ch;
  ch = ReadPositiveNumber(arg) & 0x7f;
  Serial.println(Device_GetChannelLowerLimit(ch));
  Serial.println(Device_GetChannelUpperLimit(ch));
}

void Conf_EqualChannels(char *arg){
  Device_EqualChannels(true);
}

void Conf_EqualChannelsQuery(char *arg)
{
  Serial.println(Device_GetEqualChannels()?"ON":"OFF");
}

void Conf_PGA_GainSet(char * arg){
  Device_PGAGain(ReadPositiveNumber(arg));
}

void Conf_PGA_GainQuery(char *arg){
  Serial.println(Device_GetPGAGain());
}

void Conf_VrefSet(char *arg){
   Device_PGAVref(ReadPositiveNumber(arg));
}

void Conf_VrefQuery(char *arg){
  Serial.println(Device_GetPGAVref());
}

void Conf_PGA_Configure(char *arg){
  Device_PGAConfig(ReadPositiveNumber(arg));
}

void Conf_ExternalADCSet(char *arg){
  Device_ExternalADC(ReadTrueOrFalse(arg));
}

void Conf_ExternalADCQuery(char *arg){
  Serial.println(Device_GetExternalADC()?"ON":"OFF");
}

void Conf_SPIFreqSet(char *arg){
  Device_SPIFrequency(ReadPositiveNumber(arg));
}

void Conf_SPIQuery(char *arg){
  Serial.println(Device_GetSPIFrequency());
```

```
}


#ifdef DEBUG__
void SCPI_PrintNodes(NodeTypedef *p){
  Serial.println(p->nodeName);
  for(int i = 0; i< p->nChilds; ++i){
    Serial.print("In ");
    Serial.print(p->nodeName);
    Serial.println(":");
    SCPI_PrintNodes(p->childNodes[i]);
  }
}

void PrintAllNodes(char *arg){
  SCPI_PrintNodes(SCPI_GetRootNode());
}

void PrintCurrentNode(char *arg){
  Serial.print("Current node is ");
  Serial.println(SCPI_GetCurrentNode()->nodeName);
}

/* Test commands */

void TestRSTPulse(char *arg){
  uint32_t us = ReadPositiveNumber(arg);
  Reset_Pulse_us(us);
}

void Test_PWMPulse(char *arg){
  uint32_t ms = ReadPositiveNumber(arg);
  if(ms>5000)
    ms = 5000;
  PWM_Start();
  delay(ms);
  PWM_Stop();
}

void Test_AnalogSetLevel(char *arg){
  uint32_t val = ReadPositiveNumber(arg);
  if(val > 3300)
    val = 3300;
  AnalogOut_WriteWord(val*255/3300);
}

void Test_ADC_Conv(char *arg)
{
  Serial.println(ADC_SingleConvertion());
}


void Test_I2C_DAC(char *arg)
{
  Serial.println(Vref_TestDevice()?"passed":"failed");
}
```

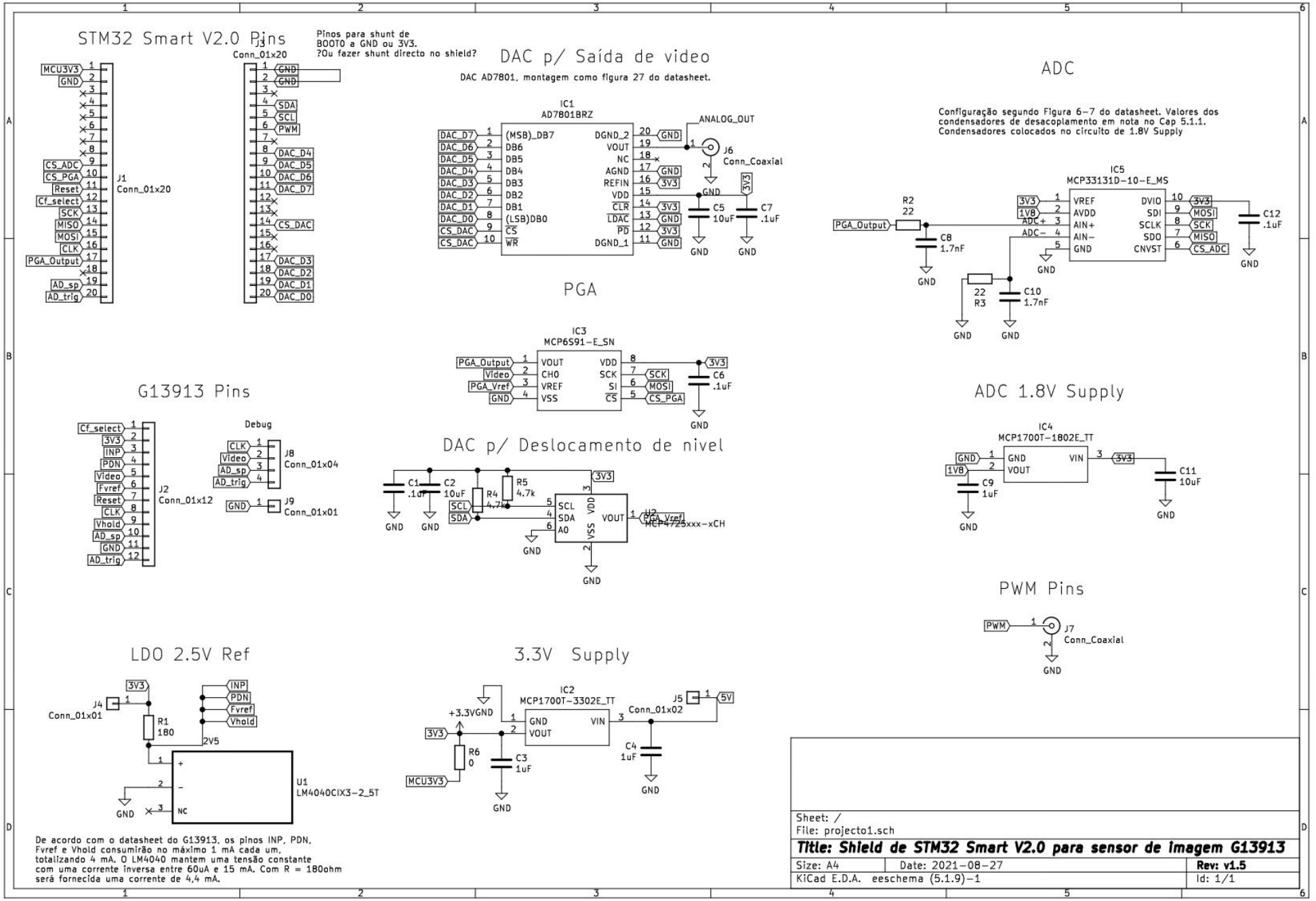
```
void Test_ADC_Cal(char *arg)
{
  ExternalADC_Calibrate();
}

#endif

void SCPI_Setup(){
  SCPI_Initialize();

  SCPI_AddNode(SCPI_GetCommonNode(), GBL_RST,   NULL, RST_Func,
DoNothing);
  SCPI_AddNode(SCPI_GetCommonNode(), GBL_IDN,   NULL, IDN_Func,
IDN_Func);
  SCPI_AddNode(SCPI_GetCommonNode(), GBL_ECHO, NULL,  EchoEnDis,
EchoState);

  NodeTypedef * root = SCPI_GetRootNode();
  SCPI_AddNode(root,   CMD_CONF,   CMD_CONF_SHORT,   DoNothing,
DoNothing);
  SCPI_AddNode(root,   CMD_MEAS,   CMD_MEAS_SHORT,   DoNothing,
DoNothing);


  NodeTypedef *confNode, *measNode;
  confNode = GetNodeIn(root,CMD_CONF);
  SCPI_AddNode(confNode, SUB_CMD_DEFAULT, "def",   Conf_Default,
DoNothing);
  SCPI_AddNode(confNode, SUB_CMD_CLK,      NULL,  Conf_ClockSet,
Conf_ClockQuery);
  SCPI_AddNode(confNode, SUB_CMD_INT_T,   NULL,  Conf_IntTimeSet,
Conf_IntTimeQuery);
  SCPI_AddNode(confNode, SUB_CMD_CSEL,     NULL,  Conf_CSelSet,
Conf_CSelQuery);
  SCPI_AddNode(confNode,    SUB_CMD_N_CH,                  NULL,
Conf_NChannelsSet, Conf_NChannelsQuery);
  SCPI_AddNode(confNode,       SUB_CMD_CH_LMT,             NULL,
Conf_ChLimitsSet,   Conf_ChLimitsQuery);
  SCPI_AddNode(confNode,          SUB_CMD_CH_EQUAL,        NULL,
Conf_EqualChannels,   Conf_EqualChannelsQuery);
  SCPI_AddNode(confNode,        SUB_CMD_PWM_PERIOD,        NULL,
Conf_PWM_PeriodSet,  Conf_PWM_PeriodQuery);
  SCPI_AddNode(confNode,      SUB_CMD_PWM_DUTY,            NULL,
Conf_PWM_DutySet,   Conf_PWM_DutyQuery);
  SCPI_AddNode(confNode,         SUB_CMD_PWM_STATE,        NULL,
Conf_PWM_State,    Query_PWM_State);

  SCPI_AddNode(confNode,      SUB_CMD_PGA_GAIN,            NULL,
Conf_PGA_GainSet,        Conf_PGA_GainQuery);
  SCPI_AddNode(confNode, SUB_CMD_PGA_VREF,  NULL,  Conf_VrefSet,
Conf_VrefQuery);
  SCPI_AddNode(confNode,      SUB_CMD_PGA_CONF,            NULL,
Conf_PGA_Configure,         DoNothing);
  SCPI_AddNode(confNode,       SUB_CMD_ADC_EXT,            NULL,
Conf_ExternalADCSet,     Conf_ExternalADCQuery);
```

```
  SCPI_AddNode(confNode,      SUB_CMD_SPI_FREQ,            NULL,
Conf_SPIFreqSet,         Conf_SPIQuery);

  SCPI_AddNode(confNode,     SUB_CMD_ANALOG_OUT,           NULL,
Meas_AnalogOutSet,  Meas_AnalogOutQuery);
  SCPI_AddNode(confNode,     SUB_CMD_ANALOG_FREQ,          NULL,
Meas_AnalogOutFrequencySet,  Meas_AnalogOutFrequencyQuery);

  measNode = GetNodeIn(root,CMD_MEAS);
  SCPI_AddNode(measNode, SUB_CMD_SINGLE,   NULL,  Measure_Single,
DoNothing);
  SCPI_AddNode(measNode,   SUB_CMD_INTERVAL,              "intv",
Meas_IntervalSet, Meas_IntervalQuery);
  SCPI_AddNode(measNode,    SUB_CMD_CONTINUOUS,           NULL,
Measure_ContinuousSet, Measure_ContinuousQuery);
  SCPI_AddNode(measNode,      SUB_CMD_TIMEOUT,            NULL,
Meas_TimeoutSet,  Meas_TimeoutQuery);

  SCPI_AddNode(measNode, SUB_CMD_STRING,  NULL,  Meas_SetString,
Meas_QueryString);
  SCPI_AddNode(measNode,  SUB_CMD_BYTES,   NULL,   Meas_SetBytes,
Meas_QueryBytes);


#ifdef DEBUG__
  NodeTypedef *testNode;
  SCPI_AddNode(root, CMD_TEST,  NULL,  DoNothing,  DoNothing);
  testNode  = GetNodeIn(root,CMD_TEST);
  SCPI_AddNode(testNode, SUB_CMD_RST_PULSE, NULL,  TestRSTPulse,
DoNothing);
  SCPI_AddNode(testNode, SUB_CMD_PWM_PULSE,  NULL, Test_PWMPulse,
DoNothing);
  SCPI_AddNode(testNode,        SUB_CMD_ANALOG_SET,        NULL,
Test_AnalogSetLevel,    DoNothing);
  SCPI_AddNode(testNode, SUB_CMD_ADC_CONV,  NULL,  Test_ADC_Conv,
DoNothing);
  SCPI_AddNode(testNode, SUB_CMD_ADC_CAL,  NULL,   Test_ADC_Cal,
DoNothing);
  SCPI_AddNode(testNode, SUB_CMD_I2C_TEST,  NULL,  Test_I2C_DAC,
DoNothing);

  SCPI_AddNode(SCPI_GetCommonNode(),              GBL_PRINT_ALL,
GBL_PRINT_ALL_SHORT,  PrintAllNodes,  DoNothing);
  SCPI_AddNode(SCPI_GetCommonNode(),            GBL_PRINT_THIS,
GBL_PRINT_THIS_SHORT,  PrintCurrentNode,  DoNothing);

#endif
}

bool SCPI_IsEchoOn()
{
  return scpi_echo;
}
```

Anexo B.18 – Código do ficheiro principal

```
#include "DeviceSCPI.h"

void setup() {
  Serial.begin(115200);
  Device_InitialSetup();
  SCPI_Setup();
}

const uint32_t serialReadFreq = 20;
static uint32_t serialTime;
static char cmd_buffer[256];

void loop()
{
  if(millis()-serialTime > (int)(1000.0/serialReadFreq) )
  {
    serialTime = millis();
    if(Serial.available() > 0 )
    {
      uint32_t i = 0;
      int ch;
      while((ch = Serial.read()) >=0 && ch != '\n')
      {
         cmd_buffer[i++] = (char)ch;
      }
      cmd_buffer[i]='\0';
      if(SCPI_IsEchoOn())
      {
        Serial.println(cmd_buffer);
      }
      SCPI_RunCommand(cmd_buffer);
    }
  }
  Device_ContAcquisition();

  delay(1);
}
```

Anexo C – Esquemático da PCB



Anexo D – Desenvolvimento da aplicação para PC

1. IDE e Frameworks

A aplicação foi desenvolvida com o *software* Visual Studio 2019, em linguagem C#, recorrendo às *framework* da Microsoft, Windows Forms e .NET Framework. O ambiente de desenvolvimento tem ferramentas que facilitam a implementação de diversas funcionalidades numa aplicação gráfica, construindo janelas através do arraste de controlos a partir de uma caixa de componentes diversos (contentores, botões, caixas de texto, barras de menu, etc…), e acedendo rapidamente às suas propriedades numa barra lateral (Figura 104).

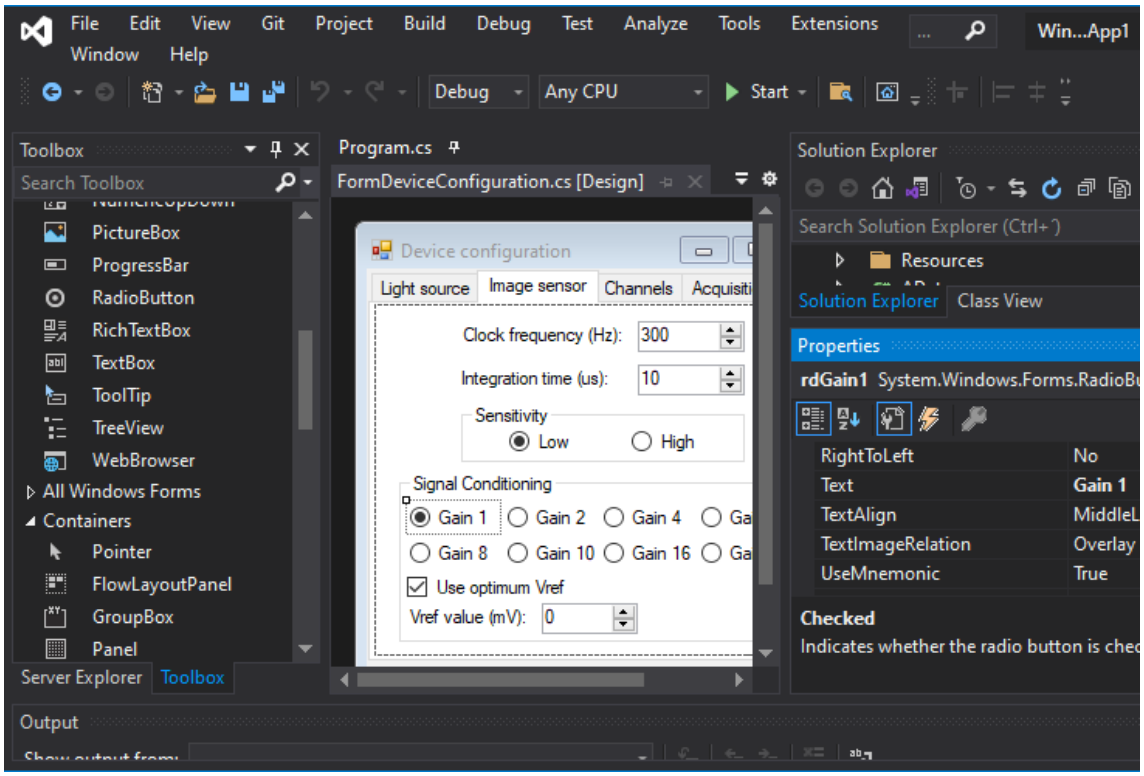


Figura 104 – Captura de ecrã do ambiente de desenvolvimento integrado utilizado para fazer a aplicação. É possível ver: à esquerda – a caixa de componentes; ao centro – uma janela da aplicação a desenvolver; à direita – a barra de propriedades do componente selecionado.

Os componentes da .NET Framework facultam também soluções rápidas para gestão e manuseamento da porta de série, janelas de diálogo e gráficos, entre outros.

2. Arquitetura da solução

As funcionalidades principais da aplicação resumem-se pela configuração do módulo de aquisição, a obtenção dos dados que este transmite e a representação gráfica destes dados.

Com esse intuito, foi desenvolvida a classe estática `ImageDevice`, que após inicialização dispõe de um conjunto de métodos para tratar toda a manipulação do dispositivo, e um conjunto de variáveis-membro onde são armazenados os dados adquiridos e alguns metadados. Suportando essa classe encontram-se as classes:

- `DeviceSettings` – Que descreve a estrutura de configuração do dispositivo;
- `SCPI` – Que resolve o envio de comandos SCPI do dispositivo e a leitura das respostas;
- `AData` – Que descreve os dados adquiridos.

Um diagrama de blocos simplificado ilustra as classes utilizadas na aplicação para controlar o módulo de aquisição (Figura 105).

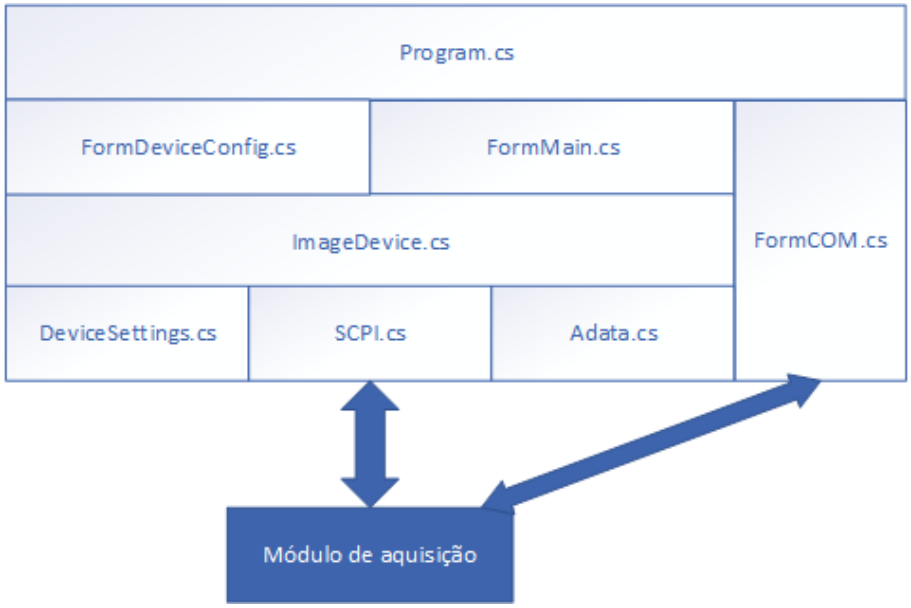


Figura 105 – Diagrama simplificado das classes da aplicação e da comunicação com o dispositivo.

A interface gráfica da aplicação inclui uma janela principal, onde se pode aceder aos menus da aplicação e ver uma representação gráfica dos dados, uma janela de configuração, onde podem ser alteradas as configurações do dispositivo, e uma janela de terminal COM, onde se podem introduzir comandos SCPI manualmente e ler as respostas. Estas janelas são definidas pelas classes `FormMain`, `FormDeviceConfig` e `FormCOM`.

O programa principal, classe `Program`, trata a inicialização necessária para uma aplicação gráfica Windows Forms, assim como da inicialização da porta série, e da janela principal. As janelas de configuração e de terminal COM são instanciadas a partir dos menus da janela principal.

Durante a execução do programa o acesso ao módulo de aquisição é limitado para evitar conflitos na comunicação, utilizando janelas modais (obrigam o utilizador a manter o foco na janela até esta ser fechada) e ativando/desativando botões e menus consoante o dispositivo esteja ou não conectado, ou em modo continuo.

3. Gestão do dispositivo

Segue-se uma descrição das classes desenvolvidas para controlar o dispositivo.

DeviceSettings

A classe `DeviceSettings` contem um conjunto de variáveis que correspondem aos parâmetros de configuração do dispositivo, assim como um conjunto de constantes com valores predefinidos. O seu método construtor inicializa as variáveis-membro com estes valores. Um método auxiliar inicializa os limites dos canais. As variáveis e constantes da classe encontram-se na seguinte tabela.

Tabela 14 – Variáveis-membro da classe DeviceSettings e valores predefinidos

Variável	Tipo	Valor predefinido
clockFrequency	uint	100000
integrationTime_us	uint	10
acquireInterval_ms	uint	1000
acquireTimeout_ms	uint	700
pwmPeriodUs	uint	1000
pga_vref_mV	uint	1700
spiFreq	uint	10000000
analogOutFreq	uint	10000
nChannels	int	128
channelLimits	int [256]	{0,0,1,1,…,127,127}
pwmDutyCycle	int	50
pga_gain	uint	1
cfselPin	bool	false
continuousModeOn	bool	false
equalLimits	bool	false
pwm_on	bool	false
analogOutOn	bool	true
externalADC	bool	false
strNotBytes	bool	false
echo	bool	false
useOptvref	Bool	true

Os valores predefinidos na aplicação podem diferir dos valores predefinidos do microcontrolador, pois este pode assumir inicialmente que será usado através de um terminal de comunicação série.

A classe dispõe também de um método `ToString()` e um método `Parse()` que permitem, respetivamente, a escrita dos parâmetros em formato de *string* e a leitura dos parâmetros a partir de uma *string*. Os métodos foram escritos para facilitar o

armazenamento de configurações em ficheiro de texto, e o seu código encontra-se no Anexo D.1.

SCPI

A classe `SCPI` disponibiliza um conjunto de métodos que escrevem e leem através da comunicação série, introduzindo os comandos SCPI que o módulo de aquisição conhece. Estes permitem afetar e questionar os valores dos parâmetros de configuração do dispositivo, além de efetuar as operações de aquisição nos modos único ou contínuo e ou executar os comandos de identificação (IDN) e reinício (RST).

No Anexo D.2. são exemplificados os métodos correspondentes aos comandos de 'SET' e de 'QUERY' do parâmetro da frequência de relógio do dispositivo. Todos os outros métodos relacionados com parâmetros foram escritos com uma lógica equivalente, de acordo com o tipo da variável. Encontra-se também o código para os métodos relacionados com a aquisição de sinal `SingleAcquisition()` e `GetAcquisitionValues()`.

O método para leitura dos valores pressupõe uma configuração prévia do formato de dados de saída em sequência binária, que é a predefinição imposta pela aplicação ao conectar o dispositivo. É lido o primeiro byte que deverá conter o número de canais de deteção ou um código de erro, mostrando uma janela de diálogo descrevendo o erro neste caso, e prossegue com a leitura da sequência binária, convertendo as palavras digitais em valores de tensão e retornando o *array* de valores.

AData

A classe `AData` contem os valores de uma aquisição de sinal sob as formas de tensão, escala logarítmica (dB) e intensidade relativa, sendo as últimas calculadas a partir dos valores de tensão passados ao construtor. A data e a hora em que o objeto é criado também são armazenados, e as variáveis não podem ser alteradas posteriormente. É mantido um registo estático dos valores máximos e mínimos, para auxiliar nas funções de representação gráfica. A classe foi definida com o código presente no Anexo D.3.

ImageDevice

A classe `ImageDevice` utiliza as três classes anteriores para solucionar a gestão do módulo de aquisição de imagem. No Anexo D.4. encontra-se o bloco de código que mostra a definição das variáveis-membro e método de inicialização da classe. Os protótipos dos restantes métodos também são apresentados.

A informação sobre a configuração atual do dispositivo é mantida em memória estática, numa instância da classe DeviceSettings. A classe estática `SCPI` é utilizada para um

acesso simples aos métodos que executam os comandos SCPI do dispositivo. Os dados adquiridos são armazenados numa lista do tipo `AData`.

O método `ApplySettings()` chama todos os métodos da classe `SCPI` relacionados com a afetação de parâmetros, passando por argumento as variáveis do objeto `deviceSettings`, e atualizando as barras de estado e de progresso da janela principal. O método `ApplyDefaultConfig()` é semelhante, revertendo às configurações predefinidas antes de chamar o método anterior.

O método `CheckSettings()` questiona todos os valores configurados no dispositivo e compara com os valores de configuração atuais no programa, atualizando também as barras de estado e de progresso. Opcionalmente pode mostrar uma janela de diálogo com a informação dos parâmetros errados, ao passar o valor '`true`' como argumento.

O método `SingleAcquisition()` envia um comando de aquisição única e faz a leitura dos dados, guardando o resultado na lista de dados, enquanto o método `GetDataContinuousMode()` apenas obtém e armazena as leituras, devendo ser chamado quando o modo continuo está ativo e existem dados para ler na porta série. O modo continuo é ativado e desativado com os métodos `StartContMode()` e `StopContMode()`, respetivamente.

O método `ClearData()` elimina os dados guardados no programa e reinicia o histórico de valores máximos e mínimos.

Um conjunto alargado de métodos faz a aplicação dos valores configurados para cada parâmetro de configuração de forma individual.

4. Janelas e controlos

Nesta secção são apresentados os controlos que constituem as janelas da aplicação, assim como outros controlos escondidos, explicando de forma sucinta como foram programados e descrevendo alguns exemplos de código.

A Figura 106 ilustra a janela principal (`FormMain`) e os tipos de componentes utilizados.

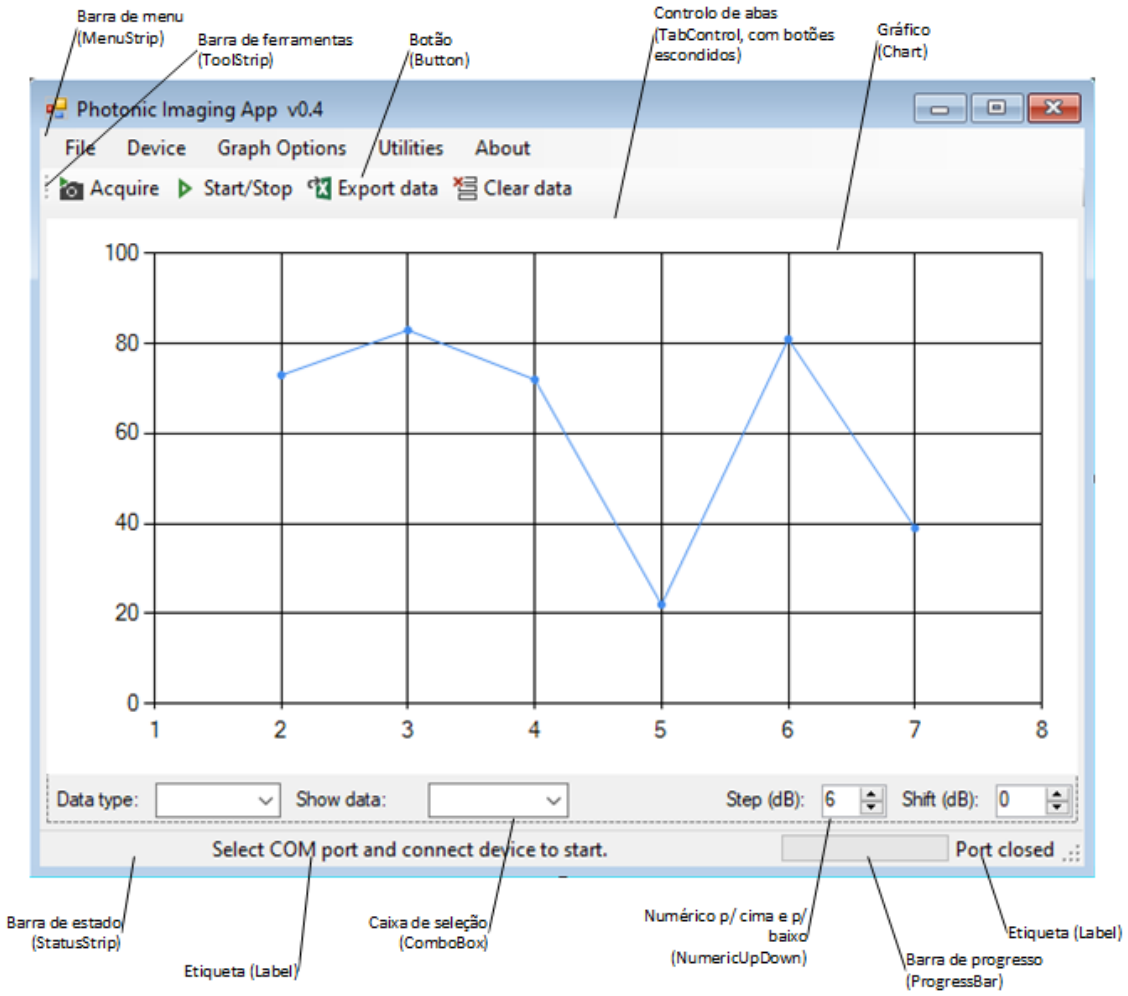


Figura 106 – Janela principal e componentes que a constituem.

No topo da janela encontra-se uma barra de menu, a partir da qual se pode aceder a todas as funcionalidades da aplicação. Por baixo desta está uma barra de ferramentas, contendo botões que funcionam como atalhos para algumas das operações mais frequentes.

No centro da janela encontra-se um controlo de abas com duas páginas, para as duas representações gráficas possíveis. Os botões deste controlo foram escondidos, fazendo-se a seleção da página visível através do menu 'Graph Options' na barra de menu. Na página da representação em linha existe um componente gráfico (Chart) e um conjunto de etiquetas, caixas combo e seletores numéricos que constituem as opções

para a representação gráfica. Na segunda página, não representada na figura, encontra-se um componente de visualização de dados em grelha (DataViewGrid) para uma visualização temporal dos dados. A solução das representações gráficas será descrita na próxima secção.

No fundo da janela encontra-se uma barra de estado, contendo uma etiqueta para mensagens de estado, outra para o estado da porta série, e uma barra de progresso.

Durante a execução, quando o utilizador clica ou seleciona uma opção num determinado controlo, é gerado um evento, ao qual pode ser associado um método. A Figura 107 exemplifica a associação de um método ao evento de clique no botão 'Acquire' da barra de ferramentas, através da caixa de propriedades na interface gráfica.

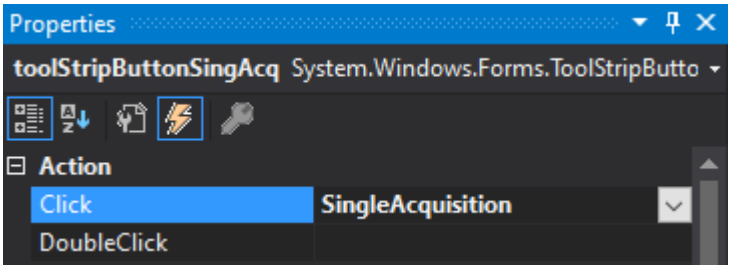


Figura 107 – Exemplo de associação de método a um evento de clique de rato.

Introduzindo o nome e premindo 'ENTER', o método é criado (se não existir já) e o IDE foca o código. Segue-se o código escrito para a função exemplificada. Esta utiliza o método `SingleAcquisition()` da classe `ImageDevice` para fazer uma aquisição de imagem, e em caso de sucesso procede à atualização dos gráficos e da barra de estado.

```
private void SingleAcquisition(object sender, EventArgs e)
{
    if (ImageDevice.SingleAcquisition())
    {
        TimePlotAddMissingRows();
        LinePlotSetup();
        LinePlotUpdate();
        comboSelectData.SelectedIndex                                    =
comboSelectData.FindString("Last");
        toolStripStatusLabel.Text = "Single acquisition done (" +
ImageDevice.lastData.DateTime.ToString() + ").";
    }
}
```

A maior parte das ações que resultam da interação com o utilizador foram programadas desta forma, facilitando o processo de desenvolvimento. Estas incluem: o/a início/paragem do modo de aquisição contínua, a exportação e a eliminação dos dados, as operações de conectar e desconectar o dispositivo, a escolha da visualização dos

dados, a escolha de opções dos gráficos, e a abertura das janelas de configuração e do terminal COM.

Outros componentes importantes podem ser adicionados através da interface gráfica, mas não se encontram na janela. Estes incluem janelas de diálogo para abrir e guardar ficheiros, temporizadores e controladores de portas série. A sua utilização não prescinde da escrita de algum código.

Por exemplo, o bloco de código seguinte contém o método chamado no evento de clique do rato no item 'Load Configuration' do menu 'File', que recorre à janela de diálogo de abertura de ficheiro para carregar um ficheiro de configurações.

```
private   void   loadConfigurationToolStripMenuItem_Click(object
sender, EventArgs e)
{
    DialogResult r = openConfigDialog.ShowDialog();

    if (r == DialogResult.OK) {
        //Read the contents of the file into a stream
        var fileStream = openConfigDialog.OpenFile();
        String fileContent = "";

        using       (StreamReader       reader       =       new
StreamReader(fileStream))
        {
            fileContent = reader.ReadToEnd();
        }
        ImageDevice.deviceSettings.Parse(fileContent);
        ImageDevice.ApplySettings();
        ImageDevice.CheckSettings(true);
        toolStripStatusLabel.Text = "Configurations imported from
'" + openConfigDialog.FileName + "'.";
    }
}
```

O método começa por mostrar a janela de diálogo do componente, utilizando o valor de retorno. O componente apenas retorna 'OK' se o ficheiro selecionado na janela de diálogo for válido, dispensando verificações adicionais. O conteúdo do ficheiro é lido até ao fim para uma *string* através de uma instância da classe `StreamReader`. De seguida a instância da classe `DeviceSettings`, contida na classe `ImageDevice`, utiliza o método `Parse()` para obter os valores de configuração a partir do formato de texto. Por fim as configurações são aplicadas e verificadas, e o texto da barra de estado é atualizado.

A Figura 108 ilustra a janela de configuração e alguns dos seus componentes. Esta janela possui um controlo de abas, desta vez com os botões visíveis, com quatro páginas. Alguns dos controlos foram agrupados em caixas de grupo, como por exemplo

o conjunto de botões rádio onde é feita a seleção do ganho do circuito de acondicionamento. Estes botões permitem a seleção de uma única opção do seu conjunto.

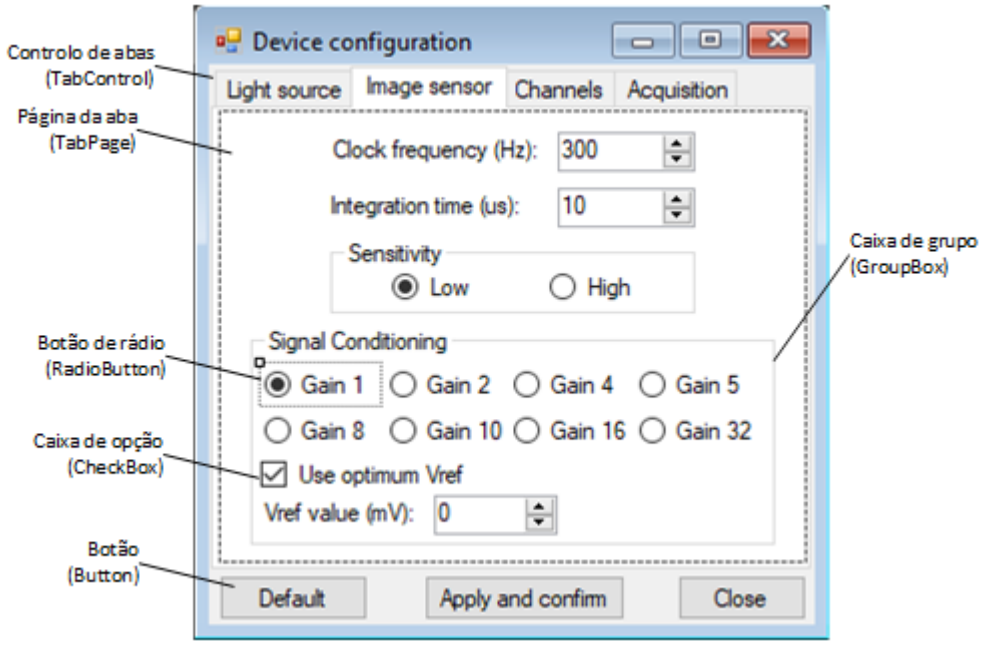


Figura 108 – Janela de configuração e componentes.

A alteração do valor de um dos controlos provoca um evento de alteração de valor. Esse evento pode ser associado à execução de um método, no qual pode ser feita a afetação no membro da estrutura de configurações e a aplicação da nova configuração, como exemplificado no seguinte bloco de código. Este é executado na mudança da seleção do controlo de rádio da sensibilidade do sensor de imagem.

```
private void rdSensHigh_CheckedChanged_1(object sender, EventArgs
e)
{
    if (!updatingControlValues)
    {
        ImageDevice.deviceSettings.cfselPin                    =
rdSensHigh.Checked;
        ImageDevice.ApplyCfSel();
    }
}
```

Desta forma, a alteração de cada parâmetro na janela é seguida da afetação individual desse parâmetro no dispositivo, mantendo a concordância entre as configurações mostradas e as efetivas. Opcionalmente, o botão ‘Apply and confirm’ permite aplicar e verificar todas as configurações.

5. Representação gráfica

Representação em gráfico de linha

Uma das representações mais simples que pode ser feita dos dados de adquiridos num determinado instante é a composição de um gráfico de linha. O componente da classe `Chart` oferece uma solução simples para este tipo de representação (Figura 109).

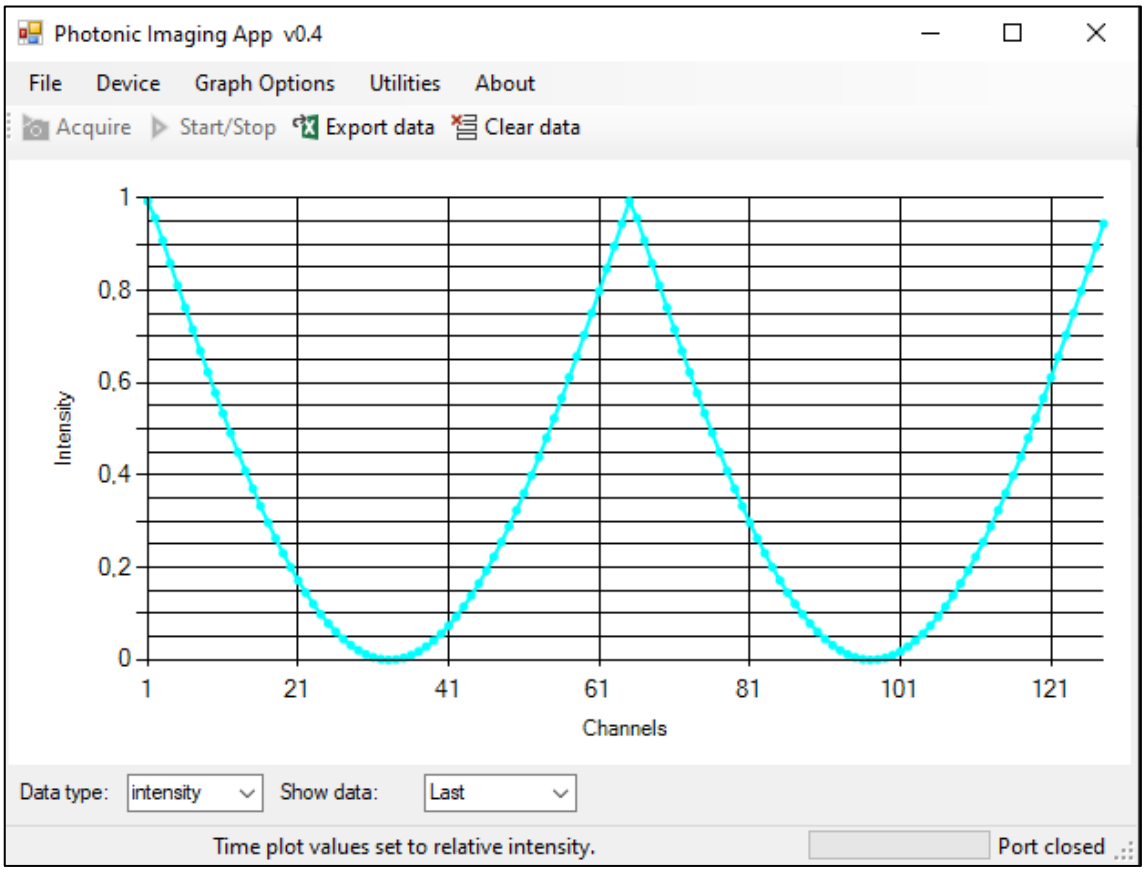


Figura 109 – Janela principal com modo de representação gráfica em linha.

Na janela incluíram-se controlos para selecionar o formato dos dados, entre intensidade relativa, decibéis ou milivolts, e para selecionar o conjunto de dados. A opção predefinida corresponde à última aquisição de sinal, podendo-se selecionar qualquer índice da lista.

Dois métodos na classe `FormMain` foram escritos para trabalhar com o componente `Chart`. O primeiro, `LinePlotSetup()`, cria uma instância de um objeto da classe `ChartArea` com as definições do gráfico consoante o formato dos dados pretendido. O segundo, `LinePlotUpdate()`, cria uma instância da classe `Series` e preenche-a com pontos (objetos da classe `DataPoint`) a partir do índice selecionado da lista de dados.

A atualização do gráfico no modo de aquisição contínua é feita recorrendo à classe *Delegate*, criando um *handler* para o método que atualiza o gráfico. Utilizando o *handler*

é possível atualizar o gráfico a partir do método que faz as leituras da comunicação série, que corre numa *thread* diferente daquela que faz a gestão da interface gráfica.

Representação temporal

A solução para a representação dos dados ao longo do tempo recorreu ao componente da classe `DataViewGrid`, que oferece uma vista em grelha para um conjunto de dados. Uma instância deste componente pode ser preenchida com linhas e colunas, podendo cada coluna corresponder ao número de um canal de deteção e cada linha a uma aquisição de sinal efetuada. A Figura 110 ilustra a janela com esta forma de representação gráfica.

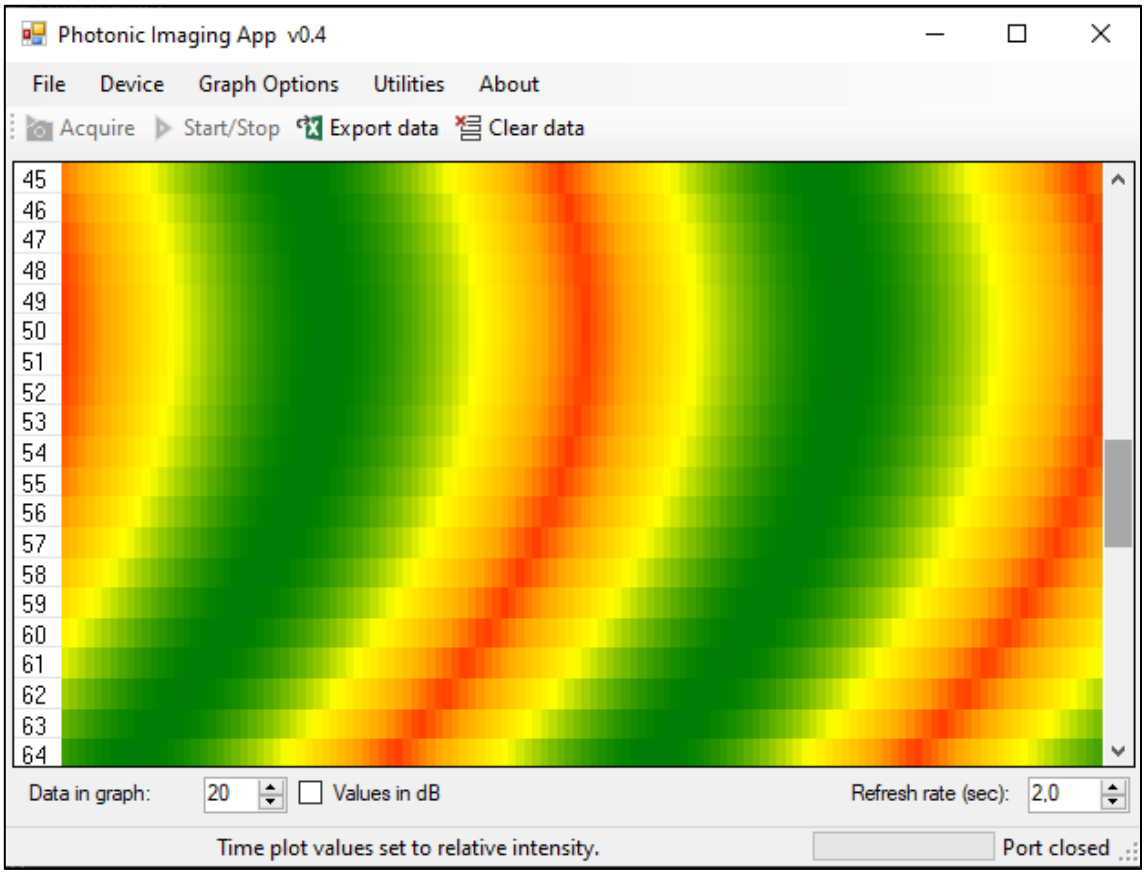


Figura 110 - Janela principal com modo de representação gráfica em tabela com código de cores.

A representação temporal terá que ser feita a três dimensões: número do canal de deteção, momento da aquisição e valor apresentado no canal. Como o componente é essencialmente uma tabela, com linhas e colunas, a terceira dimensão é representada através da codificação de cor de uma célula.

A dimensão horizontal das células é ajustada de forma automática para preencher toda a largura do componente, enquanto a dimensão vertical é ajustada por forma a mostrar um determinado número de aquisições. O componente disponibiliza uma barra de deslocamento vertical para mover as linhas visualizadas.

No entanto, à medida que as linhas são adicionadas ao componente, o seu processamento vai-se tornando mais pesado, especialmente utilizando um maior número de canais de deteção. Por esta razão, quando o modo de aquisição contínua se encontra ativo a atualização dos dados mostrados no componente é a um ritmo independente do intervalo de aquisição, pois em intervalos mais curtos o programa fica sobrecarregado facilmente.

Neste caso a temporização para a atualização da representação gráfica é feita com um temporizador, cujo período pode ser ajustado através do componente numérico no canto inferior direito da janela.

Tal como para a representação gráfica anterior, dois métodos foram criados na classe `FormMain` para gerir o gráfico temporal: `TimePlotSetup()` e `TimePlotAddMissingRows()`. O primeiro define os parâmetros de configuração da tabela e elimina quaisquer linhas ou colunas. Este chama também um método auxiliar que cria um *array* com os códigos de cores. O valor de uma determinada célula pode ser convertido para um tipo inteiro e utilizado para aceder a esse *array*, obtendo a cor correspondente. O segundo método adiciona as linhas à tabela, verificando a diferença entre o número de linhas da tabela e o tamanho da lista de dados.

Os métodos utilizados para a gestão da representação gráfica dos dados encontram-se no Anexo D.5.

Anexo D.1 – Métodos da classe DeviceSettings

```
override public String ToString()
{
    StringBuilder sb = new StringBuilder();

    sb.AppendLine("echo: " + this.echo.ToString());
    sb.AppendLine("clock: " + this.clockFrequency.ToString());
(…)
    sb.AppendLine("SPI freq: " + this.spiFreq.ToString());

    return sb.ToString();
}


public void Parse(String str)
{
    String[] lines = str.Split('\n');

    foreach (var line in lines)
    {
        if (line.StartsWith("echo:"))
        {
            if (!Boolean.TryParse(line.Substring(line.IndexOf(':') + 1), out
this.echo))
            {
                this.echo = defaultEcho;
            }
        }
        else if (line.StartsWith("clock:"))
        {
            if (!UInt32.TryParse(line.Substring(line.IndexOf(':') + 1), out
this.clockFrequency))
            {
                this.clockFrequency = defaultClkFreq;
            }
        }
(…)
        else if (line.StartsWith("SPI freq:"))
        {
            if (!UInt32.TryParse(line.Substring(line.IndexOf(':') + 1), out
this.spiFreq))
            {
                this.spiFreq = defaultSPIfreq;
            }
        }
    }
}
```

Anexo D.2 – Exemplos de métodos da classe SCPI

```
public static void SetClockFrequency(uint clk)
{
    if (serialPort.IsOpen)
    {
        serialPort.WriteLine(":conf:clock " + clk);
    }
}

public static uint QueryClockFrequency()
{
    uint clk = 0;
    if (serialPort.IsOpen)
    {
        serialPort.WriteLine(":conf:clock?");
        try
        {
            string line = serialPort.ReadLine();
            clk = uint.Parse(line);
        }
        catch(Exception e)
        {
            MessageBox.Show(e.ToString());
        }
    }
    return clk;
}

public static double[] SingleAcquisition()
{
    if (!serialPort.IsOpen)
        return null;
    serialPort.WriteLine(":meas:single");
    return GetAcquisitionValues();
}

public static double[] GetAcquisitionValues()
{
    if (!serialPort.IsOpen)
    {
        return null;
    }

    double[] resultado = new double[0];

    Int32 i = 0;
    try
    {
        int n = serialPort.ReadByte();
        if (n == 0xff)
        {
            MessageBox.Show("An unknown error ocurred during signal
aquisition.");
            return null;
        }
        else if (n == 0xfe)
        {
            MessageBox.Show("Timeout error ocurred during signal acquisition.
Timeout value must be adjusted to integration time and clock frequency.");
            return null;
```

```
        }
        resultado = new double[n];
        while (i < n)
        {
            int high = serialPort.ReadByte();
            int low = serialPort.ReadByte();
            Int32 val = Convert.ToInt32((high << 8) + low);
            resultado[i++] = Convert.ToDouble(Math.Min(val, 32768)) / 32767.0 *
3300;
        }
    }
    catch (Exception e)
    {
        MessageBox.Show(e.ToString());
        return null;
    };

    return resultado;
}
```

Anexo D.3 – Classe AData

```
public class AData
{
    private readonly DateTime dateTime;
    private readonly double[] millivolts;
    private readonly double[] db;
    private readonly double[] intensity;
    public static double maxValuedB = -90;
    public static double minValuedB = 0;

    public AData(double[] val, DateTime dt)
    {
        dateTime = dt;

        if (val.Length == 0)
        {
            millivolts = new double[0];
            db = new double[0];
            intensity = new double[0];
            return;
        }

        millivolts = val;
        db = new double[millivolts.Length];
        intensity = new double[millivolts.Length];

        for (int i = 0; i < millivolts.Length; ++i)
        {
            if (ImageDevice.deviceSettings.pga_gain == 1)
            {
                intensity[i] = 1 - Math.Max(Math.Min((millivolts[i] - 300) /
2200, 1), 0);
            }
            else
            {
                intensity[i] = Math.Max(1 - millivolts[i] / 3215, 0); //3300;
recalibrado para tensão máxima medida no dispositivo, ~3215mV
            }
            db[i] = 20 * Math.Log10((Math.Max(intensity[i], 3.05e-5)));
            if (db[i] > maxValuedB)
            {
                maxValuedB = db[i];
            }
            else if (db[i] < minValuedB)
            {
                minValuedB = db[i];
            }
        }
    }

    public AData(double[] val) : this (val, DateTime.Now)
    {
    }
(…)
}
```

Anexo D.4 – Resumo da classe ImageDevice

```
public static class ImageDevice
{
    public static DeviceSettings deviceSettings;
    public static List<AData> acquisitionData;

    public static void Init(SerialPort sp)
    {
        deviceSettings = new DeviceSettings();
        SCPI.Init(sp);
        acquisitionData = new List<AData>();
        AData.ResetMaxMin();
    }(…)
}
```

Restantes métodos da classe:

```
public static bool SingleAcquisition();
public static void StartContMode() ;
public static void StopContMode();
public static bool GetDataContinuousMode();
public static void ClearData();
public static void ApplySettings() ;
public static bool CheckSettings(bool printMessage);
public static void Reset();
public static string IDN();
public static void ApplyDefaultConfig();
public static void ApplyClockFrequency();
public static void ApplyIntegrationTime();
public static void ApplyCfSel();
public static void ApplyChannelsNumber();
public static void ApplyAllChannelLimits();
public static void ApplyChannelLimits(int channel);
public static void ApplyEqualChannelLimits();
public static void ApplyPWMDutyCycle();
public static void ApplyPWMPeriod();
public static void ApplyPWMState();
public static void ApplyAcquireInterval();
public static void ApplyAcquireTimeout();
public static void ApplyPGAGain(bool optimumDC);
public static void ApplyPGAVref();
public static void ApplyExternalADCOn();
public static void ApplySPIFrequency();
public static void ApplyAnalogOutOn() ;
public static void ApplyAnalogOutFreq();
public static void ReadSettings();
```

Anexo D.5 – Métodos para representação gráfica dos dados

```
private void LinePlotSetup()
{
    ChartArea newChartArea = new ChartArea();

    chart1.Series.Clear();

    newChartArea.AxisY.MajorGrid.Enabled = true;
    newChartArea.AxisY.MajorTickMark.Enabled = true;
    newChartArea.AxisY.MinorGrid.Enabled = true;
    newChartArea.AxisY.MinorTickMark.Enabled = true;
    newChartArea.AxisX.Minimum = 1;
    newChartArea.AxisX.Maximum = ImageDevice.deviceSettings.nChannels;
    newChartArea.AxisX.Title = "Channels";

    if (datatype.CompareTo("dB") == 0)
    {
        newChartArea.AxisY.Minimum = -5 * stepDB + shift;
        newChartArea.AxisY.Maximum = 0 + shift;
        newChartArea.AxisY.Title = "Logarithmic";
        newChartArea.AxisY.IsReversed = false;
        newChartArea.AxisY.MajorGrid.Interval = stepDB * 2;
        newChartArea.AxisY.MajorTickMark.Interval = stepDB * 2;
        newChartArea.AxisY.MinorGrid.Interval = stepDB;
        newChartArea.AxisY.MinorTickMark.Interval = stepDB;
        numStepdB.Visible = true;
        numShift.Visible = true;
        labelShift.Visible = true;
        labelStep.Visible = true;

    }
    else if (datatype.CompareTo("mV") == 0)
    {
        newChartArea.AxisY.Minimum = 0;
        newChartArea.AxisY.Maximum = 3300;
        newChartArea.AxisY.Title = "mV";
        newChartArea.AxisY.IsReversed = false;
        newChartArea.AxisY.MajorGrid.Interval = 500;
        newChartArea.AxisY.MajorTickMark.Interval = 500;
        newChartArea.AxisY.MinorGrid.Interval = 500;
        newChartArea.AxisY.MinorTickMark.Interval = 100;
        numStepdB.Visible = false;
        numShift.Visible = false;
        labelShift.Visible = false;
        labelStep.Visible = false;
    }
    else if (datatype.CompareTo("intensity") == 0)
    {
        newChartArea.AxisY.Minimum = 0;
        newChartArea.AxisY.Maximum = 1;
        newChartArea.AxisY.Title = "Intensity";
        newChartArea.AxisY.IsReversed = false;
        newChartArea.AxisY.MajorGrid.Interval = 0.1;
        newChartArea.AxisY.MajorTickMark.Interval = 0.1;
        newChartArea.AxisY.MinorGrid.Interval = 0.05;
        newChartArea.AxisY.MinorTickMark.Interval = 0.1;
        numStepdB.Visible = false;
        numShift.Visible = false;
        labelShift.Visible = false;
        labelStep.Visible = false;
    }
```

```
    chart1.ChartAreas.Clear();
    chart1.ChartAreas.Add(newChartArea);

    LinePlotUpdate();

}


private void LinePlotUpdate()
{

    Program.updatingChart = true;

    Series series = new Series();
    series.IsXValueIndexed = true;
    series.ChartType = SeriesChartType.Line;
    series.XValueMember = "Channel";
    series.YValueMembers = datatype;
    series.BorderWidth = 2;
    series.MarkerStyle = MarkerStyle.Circle;
    series.MarkerSize = 5;
    series.Color = Color.Cyan;

    AData data = new AData(new double[0]);

    if (selectedData.CompareTo("Last") == 0)
    {
        if (ImageDevice.acquisitionData.Count > 0)
        {
            data = ImageDevice.acquisitionData.Last();
        }

    }
    else
    {
        Int32 r;
        if (Int32.TryParse(selectedData, out r))
        {
            data = ImageDevice.acquisitionData[r];
        }
    }

    if (datatype == "mV")
    {
        for (int i = 0; i < data.Length; ++i)
        {
            DataPoint dp = new DataPoint(i + 1, data.Millivolts[i]);
            series.Points.Add(dp);
        }
    }
    else if (datatype == "dB")
    {
        for (int i = 0; i < data.Length; ++i)
        {
            DataPoint dp = new DataPoint(i + 1, data.ValuesdB[i]);
            series.Points.Add(dp);
        }
    }
    else if (datatype == "intensity")
    {
        for (int i = 0; i < data.Length; ++i)
```

```
        {
            DataPoint dp = new DataPoint(i + 1, data.Intensity[i]);
            series.Points.Add(dp);
        }
    }

    chart1.Series.Clear();
    chart1.Series.Add(series);
    chart1.Update();

    Program.updatingChart = false;
}

static List<Color> colors;
static List<Color> colors_dB;
List<Color> interpolateColors(List<Color> stopColors, int count)
{
    SortedDictionary<float, Color> gradient = new SortedDictionary<float,
Color>();
    for (int i = 0; i < stopColors.Count; i++)
        gradient.Add(1f * i / (stopColors.Count - 1), stopColors[i]);
    List<Color> ColorList = new List<Color>();

    using (Bitmap bmp = new Bitmap(count, 1))
    using (Graphics G = Graphics.FromImage(bmp))
    {
        Rectangle bmpCRect = new Rectangle(Point.Empty, bmp.Size);
        LinearGradientBrush br = new LinearGradientBrush
                                 (bmpCRect, Color.Empty, Color.Empty, 0, false);
        ColorBlend cb = new ColorBlend();
        cb.Positions = new float[gradient.Count];
        for (int i = 0; i < gradient.Count; i++)
            cb.Positions[i] = gradient.ElementAt(i).Key;
        cb.Colors = gradient.Values.ToArray();
        br.InterpolationColors = cb;
        G.FillRectangle(br, bmpCRect);
        for (int i = 0; i < count; i++) ColorList.Add(bmp.GetPixel(i, 0));
        br.Dispose();
    }
    return ColorList;
}

private void TimePlotSetup()
{
    dataGridView1.RowHeadersVisible = true;
    dataGridView1.RowHeadersWidth = 26;
    dataGridView1.ColumnHeadersVisible = false;
    dataGridView1.AllowUserToAddRows = false;
    dataGridView1.AllowUserToOrderColumns = false;
    dataGridView1.AllowUserToResizeRows = false;
    dataGridView1.AllowUserToResizeColumns = false;
    dataGridView1.AllowUserToDeleteRows = false;
    dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
    dataGridView1.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.None;
    dataGridView1.CellBorderStyle = DataGridViewCellBorderStyle.None;

    List<Color> baseColors = new List<Color>();  // create a color list
    baseColors.Add(Color.Red);
    baseColors.Add(Color.Orange);
    baseColors.Add(Color.Yellow);
    baseColors.Add(Color.Green);
    baseColors.Add(Color.Blue);
```

```
    colors = interpolateColors(baseColors, 3301);
    colors_dB = interpolateColors(baseColors, Convert.ToUInt16((numMax.Value -
numMin.Value + 1)*10));

    dataGridView1.Columns.Clear();
    dataGridView1.Rows.Clear();

    for (int c = 0; c < ImageDevice.deviceSettings.nChannels; ++c)
    {
        dataGridView1.Columns.Add("", "");
        dataGridView1.Columns[c].MinimumWidth = 2;
    }
    TimePlotAddMissingRows();
    TimePlotResizeRows();
}

private void TimePlotAddMissingRows()
{
    int data_i;
    if ( (ImageDevice.acquisitionData.Count - dataGridView1.Rows.Count) > 1000)
    {
        MessageBox.Show("Warning: high volume of data being processed to be shown
on time plot.");
    }
    while (dataGridView1.Rows.Count < ImageDevice.acquisitionData.Count)
    {

        data_i = dataGridView1.Rows.Count;
        dataGridView1.Rows.Add("", "");
        dataGridView1.Rows[data_i].HeaderCell.Value = String.Format("{0}",
data_i);
        dataGridView1.Rows[data_i].HeaderCell.ToolTipText = "Data index";

        for (int c = 0; c < ImageDevice.deviceSettings.nChannels; c++)
        {
            if (c >= ImageDevice.acquisitionData[data_i].Length)
            {
                dataGridView1[c, data_i].Style.BackColor = Color.White;
                dataGridView1[c, data_i].ToolTipText = "Not available.";
            }
            else
            {
                if (chkTimePlotDB.Checked)
                {
                    Int16 val =
Convert.ToInt16(Math.Abs(ImageDevice.acquisitionData[data_i].ValuesdB[c]) * 10);
// All dB values are negative or zero
                    val += Convert.ToInt16(numMax.Value*10);

                    if (val >= 0 && val < colors_dB.Count)
                    {
                        dataGridView1[c, data_i].Style.BackColor =
colors_dB[val];
                    }
                    else
                    {
                        numMax.Value =
Convert.ToDecimal(Math.Ceiling(AData.maxValuedB));
                        numMin.Value =
Convert.ToDecimal(Math.Floor(AData.minValuedB));
                        TimePlotSetup();
                    }
```

```
                    dataGridView1[c, data_i].ToolTipText = "Ch " + (c + 1) + ": "
+ (Math.Round(ImageDevice.acquisitionData[data_i].ValuesdB[c] * 100) / 100.0) + "
dB at " + ImageDevice.acquisitionData[data_i].DateTime.ToString();
                }
                else
                {
                    UInt16 val =
Convert.ToUInt16(ImageDevice.acquisitionData[data_i].Millivolts[c]);

                    if (val < colors.Count)
                    {
                        dataGridView1[c, data_i].Style.BackColor = colors[val];
                    }
                    else
                    {
                        dataGridView1[c, data_i].Style.BackColor = Color.Black;
                    }
                    dataGridView1[c, data_i].ToolTipText = "Ch " + (c + 1) + ": "
+ Math.Round(ImageDevice.acquisitionData[data_i].Intensity[c] * 1000) / 1000.0 +
" relative intensity at " +
ImageDevice.acquisitionData[data_i].DateTime.ToString();
                }

            }
        }
    }
    if (dataGridView1.RowCount > 0)
        dataGridView1.FirstDisplayedScrollingRowIndex = dataGridView1.RowCount -
1;

}

private void TimePlotResizeRows()
{
    /* Variables to adjust row height to window size */
    int maxRow = (int)showN;

    int rowHeight = dataGridView1.ClientSize.Height / maxRow;
    double rowAddEvery = maxRow / (double)(dataGridView1.ClientSize.Height -
(maxRow * rowHeight));
    double rowCnt = 0;

    for (int r = 0; r < dataGridView1.Rows.Count; ++r)
    {
        if (rowCnt >= rowAddEvery)
        {
            dataGridView1.Rows[r].Height = rowHeight + 1;
            rowCnt -= rowAddEvery;
        }
        else
        {
            dataGridView1.Rows[r].Height = rowHeight;
        }
        ++rowCnt;
    }

    dataGridView1.RowTemplate.Height = rowHeight;
}
```