Universidade de Coimbra

Faculdade de Ciência e Tecnologia

Departamento de Física

NeuroCórnea – Desenvolvimento de Modulo Confocal para Aplicação a Lâmpada de Fenda

A Eletrónica de Controlo e Aquisição de Imagem

João Pedro Soares Albergaria Lamas

Mestrado Integrado em Engenharia Biomédica

Coimbra

This work is funded by FEDER, through the Programa Operacional Factores de Competitividade-COMPETE and by National funds through FCT- Fundação para a Ciência e Tecnologia in the frame of project PTDC/SAU-BEB/104183/2008, F-COMP-01-0124-FEDER-010941

Este trabalho é financiado pelo FEDER, através do Programa Operacional Factores de Competitividade-COMPETE e fundos nacionais através da FCT- Fundação para a Ciência e Tecnologia no âmbito do projeto PTDC/SAU-BEB/104183/2008, F-COMP-01-0124-FEDER-010941

Agradecimentos

A realização deste projeto foi a fase final de um percurso académico ao qual me propus. Além de todo o esforço e dedicação aplicados na realização deste projeto foi indispensável o apoio de um conjunto de pessoas para a conclusão deste. A todas as pessoas que de seguida mencionarei, fica expresso o meu sincero agradecimento.

Ao meu orientador, Professor Dr. José Paulo Domingues, pela sua disponibilidade orientação e saber que me forneceu ao longo da realização do projeto os quais foram fundamentais para a realização deste.

Ao Professor Dr. Custódio Loureiro por toda a ajuda prestada no desenvolvimento do projeto a qual me permitiu solucionar problemas enfrentados no decurso do trabalho desenvolvido.

As todos os elementos do projeto NeuroCórnea pelas suas sugestões ao longo do ano académico para o desenvolvimento do projeto.

Ao Eng. Tiago Marçal pela sua disponibilidade e soluções que me forneceu no desenvolvimento do firmware.

Aos meus pais, irmãos e irmã que sempre me apoiaram no meu percurso académico, só o seu empenho e dedicação permitiu que chegasse a este ponto. Aos meus tios que me acompanharam de perto nesta etapa da minha vida.

A todos os meus amigos e colegas que conheci ao longo do meu percurso académico pelo seu apoio, amizade e sugestões que me proporcionaram para a realização do trabalho desenvolvido.

O meu agradecimento à Sara que me sempre me apoiou e incentivou no desenrolar do projeto.

Resumo

A necessidade do diagnóstico precoce da neuropatia diabética periférica é importante para se tomarem medidas de maneira a reduzir a morbilidade causada por esta, cuja prevalência tem aumentado nos últimos anos.

A microscopia confocal da córnea é um meio de diagnóstico não invasivo que permite fazer o diagnóstico precoce desta neuropatia. O projeto NeuroCórnea tem como objetivo o desenvolvimento de um módulo confocal para aplicação a lâmpada de fenda para captação e análise de imagens dos nervos da córnea. No desenvolvimento deste projeto há a necessidade de desenvolver eletrónica para aquisição de imagens proveniente do sistema ótico.

O desenvolvimento da eletrónica para aquisição de imagem tem como pressuposto a utilização de um microcontrolador para fazer o controlo de uma câmara CCD de alta sensibilidade que permita obter imagens em ambientes de baixa luminosidade.

A conceção deste sistema de aquisição teve como objetivo a criação de um sistema de alimentação para fornecer as necessidades energéticas de todos os componentes, a utilização de um digitalizador de sinal para converter o sinal fornecido pela câmara CCD assim como a utilização de comunicação USB para envio do sinal digitalizado para um computador pessoal.

Palavras-chave: Neuropatia Diabética, Câmara CCD, Microcontrolador, USB

Abstract

The necessity of early diagnosis in peripheral diabetic neuropathy is important to take measures to reduce the morbidity caused by this neuropathy, whose prevalence has increased in recent years.

The confocal microscopy of the cornea is one way of non-invasive diagnosis, which allows the early diagnosis of this neuropathy. The NeuroCórnea project has as goal the development of a confocal module to be applied in a slit lamp to acquire and analise the image of cornea nerves. In the development of this project there is the necessity of electronics development to acquire the images from the optical system.

The development of electronics to image acquisition has as assumption the use of a microcontroller to control a high sensitivity CCD camera that allows the image acquisition in low light environment.

The conception of this acquisition system had as goal the development of power supply system to reach all the energetic needs of all used components, the use of a digital converter to convert the signal of the CCD camera and the use of USB communication to send the digital signal to a personal computer.

Keywords: Diabetic Neuropathy, CCD Camera, Microcontroller, USB.

Índice

Índice de Figuras

Índice de Tabelas

1.Introdução

1.1 Motivação

A *Diabetes Mellitus* é uma doença crónica que afeta um grande número de pessoas na nossa sociedade, sendo mais frequente com o aumento da idade. A taxa de prevalência da diabetes é de 7,3% (diagnosticada) na população portuguesa no ano de 2009 [1]. No ano de 2004, na Europa viviam 60 milhões de pessoas com diabetes e mais de metade não tinha conhecimento desta condição. Esta doença é a quarta maior causa de morte na Europa assim como a maior causa de falha renal e neuropatia [2]. Nos Estados Unidos, no ano de 2010, 25.8 milhões de pessoas têm diabetes o que corresponde a 8.3% da população total deste país e 79 milhões de pessoas vivem em condição de pré-diabetes. A diabetes contribuiu para a morte de mais de 230.000 pessoas em 2007 nos Estados Unidos. Cerca de 60% a 70% de pessoas com diabetes têm formas leves a severas de danificação do sistema nervoso devido a neuropatia [3].

A *Diabetes Mellitus* pode levar a complicações crónicas a diferentes níveis: oftalmológicos (retinopatia diabética, edema da mácula), renais, gastrointestinais, uro-genitais, cardiovasculares, dos membros inferiores e neurológicos. Das complicações crónicas a nível neuronal fazem parte a polineuropatia simétrica distal, as mononeuropatias e a neuropatia autonómica (neurovegetativa) [4].

O estudo da destruição nervosa devido à elevada concentração de glucose na corrente sanguínea tem vindo a aumentar. A neuropatia diabética é uma desordem ao nível dos nervos desenvolvida ao longo do tempo no corpo do paciente diabético [5]. A neuropatia diabética é uma polineuropatia específica [4].

O diagnóstico precoce é importante para se poderem tomar medidas em relação aos problemas que possam surgir com a neuropatia diabética [5].

A integração no projeto NeuroCórnea, que tem como objetivo o diagnóstico precoce de doentes que padeçam de neuropatia diabética, constituiu uma fonte de

motivação para o desenvolvimento do projeto curricular assim como para a escrita da tese de Mestrado.

A conclusão do curso de Mestrado Integrado em Engenharia Biomédica foi igualmente uma fonte de motivação para a realização deste projeto.

1.2 Objetivos

O objetivo deste projeto consiste no desenvolvimento de eletrónica para controlo e aquisição de imagem. O trabalho elaborado ao longo do ano académico fez parte do projeto NeuroCórnea – Diagnóstico precoce e seguimento da neuropatia diabética periférica através da análise automática *in vivo* da morfologia dos nervos da córnea, focando-se no desenvolvimento de hardware associado a esse projeto.

O trabalho foi realizado por diferentes etapas que se enumeram sucintamente:

- Análise de diferentes câmaras CCD existentes no mercado;
- Desenvolvimento de um sistema de alimentação para o módulo de Hardware e para duas câmaras CCD pré-selecionadas;
- Seleção de um microcontrolador e de um ADC para desenvolvimento do sistema;
- Programação de firmware em microcontrolador *PIC*;
- Desenvolvimento de software para interface com utilizador;
- Controlo e aquisição de uma das câmaras CCD com digitalização de sinal.

Nesta Tese de Mestrado é pretendido dar a conhecer todo o trabalho realizado nas diferentes etapas enumeradas, assim como os resultados atingidos.

No decurso da escrita da tese é também pretendido dar um conhecimento teórico sobre as bases científicas do projeto NeuroCórnea assim como da teoria inerente à programação utilizada para a realização do sistema de aquisição.

Como objetivo pessoal, a aquisição de conhecimento constituiu o objetivo principal na realização deste projeto.

1.2.1 Estrutura da Tese

A Tese de Mestrado está dividida em diferentes capítulos:

Capítulo 1 – Introdução

Este capítulo dá a conhecer as motivações para o desenvolvimento deste projeto assim como os objetivos implícitos.

Capítulo 2 – Contexto

Neste capítulo é apresentado um contexto teórico que o aluno teve que adquirir para enquadramento no projeto NeuroCórnea assim como um resumo deste mesmo projeto.

Capítulo 3 – Ferramentas Técnicas

A utilização de diferentes ferramentas para a realização deste projeto requer um conhecimento teórico sobre elas. Neste capítulo são apresentados alguns aspetos fundamentais e relevantes relativamente a essas ferramentas, mais concretamente sobre os microcontroladores e a programação destes, programação em *Visual Studio C++* e sobre as câmaras CCD.

Capítulo 4 – Desenvolvimento

Neste capítulo será explicado em detalhe o trabalho prático desenvolvido ao longo do projeto.

Capítulo 5 – Resultados

Neste capítulo são apresentados os resultados obtidos com o hardware desenvolvido.

Capítulo 6 – Conclusão

Aqui será apresentada a conclusão final do trabalho assim como o trabalho a desenvolver no futuro.

2. Contexto

2.1 Neuropatia Diabética

A neuropatia diabética é uma desordem neurológica que advém de uma doença primária, a *Diabetes Mellitus*. Esta desordem neurológica é uma das complicações da diabetes que mais condiciona a vida do doente devido à dor, desconforto e incapacidade que pode causar [6].

Este tipo de neuropatia pode afetar diferentes partes do corpo humano, podendo se focar num nervo especifico ou numa região do organismo [6].

As principais características morfológicas da neuropatia diabética incluem uma alteração às fibras mielinizadas, resultando numa desmielinização destas assim como a sua degeneração axonal. A microangiopatia endoneural é também uma consequência da neuropatia diabética. Num estado final existe uma perda de fibras nervosas [7].

A prevalência de neuropatia diabética é de aproximadamente 55% em doentes com *Diabetes Mellitus* do tipo I e de 45% para doentes de tipo II, durante a vida [4].

2.2 Instrumentação Médica – Microscopia Confocal

O desenvolvimento da microscopia confocal surge com a necessidade de ver células e estruturas num modo mais dinâmico do que a microscopia convencional [8].

A microscopia confocal, desenvolvida inicialmente por Marvin Minsky em 1955, baseia-se na iluminação ponto a ponto da amostra, utilizando um orifício de abertura para evitar a maioria da luz não desejada por *scattering.*

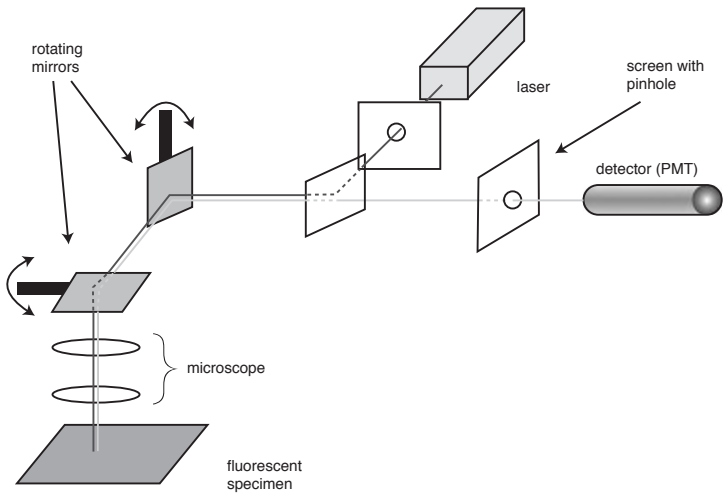


Figura 1 - Esquemático de Microscópio Confocal [9].

Como demostra a imagem, o principio ótico da microscopia confocal está na utilização de um orifício de abertura tanto a partir do ponto de iluminação como no ponto de deteção. Deste modo será possível eliminar a luz refletida acima e abaixo do plano de focagem o que irá permitir uma melhor resolução e contraste do ponto de iluminação pretendido [9].

Para obter um maior campo de visão basta amostrar ponto a ponto, deslocando-se ou a amostra ou o ponto de iluminação de modo síncrono [9].

Como a obtenção de imagem é feita ponto a ponto e a área de cada ponto é restringida pela abertura do orifício, da luz refletida poucos fotões chegarão ao detetor, sendo necessário para tal um longo tempo de exposição por ponto. Isso fará com que o tempo necessário para obter uma imagem seja elevado. Para contornar esta limitação tem que se utilizar uma fonte de luz de alta intensidade [9]. Inicialmente, Minsky utilizava lâmpadas de arco de zircónio, atualmente são utilizados lasers como fonte de iluminação [9].

O microscópio confocal permite uma melhor resolução lateral quando comparado com o microscópio convencional. Esta melhoria da resolução lateral pode ser verificada pelo *point spread function* (demonstra a resposta a uma fonte pontual) dos dois tipos de microscopia como demonstra a imagem seguinte [10].

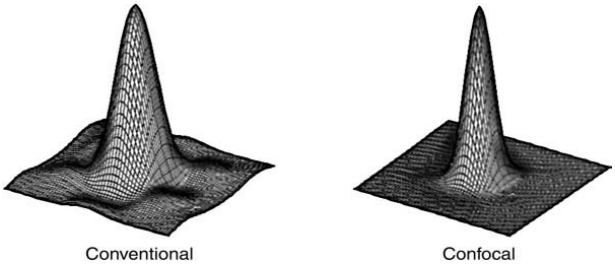


Figura 2 - Point Spread Function para Microscópio Convencional e Confocal [10]

A melhoria da resolução axial é explicada pela imagem seguinte, onde se podem verificar que a imagem proveniente de um plano abaixo do de focagem, não chega ao detetor [10].

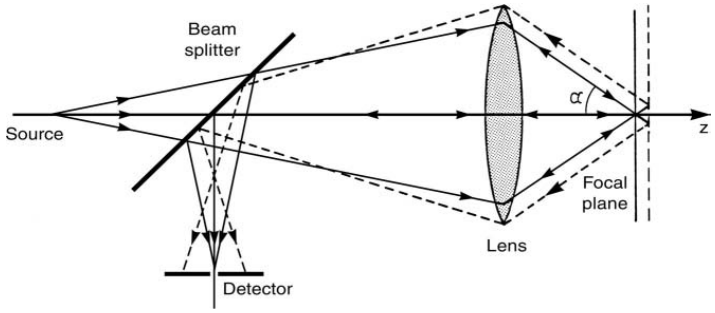


Figura 3 - Esquemático de incidência de luz numa amostra utilizando Microscópio Confocal [10].

2.2.1 Microscopia Confocal da Córnea

A córnea humana é constituída por diferentes camadas: células epiteliais de superfície, células adjacentes, células epiteliais basais, camada de Bowman, estroma, membrana de Descement e células endoteliais. A córnea humana é um dos tecidos com mais fibras nervosas sensoriais [11].

A biomicroscopia da córnea tem como objetivo a melhor compreensão da estrutura e função da córnea no paciente saudável, no paciente doente, com o envelhecimento ou após cirurgia. A biomicroscopia da córnea partiu das lâmpadas de fenda aos microscópios confocais atuais que possuem melhores resoluções laterais e axiais ao nível micrométrico o que permite uma melhor visualização das diferentes estruturas da córnea [12].

A utilização de microscópios confocais permite obter imagens de córneas para estudo, mais concretamente, na regeneração nervosa da córnea pós cirurgia, na formação de cicatriz na córnea, anomalias na córnea causadas por doença e outras finalidades [12].

É possível obter com a microscopia confocal as estruturas celulares e sub-celulares em toda a córnea [12].

A microscopia confocal tem sido um método amplamente utilizado para observação da córnea, pois é um método não invasivo que permite obter imagens *in vivo* da córnea.

2.3 O Projeto NeuroCórnea

O projeto NeuroCórnea tem como objetivo principal o desenvolvimento de uma nova técnica não invasiva para diagnóstico precoce e seguimento de doentes com neuropatia diabética periférica. Este projeto é baseado no desenvolvimento de software para análise automática *in vivo* de imagens dos nervos sub-basais da córnea, assim como no desenvolvimento de instrumentação simples e de baixo custo de maneira a ser possível diagnosticar um número máximo de diabéticos que desenvolvam a neuropatia diabética. Deste modo, o desenvolvimento da instrumentação é feito com o objetivo de ser acoplado a uma lâmpada de fenda de maneira a ser amplamente utilizado por médicos da especialidade [13].

O projeto NeuroCórnea é portanto desenvolvido em diferentes secções. Uma parte está relacionada com o desenvolvimento de software para análise automática de imagens dos nervos da córnea, a outra parte está relacionada com o desenvolvimento de instrumentação do projeto, a qual se pode dividir em dois módulos distintos: o desenvolvimento de ótica e o desenvolvimento de eletrónica [13].

2.3.1 Módulo Ótico

Atualmente a observação dos nervos sub-basais da córnea são realizados utilizando microscopia confocal, no entanto os microscópios confocais são muito caros e encontram-se somente em hospitais centrais.

Esta parte do projeto NeuroCórnea tem como finalidade a construção de um módulo confocal para aplicar a uma lâmpada de fenda. O desenvolvimento do módulo confocal será realizado com o seu próprio sistema de iluminação, sendo utilizada uma lâmpada de halogéneo [13].

2.3.2 Módulo Eletrónico

O módulo eletrónico consistiu no desenvolvimento de hardware para aquisição de imagem proveniente do sistema ótico do projeto NeuroCórnea. Esta parte do projeto partiu como base para esta tese de mestrado e é de salientar que o sistema foi feito de raiz.

O desenvolvimento do hardware teve como pressuposto a utilização de microcontroladores como ponte de ligação entre o instrumento de aquisição (câmara CCD) e o utilizador final [13].

3 Ferramentas/Técnicas

3.1 Programação em C

A linguagem de programação C é uma linguagem computacional flexível e poderosa que permite a rápida execução de programas e que propicia baixas restrições ao programador.

Esta linguagem de programação foi criada por Dennis Ritchie em 1972 nos *Bell Telephone Laboratories*. A linguagem em C foi desenvolvida para permitir a escrita do sistema operativo *Unix*.

A programação em C permite uma rápida execução dos programas porque tem a capacidade de aceder a comandos de baixo nível no entanto utiliza uma sintaxe de nível superior quando comparado por exemplo com linguagem *Assembler*.

Este tipo de linguagem é flexível pois o programador pode obter de diferentes maneiras uma mesma finalidade. Para além disso, a programação em C permite fazer operações de baixo nível como manipulação de bits (necessária ao desenvolvimento do projeto).

A programação em C ainda tem a capacidade de uso de librarias, que podem ser utilizadas sempre que sejam precisas, permitindo aceder a funções existentes nessas librarias para todo o tipo de tarefas, aumentando deste modo as funcionalidades desta linguagem [14].

O desenvolvimento de aplicações em C implica 4 fases distintas:

- Edição do código fonte – trabalho realizado pelo programador, escrita de ficheiros com extensão .c;
- Compilação do programa – verificação se a sintaxe das instruções está correta. Caso haja erros o utilizador terá que voltar à edição do código fonte. O compilador ainda verifica se o código fonte levanta suspeitas em alguma

situação emitindo avisos (*Warnings*). No final da compilação é criado pelo compilador um ficheiro objeto (.obj em *DOS* ou .o em *Unix*);

- "*Linkagem*" dos objetos – a fase de *linkagem* verifica símbolos e funções usados no código fonte. Verifica se os símbolos e funções estão no código fonte ou em librarias utilizadas. Este processo por norma realiza-se juntamente com a compilação do programa. No final é criado um ficheiro executável.
- Execução do Programa – Se o processo de *linkagem* não deu erro, o ficheiro executável é criado e executa o que é suposto fazer [15].

O processo de edição de código fonte por norma é realizado num editor especifico que vem associado a cada compilador o qual pode oferecer ajuda na escrita, desenvolvimento e teste dos programas. A este conjunto é dado o nome de *integrated development environment*, ou *IDE*. No caso do desenvolvimento de *firmware* do projeto é utilizado o *MPLAB IDE* e no desenvolvimento do software é utilizado o *Microsoft Visual Studio 2008 IDE*.

Os programas escritos com linguagem C têm que possuir no mínimo uma função a qual tem que obedecer a uma estrutura predeterminada. A única função que tem obrigatoriamente de ser criada no programa é chamada de *main*. No entanto outras funções podem ser criadas e posteriormente invocadas dentro desta função, pois esta função *main* é a primeira a ser executada pelo programa [16].

Na imagem seguinte é apresentada a estrutura base de um programa em C.

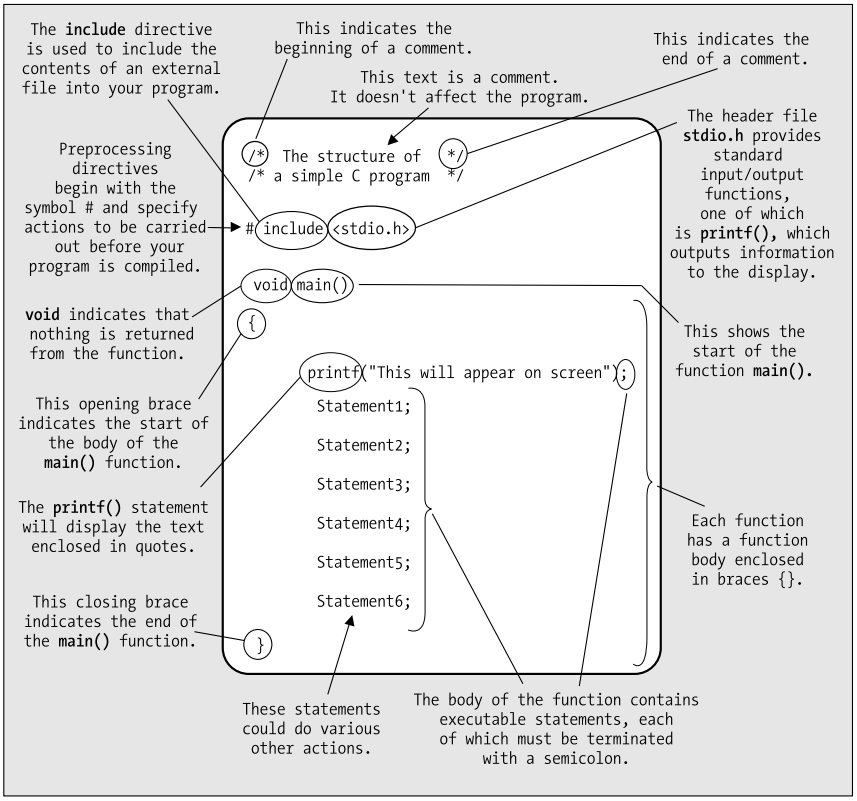


Figura 4 - Estrutura de um programa em C [16].

Na construção de programas em C quando se define uma função todo o código a executar nessa função encontra-se entre chavetas [16].

Um ponto essencial em programação C está relacionado com as diretivas de pré-processamento. Quando um programa é compilado todas as instruções ou diretivas de pré-processamento, que são precedidas pelo caractere cardinal #, são tratadas pelo Pré-Processador [16].

As diretivas de pré-processamento mais usadas são o *#define* e o *#include*. A primeira define constantes ou macros (macros são pequenas porções de código que o pré-processador substitui antes do compilador passar pelo código) enquanto a segunda serve para o pré-processador substituir a linha pelo conteúdo do ficheiro declarado nessa linha [16].

A declaração de variáveis é outro ponto essencial na programação em C. A declaração de variáveis tem que ser feita antes de qualquer instrução e da sua respetiva utilização. Estas são armazenadas em memória e a cada variável está associado um tipo. Os tipos básicos são *char* (caractere), *int* (inteiro), *float* (real) e

double (real). As suas dimensões são diferentes tendo o *char* a dimensão de 1 byte, *int* 2 bytes, *float* 4 bytes e *double* 8 bytes [15].

Para além disso ainda é possível atribuir a designação de *short* e *long* aos inteiros. Designando um inteiro sem *short* ou *long*, automaticamente é lhe atribuído a dimensão de *short* (2 bytes) tendo o *long* a dimensão de 4 bytes [15].

Outra atribuição dada às variáveis é a de *signed* e *unsigned* que indica com sinal e sem sinal respetivamente (por exemplo uma variável de 2 bytes em *signed* varia entre -32768 e +32767 enquanto que em *unsigned* varia entre 0 e 65535) [15].

Para finalizar, outro ponto essencial à realização deste projeto foi a manipulação de bits e o uso de operadores. Os operadores para programação em C são o *&* (realiza um E bit a bit), | (realiza um OU bit a bit), ^ (realiza um OU Exclusivo bit a bit), ~ (transforma todos os bits no bit contrário) e os operadores << e >> (rodam n bits à esquerda e à direita respetivamente) [15].

3.2 Microcontrolador

Para o desenvolvimento do módulo de aquisição, neste projeto, o uso de microcontroladores foi essencial. Ao longo do projeto grande parte do tempo despendido foi na aquisição de conhecimento deste componente.

Nesta secção será abordada a parte teórica acerca dos microcontroladores e no capítulo seguinte será feita uma abordagem mais especifica do microcontrolador utilizado (*PIC32*) e de todos os módulos do microcontrolador necessários para a realização do projeto.

Os microcontroladores tiveram o seu inicio com o desenvolvimento de circuitos integrados. Os microcontroladores funcionam como uma unidade de processamento aliada a diferentes componentes tais como memória ou unidades para envio e receção de informação. Diferem portanto dos microprocessadores por integrarem num só chip todos os elementos que o tornam versátil à sua função. Funcionam portanto como um computador num chip [17].

Ao serem de pequenas dimensões e terem a função de microprocessador para uma função específica, estes dispositivos vieram contribuir significativamente para o desenvolvimento de dispositivos eletrónicos poupando espaço e tempo na sua construção. A possibilidade de serem programados e obedecerem a determinadas instruções (*firmware*) fazem com que sejam versáteis para as suas múltiplas aplicações [17].

Na imagem seguinte é possível ver esquematicamente a integração da unidade de processamento com os diferentes componentes.

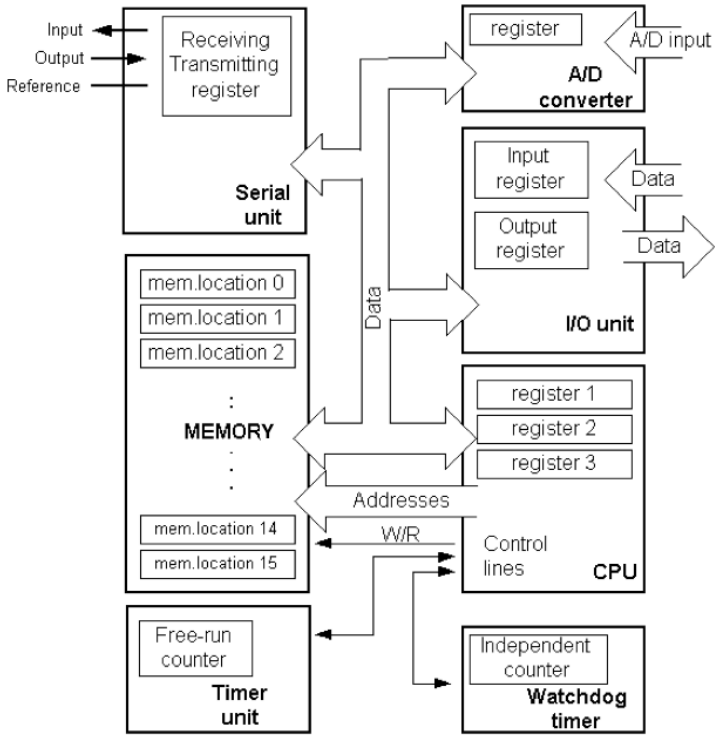


Figura 5 - Esquemático da estrutura interna de um microcontrolador. Adaptado de [17].

Seguidamente serão descritos os constituintes do microcontrolador.

- Unidade de Memória

É nesta unidade que é possível guardar toda a informação. Pode ser vista como um local de espaços vazios os quais podem ser acedidos para escrita ou leitura. Esses locais são acedidos por endereçamento ao local [17].

- Unidade Central de Processamento – CPU (*Central Processing Unit*)

O CPU é a unidade responsável pelo processamento de todas as instruções dadas no *firmware*, todas as operações lógicas e aritméticas. Está diretamente ligada à unidade de memória pois na realização de operações ou na manipulação de dados é necessário a criação de registos onde se colocam esses dados. Estes registos são portanto locais de memória utilizados pelo CPU para processamento de dados [17].

- Bus

O Bus é visto fisicamente como uma linha de 8, 16 ou mais vias. Existem dois tipos: o bus de endereçamento e o bus de dados. O de endereçamento serve para transmitir os endereços da memória do CPU e o de dados serve para fazer a conexão entre todos os blocos internos do microcontrolador [17].

- Unidade de Entrada-Saída

São locais de entrada e saída de informação aos quais se dão o nome de Portas. Podem ser Portas de entrada, saída ou bidirecionais, e quando os utilizamos necessitamos de declarar a sua função e depois receber ou enviar informação. Para a troca de informação utilizam-se os pinos do microcontrolador [17].

- Comunicação em Série

Para trocas de informação com exterior é possível utilizar a unidade de Entrada-Saída para a transmissão de dados. A troca de informações com o exterior tem que ser realizada através da comunicação em série, caso fosse comunicação em paralelo um número elevado de linhas eram necessárias, razão pela qual se torna impraticável num microcontrolador. Sendo realizada num número reduzido de pinos tem que obedecer a determinadas regras – protocolo. Assim sendo a comunicação em série é realizada bit a bit a uma determinada frequência sempre na mesma linha. A informação é guardada na memória após ser recebida e lida do seu registo [17].

Este tipo de comunicação pode ser efetuado bidirecionalmente num número reduzido de linhas, tudo dependente do protocolo utilizado. De entre os protocolos de comunicação em série existentes os mais comuns são *SPI, UART* e I^2C [18].

A comunicação por I^2C é uma comunicação síncrona e é, por exemplo, realizada utilizando duas linhas: um clock e uma linha de transferência de dados bidirecional [18].

A comunicação por *SPI* é igualmente uma comunicação de modo síncrono mas utiliza duas linhas para transferência de dados: uma para receber e outra para enviar. Utiliza igualmente uma linha correspondente ao clock [18].

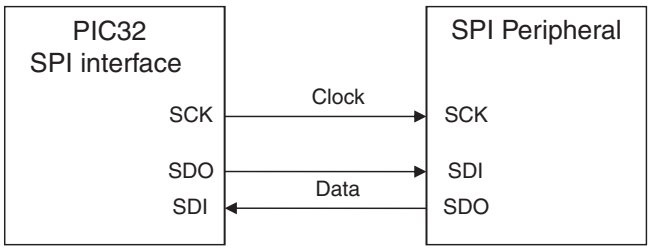


Figura 6 - Esquema de comunicação SPI [18].

Para além disso este tipo de protocolo de comunicação utiliza uma outra linha que é o SS (*Slave Select*) que inicia a transferência de dados com o periférico pretendido. Deste modo todos os periféricos partilham os mesmos pinos de transferências de dados (SDI e SDO) no entanto só efetuam transferências por controlo do *Slave Select* o que irá permitir um maior número de dispositivos ligados ao microcontrolador [18].

O protocolo de comunicação *UART* é um protocolo assíncrono (não tem linha de clock) e possui tipicamente duas linhas uma de receção e outra de envio. Opcionalmente, este tipo de comunicação possui ainda duas linhas de *handshake* que confirmam o envio ou a receção de dados [18].

No quadro seguinte é demonstrado as principais diferenças entre os diferentes tipos de protocolos de comunicação.

	Synchronous		Asynchronous
Peripheral	SPI	I^2C	UART
Max bit rate	20 Mbit/s	1 Mbit/s	500 kbit/s
Max bus size	Limited by number of pins	128 devices	Point to point (RS232), 256 devices (RS485)
Number of pins	3 + n x CS	2	2(+ 2)
Pros	Simple, low cost, high speed	Small pin count, allows multiple masters	Longer distance (use transceivers for improved noise immunity)
Cons	Single master, short distance	Slowest, short distance	Requires accurate clock frequency
Typical application	Direct connection to many common peripherals on same PCB	Bus connection with peripherals on same PCB	Interface with terminals, personal computers, and other data acquisition systems
Examples	Serial EEPROMs (25CXXX series), MCP320X A/D converter, ENC28J60 Ethernet controller...	Serial EEPROMs (24CXXX series), MCP98XX temperature sensors, MCP322x A/D converters . . .	RS232, RS422, RS485, LIN bus, MCP2550 IrDA interface . . .

Tabela 1 - Comparativo entre diferentes meios de comunicação em séria em microcontroladores. Adaptado de [18]

- Unidade Temporal - Timers

A unidade temporal não é mais que um contador (registo que é incrementado a soma de um valor a uma determinada frequência) controlado por um oscilador de cristal externo. Esta unidade é uma das mais importantes quer no controlo de duração de tarefas, quer no uso de protocolos de comunicação [17].

- *Watchdog*

Esta unidade é responsável pelo reiniciar do microcontrolador quando este está a funcionar incorretamente. Como não há nenhum mecanismo ao qual possamos aceder para fazer reset ao microcontrolador, esta unidade funciona como um

contador independente que está à espera de uma escrita de um valor zero sempre que o programa é executado corretamente. Caso não haja essa escrita é dada uma ordem de reiniciar ao microcontrolador [17].

- Conversor Analógico-Digital

Como podemos enviar informação para o microcontrolador e este é um dispositivo digital, caso esteja a receber um sinal analógico, é necessário converter o sinal analógico em linguagem binária.

Assim sendo os microcontroladores incorporam um conversor analógico-digital para o CPU poder processar todas as informações que recebe [17].

3.3 Programação *Visual Studio C++*

O *Microsoft Visual Studio* foi a aplicação utilizada para o desenvolvimento de software do projeto. Através deste programa foi possível desenvolver uma interface gráfica com o utilizador (GUI – *Graphical User Interface*) que tivesse interação com o hardware desenvolvido.

O *Microsoft Visual Studio* é um produto da *Microsoft* e constitui um IDE (Integrated Development Environment) no qual fazem parte um editor, um compilador, um *linker* e librarias. Estas constituem as ferramentas básicas para desenvolvimento de aplicações em C++ [19].

Com a utilização do *Visual C++ 2008* existem dois tipos fundamentais de aplicações C++ que se podem fazer. O primeiro é a realização de aplicações nativas em C++ (utilização standard desta linguagem de programação) podendo ser utilizado o *Microsoft Foundation Classes* (MFC) na elaboração de aplicações em *Windows.* A outra opção é realizar aplicações com o controlo do CLR (*Common Language Runtime*). O CLR é um ambiente no qual se pode programar utilizando diferentes linguagens de programação de alto nível tais como Visual Basic, C#, e obviamente C++. A este tipo de programação em C++ utilizando o CLR também se pode referenciar como *managed C++* por ser realizado sobre o controlo do CLR [20].

O CLR faz parte de um conceito do *Visual Studio* que são as bibliotecas de classes (*Framework Classes*).

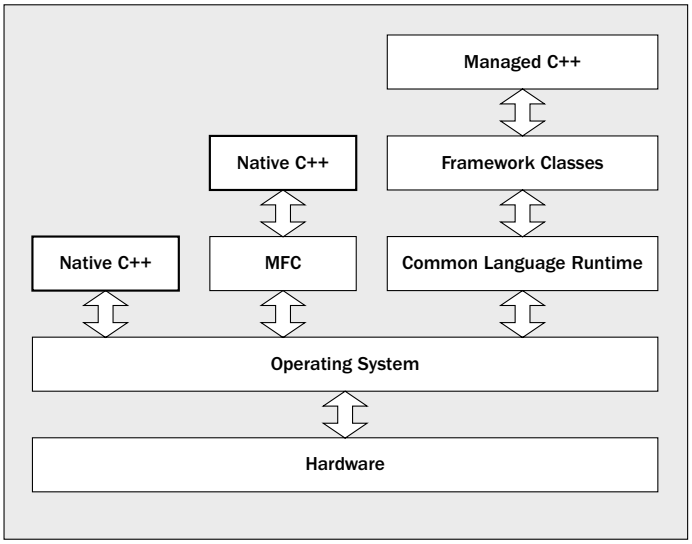


Figura 7 - Esquema de diferentes meios de programação em C++ [20]

O desenvolvimento de aplicações em C++ executando o CLR permite obter uma fácil construção de aplicações em janelas. Para tal, utiliza-se o *Windows Forms* como base para o desenvolvimento de uma GUI. A utilização do *Windows Forms* permite obter uma rápida criação de um programa em janelas, sendo possível montar todos os objetos (botões, gráficos, caixas de texto, etc.) numa interface gráfica com auxilio de uma *toolboox*, sendo o código gerado automaticamente, necessitando somente de ser alterado para obter a funcionalidade pretendida [20].

No desenvolvimento de aplicações em janelas é possível recriar várias interações a qualquer momento entre o utilizador e a aplicação, tal como clicar num botão, clicar num menu ou em qualquer parte da janela. A primeira instância de comunicação com o utilizador é o sistema operativo *Windows*, e todas as ações do utilizador são consideradas pelo *Windows* como eventos. Estes eventos é que vão fazer com que uma parte do programa desenvolvido seja executada. Os programas são portanto dirigidos a eventos (*event-driven programs*) como demonstra a imagem seguinte [20].

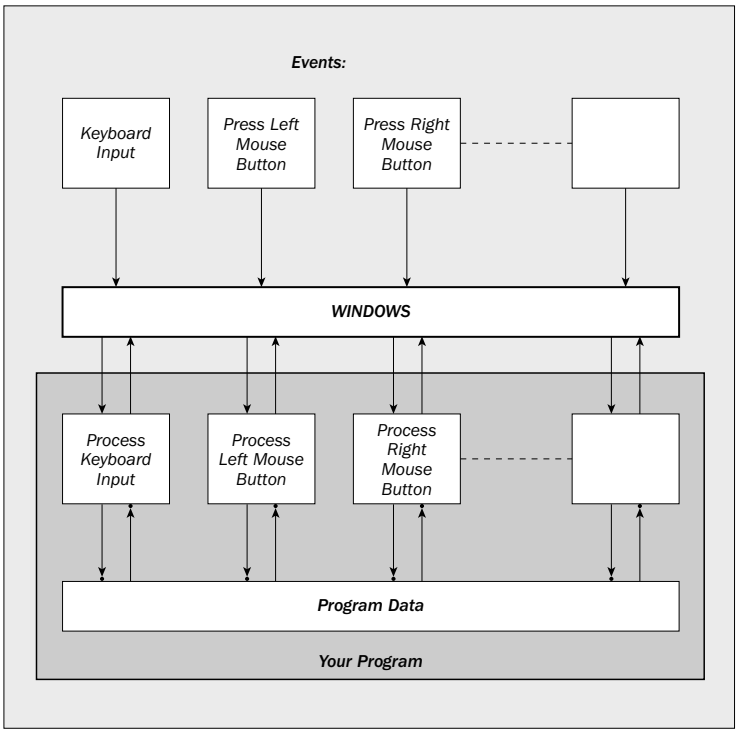


Figura 8 - Esquema de interface entre eventos e o programa [20]

Em relação a variáveis e operadores, a programação em C++ utilizada para tratamento de dados neste projeto é muito semelhante à programação em C já descrita anteriormente, pois C++ é a linguagem C com classes [20].

3.4 Câmaras CCD

O CCD (*Charge-Coupled Device*) foi inventado por Willard Boyle e George Smith em 1969 nos *Bell Telephone Laboratories* [21].

Genericamente, o CCD é um sensor utilizado para obter imagens e baseia-se na carga acumulada numa pequena área por efeito fotoelétrico. Esta carga fica acumulada em poços de potencial a qual é posteriormente transferida de poço em poço por aplicação de uma tensão. Este poços de potencial são baseados num condensador MOS (*Metal Oxide Semiconductor*) como demonstra a imagem [22].

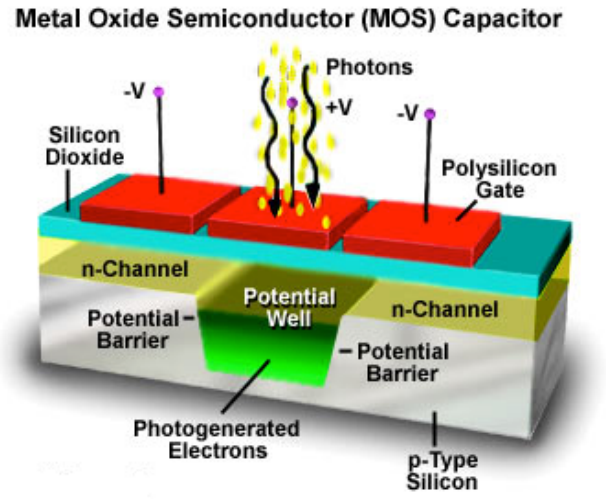


Figura 9 - Condensador MOS. Adaptado de [22].

A proximidade dos condensadores e a aplicação de uma tensão permite que a carga acumulada em cada poço seja transferida ao poço de potencial vizinho, e assim sucessivamente, sendo acumuladas numa parte terminal da região fotossensível e medida a soma de todas as cargas dessa região. Diferentes valores de carga são medidos em diferentes regiões (pixéis) e daí é possível obter uma imagem [22].

Ao longo dos tempos os sensores CCD tem vindo a evoluir. A resolução de imagem é dada pelo número de pixéis que tem vindo a aumentar ao longo dos anos permitindo obter imagens de elevada resolução [21].

Na fase inicial deste projeto foi feito um estudo de diferentes câmaras CCD existentes no mercado, sendo as especificações de sensibilidade (detetividade) e custo os parâmetros mais significativos nesse estudo, pois a aplicação da câmara CCD é para baixos níveis de luminosidade.

Destaque para algumas das principais marcas de fabricantes de câmaras CCD, a Hamamatsu ou a Proxitronic.

Na fichas técnicas das câmaras CCD são fornecidas características das câmaras tais como área de pixel, tamanho de pixel, sensibilidade (resposta de saída em relação à entrada – em termos de ganho), detetividade (quantidade mínima de luz que a câmara mede com certeza), corrente no escuro, tempo máximo de exposição (para

não ocorrer saturação da câmara – existe um valor de carga máximo que o poço de potencial consegue acumular), entre outras.

Na avaliação das diferentes CCD, serão apresentadas as características com mais interesse para a seleção da câmara.

4. Desenvolvimento

4.1 Hardware

O desenvolvimento de hardware para este projeto partiu com base no principio de utilização de um microcontrolador para controlo de diferentes dispositivos. Inicialmente pretendia-se que o microcontrolador fosse a unidade de controlo para ativação da Câmara Proxitronic HL5, receção do sinal digitalizado desta e envio de vídeo para um computador.

O microcontrolador seria também o dispositivo responsável para ativação da Câmara Hamamatsu C5809, receção do sinal digitalizado e envio para o computador.

Para além disso pretendeu-se fazer com que o microcontrolador pudesse receber um sinal digitalizado não designado, funcionando como um osciloscópio digital.

A interface de comunicação pretendida entre o microcontrolador e o computador foi a comunicação USB (*Universal Serial Bus*).

Devido à complexidade do sinal em vídeo composto (CVS) fornecido pela Câmara HL5 procedeu-se, em alternativa, à utilização de um *frame grabber* para adquirir o sinal de vídeo desta câmara CCD.

4.1.1 Diagrama Geral

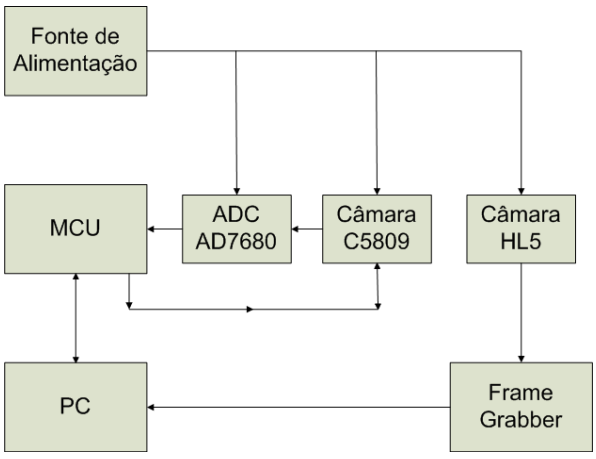


Figura 10 - Diagrama de blocos do hardware desenvolvido.

Neste diagrama de blocos podemos ver a arquitetura do hardware desenvolvido. No desenvolvimento do sistema, foi realizada a construção de um sistema de alimentação, partindo da mesma fonte de alimentação, no qual fosse possível alimentar o ADC (*Analog-to-Digital Converter*) e as duas câmaras CCD utilizadas.

Neste sistema, a comunicação entre o microcontrolador e o ADC é realizada por comunicação SPI, comunicação bidirecional para aquisição do sinal digitalizado.

Existe uma interface bidirecional entre o microcontrolador e a Câmara CCD C5809 da Hamamatsu na qual o microcontrolador envia sinais de controlo (um *Clock* e um *Start*) para iniciar a CCD com determinadas características enquanto esta envia determinados sinais (*Trigger* e *End Of Scan*) para o microcontrolador.

O sinal de vídeo da Câmara CCD C5809 da Hamamatsu é enviado diretamente para o digitalizador.

A Câmara HL5 da Proxitronic envia o sinal de vídeo para um *frame grabber* acoplado a um computador, como já anteriormente referido.

Seguidamente serão descritos todos os elementos de hardware utilizados, as suas características e o trabalho desenvolvido com estes.

4.1.2 Câmaras CCD a usar

Inicialmente, foi feito um estudo de diferentes câmaras CCD existentes no mercado para aquisição.

A utilização da câmara CCD é realizada em condições de baixa luminosidade, pelo que o dispositivo pretendido teria que possuir sensibilidades elevadas (sensibilidade aqui é entendida não como ganho, mas sim como detetividade).

Foi necessário comparar diferentes modelos de câmaras, utilizando como critério de seleção a sensibilidade e o preço.

Na especificações das câmaras CCD, o valores de sensibilidade que não estavam disponíveis na unidade fotométrica de iluminância (*lux*), tiveram que se calcular a partir da leitura de ruído da CCD (*readout noise*) fazendo a conversão de unidade radiométrica para fotométrica, sendo necessário utilizar um tempo de exposição comum (20ms).

Partindo da energia de fotão:

$$E = n\,.h\,.\nu\; = \frac{n\,.h\,.c}{\lambda}\quad (\mathrm{J})$$

Utilizando a eficiência quântica nos 500 nm, para cálculo do valor de n (número de fotões incidentes), foi utilizado em todos os cálculos duas vezes o valor do *readout noise* (partindo do princípio que por cada dois fotões incidentes é gerado um eletrão para além da eficiência quântica).

$$n = \frac{2 \times Readout\ Noise}{Ef.Quântica}$$

Da energia obtida, calculou-se o fluxo utilizando um tempo de exposição de 20 ms.

$$\phi = \frac{E}{T_{exp}}\ (W)$$

Do fluxo obteve-se a Irradiância (E_e) utilizando a área da superfície na qual os eletrões incidem que corresponde à área do pixel.

$$E_e\ = \frac{\phi}{A_{pixel}}\ (\mathrm{W}.m^{-2})$$

Fazendo a conversão para unidade fotométrica, utilizando a resposta fotópica nos 500 nm que corresponde a um fator de 0.33 aproximadamente, sabendo que a 1W correspondem 683 lm (lúmen), obteve-se a iluminância (E_v) [23] e correspondente sensibilidade.

$$E_v = E_e \times 683 \times 0.33\ (\mathrm{Lx})$$

Os cálculos obtidos para todas as CCDs seguem no anexo 1.

Na tabela seguinte são demostradas as diferentes câmaras analisadas com as suas características principais, não sendo possível apurar o valor da CCD da Hitachi por não envio do valor pelo fabricante.

Marca	Proxitronic	Proxitronic	Proxitronic	Hamamatsu	Hamamatsu
Modelo	µHR0	µHL4 AGC	eCam 200	ORCA Flash 2.8	ORCA R2
Tipo de Sensor	CCD - 1/2" ICX 429 ALL	CCD – com Intensificador	EMCCD - 1/2"	CMOS FL-280	CCD ER-150
NºPixeis (H)x(V)	752 x 582	752 x 582	658 x 496	1920x1440	1344 x 1024
Tam. Pixel (µm) (H)x(V)	8,6 x 8,3	8,6 x 8,3	10 x 10	3,63x3,63	6,45x6,45
Sensibilidade (mLx)*	**10**	**0,01**	**0,05 (a 25 ms)**	**2,8**	**2,8**
Tempo de Exposição	ND	ND	ND	0.02ms a 10s	0,01ms a 4200s
Arrefecimento	ND	ND	ND	sim	sim
Preço (€)	**1100**	**11400**	**8700**	**9500**	**13300**
Vídeo Output/Data Interface	CVS	CVS	CVS	Camera Link	IEEE 1394b-2002

***Cálculos para t(exp)=20ms**

Marca	Hamamatsu	Hamamatsu	Hamamatsu	Pixelfly	Hitachi
Modelo	ORCA 03G	ORCA 05G	C9300-221	Qe	KP-E500
Tipo de Sensor	CCD ER-150	CCD ER-150	CCD ER-150	CCD - 2/3" ICX 285 ALL	EMCCD - 1/2"
Pixéis (H)x(V)	1344 x 1024	1344 x 1024	640x480	1392 x 1024	680 x 500
Tam. Pixel (µm) (H)x(V)	6,45x6,45	6,45x6,45	7,4x7,4	6,45 x 6,45	10 x 10
Sensibilidade (mLx)*	**2,24**	**2,8**	**5,3**	**2,21**	**0,0008**
Tempo de Exposição	0,01ms a 10s	0,01ms a 1s	0,039ms a 10s	5µs a 65s	ND
Arrefecimento	sim	não	sim	ND	ND
Preço (€)	**7500**	**4400**	**6640**	**5800**	**ND**
Vídeo Output/ Data Interface	IEEE 1394-1995	IEEE 1394-1995	Camera-Link	PCI	CVS

Tabela 2 - Comparativo das diferentes Câmaras CCD analisadas. ND- Informação não disponível.

No final da seleção, a opção recaiu por uma câmara CCD (Proxitronic HL5) existente no centro de investigação *IBILI* (Instituto Biomédico de Investigação da Luz e da Imagem) que não estava a ser utilizada, que reunia os requisitos prévios, evitando no curto prazo mais um custo, ao mesmo tempo que permitia adquirir “know how”.

Mas no projeto foram utilizadas duas câmaras com diferentes finalidades. Uma das câmaras foi a Câmara CCD Proxicam HL5 da Proxitronic para aquisição de imagem proveniente do sistema ótico do projeto NeuroCórnea, sendo utilizado um *frame grabber* para adquirir imagens. A outra câmara utilizada foi a Câmara *FFT-CCD Image Sensor Multichannel Detetor Head C5809* da Hamamatsu. Esta câmara CCD faz parte de um equipamento de diagnóstico em oftalmologia, que é designada por PAF (*Photodiode Array Fluorometer*), existente no IBILI. Este equipamento de diagnóstico baseia-se na fluorómetria ocular, um método não invasivo de diagnóstico por medição da fluorescência em tecidos e fluídos oculares para diagnóstico de patologias oculares [24].

Será dado maior enfâse à câmara da Hamamatsu pois o trabalho desenvolvido com esta foi mais complexo do que o da Proxitronic.

Proxitronic HL5

A Câmara Proxicam HL5 é uma câmara de alta resolução, de alta sensibilidade e baixo ruído ideal para detetar baixos níveis de luz [25].

Figura 11 - Câmara CCD Proxitronic HL5 [25].

Esta câmara utiliza um sensor *CCD Sony ICX 024, black/white 2/3"* com 756 (H) x 581 (V) pixéis versão CCIR 625 linhas a 50 Hz. A saída de vídeo é em sinal analógico de vídeo composto (CVS – *Composite Video Signal*) através de um conector BNC [25].

A Proxicam HL5 possui um intensificador de imagem que possibilita um aumento elevado da sensibilidade quando comparada com uma câmara CCD standard. A sensibilidade espectral vai de 350 nm a 900 nm com um máximo de sensibilidade nos 500 nm sendo a sensibilidade da câmara de 5mLx [25].

A alimentação necessária para esta CCD é de +12V ± 10% com um consumo de corrente até 500mA [25].

Hamamatsu FFT-CCD C5809

A Câmara CCD C5809 da *Hamamatsu Photonics* possui um módulo (*detetor head*, c5809) termoelectricamente arrefecido, que incorpora um sensor de imagem de configuração FFT (*Full Frame Transfer*) da série S5469 da Hamamatsu. Juntamente com este sensor a câmara possui um circuito de controlo/amplificação acoplado e um circuito de controlo de temperatura. Este módulo foi especificamente desenvolvido para espectroscopia e níveis muito baixos de luminosidade. A Câmara CCD C5809 apresenta as seguintes características gerais [26]:

Nº Efetivo de Pixeis	512(H) x 64(V)
Tamanho do Pixel (µm)	24(H) x 24 (V)
Área Ativa Efetiva (mm)	12.28(H) x 1.54(V)
Ruído de Leitura	20 e^-rms
Gama Dinâmica	24 K
Corrente no Escuro	200 e^-/pixel/s
Temperatura do Arrefecimento	0 ^oC

Tabela 3 - Principais características da Câmara CCD Hamamatsu C5809. Adaptado de [26].

O módulo de deteção C5809 opera com sinais externos simples, e no momento em que é ligada, funciona a uma baixa temperatura com auxilio do sistema de

arrefecimento, caso este falhe e provoque sobreaquecimento a câmara desliga automaticamente [26].

Este módulo pode ser dividido em quatro secções distintas que são:

- Sensor de Imagem FFT-CCD termoelectricamente arrefecido

O sensor de imagem é do tipo FFT no qual a secção de transferência de carga é ao mesmo tempo a área fotossensível. Este tipo de sensores permitem um menor ruído e maior velocidade a processar o sinal [26].

O sensor de imagem da câmara, originalmente concebido para operar como sensor de área, pode operar como sensor linear através da operação de *binning* vertical na qual todas as cargas dos registos verticais são somadas no seu correspondente registo horizontal [27].

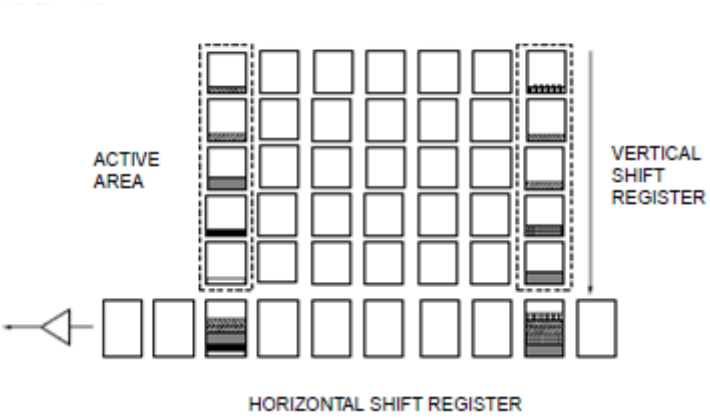


Figura 12 - Binning Vertical na FFT-CCD [27].

Assim sendo, para um sensor com uma área de 512 pixeis horizontais e 64 pixeis verticais, como é o caso desta CCD, obteremos uma imagem linear de 512 pixeis no qual cada pixel horizontal contém a soma dos registos verticais. Estes 512 valores são os valores que serão enviados para o ADC para posterior envio para o microcontrolador [26].

Por ser termoelectricamente arrefecido este sensor permite obter integração de carga por longos períodos de tempo [26].

- Circuito de controlo/amplificação

São duas as funções do circuito de controlo/amplificação: é responsável por fornecer os diferentes sinais de temporização que o sensor necessita para operar e é responsável por processar o sinal de vídeo analógico que vem do sensor com um baixo ruído [26].

São necessários dois sinais de controlo externos para o circuito operar, como já anteriormente referido: um *clock* (CLK) e um *start*. São necessárias quatro tensões de alimentação (+5V, +15V, -15V, +24V) que serão fornecidas pelo sistema de alimentação construído e mais à frente explicitado [26].

Este circuito de controlo/amplificação tem quatro secções: gerador de sinais temporizadores intrinsicos (para o sensor CCD) e externos, *driver* da CCD, regulador de tensão e processador de sinal vídeo [26].

O gerador de sinais temporizadores encontra-se sincronizado com o clock externo (CLK) e é inicializado pelo impulso de inicio externo (Start). Este gerador de sinais para além de fornecer todos os sinais de temporização para o próprio sensor CCD operar, fornece o sinal externo *trigger* para posterior conversão analógico-digital [26].

O *driver* da CCD é responsável por converter os sinais de nível H-CMOS em níveis $V_L=V_{EE}$ e $V_H=V_{DD}$.

O regulador de tensão é responsável por fornecer diferentes tensões de alimentação necessárias para o sensor de imagem operar com elevada precisão e estabilidade.

O processador de sinal vídeo intrínseco à câmara processa o sinal analógico do sensor de imagem com o objetivo de fornecer um sinal com polaridade positiva para conexão ao ADC [26].

- Circuito de controlo de temperatura

O circuito de controlo de temperatura utiliza um termístor montado no sensor de imagem FFT-CCD para monitorização deste enquanto opera. Utiliza a linha +5V e

arrefece o sensor até uma temperatura predeterminada (0^{o}C). Caso o sistema falhe e entre em sobreaquecimento o circuito desliga e um LED indicador vermelho liga [26].

- Revestimento

O revestimento da câmara FFT-CCD, apesar de ser compacto, fornece boa dissipação de calor, assim como a ligação a componentes externos [26].

Funcionamento da FFT-CCD

O funcionamento da câmara CCD requer as ligações às tensões de alimentação, tendo particular atenção à tensão de +5V que requer um valor de corrente inicialmente a 2A até atingir um valor estável entre os 1 e 1,5A [26].

Os sinais de controlo são seguidamente fornecidos à CCD e há alguns aspetos a ter em consideração.

O sinal *clock* e *start* serão fornecidos pelo microcontrolador *PIC32* e iniciam o funcionamento do circuito de condução/amplificação da câmara. Estes dois sinais devem estar sincronizados o máximo possível e a largura do pulso do *start* deve ser maior do que a largura do pulso de *clock*. A largura do *start* determina o tempo de aquisição do sensor e a frequência do clock determina a frequência de leitura do sinal *Data Vídeo* [26].

Da câmara CCD serão fornecidos três sinais: o *Data Vídeo* que será conectado diretamente ao ADC, e os sinais *Trigger* e *End-of-Scan* que serão conectados ao *PIC32*.

No momento em que o sinal *Trigger* é enviado, a conversão analógico-digital deve ser iniciada pois o *Trigger* indica que estará a ser enviado pela CCD um sinal *Data Vídeo* válido. Por cada aquisição, serão enviados 512 *Triggers* que corresponderão aos 512 pixeis da imagem linear [26].

No final a CCD envia o sinal *End-of-Scan* que indica o final da aquisição [26].

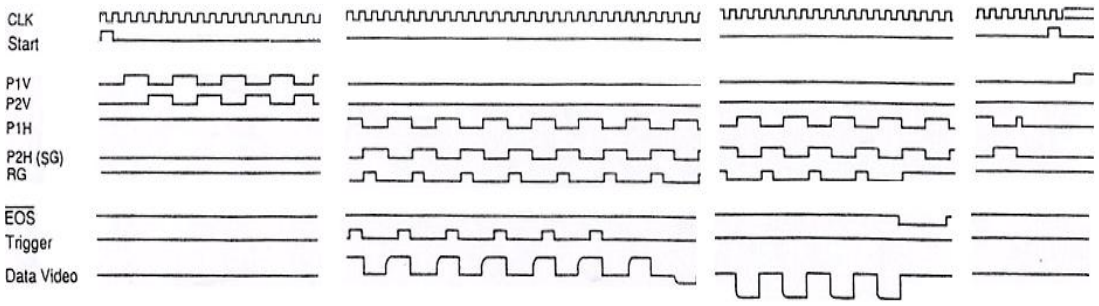


Figura 13 - Sinais da Câmara Hamamatsu C5809. Adaptado de [26].

4.1.3 *USB Starter Kit II* da *Microchip*

O *USB Starter Kit II* é um sistema modular (pode acoplar outras placas) de baixo custo para microcontroladores de 32-bit (*layout* no anexo 2). Este sistema desenvolvido pela *Microchip Technology*, é uma placa de desenvolvimento a qual integra um microcontrolador de 32-bit na sua estrutura, assim como outros componentes que permitem o desenvolvimento de comunicações USB. Esta placa permite igualmente o desenvolvimento de outras aplicações, utilizando uma placa de expansão [28].

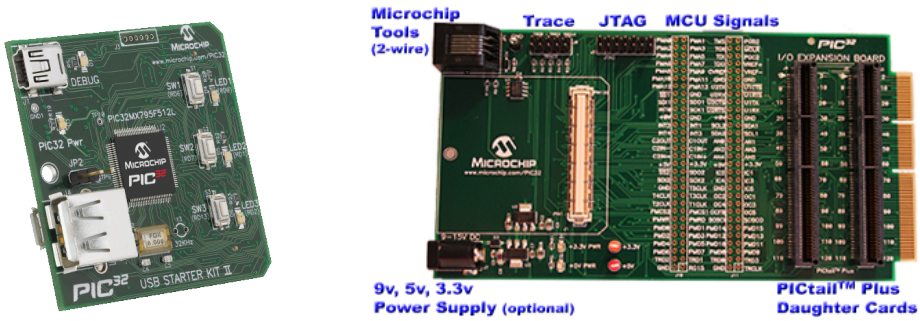


Figura 14 - USB Starter Kit II e Explorer Board IO. [29] e [30].

O *USB Starter Kit II* permite desenvolver comunicações USB de vários tipos: USB *Embedded Host, USB Device* e *USB On-The-Go*. Para isso, esta placa tem como componentes portas USB do tipo A e do tipo micro-AB [28].

Neste projeto para além do *USB Starter Kit II*, foi utilizada a placa de expansão *Starter Kit I/O* (diagrama no anexo 3). Esta placa disponibiliza o acesso a todos os pinos do microcontrolador e a possibilidade de acoplar outros componentes através

da entrada de componentes na *socket PICtail*. Foi ainda utilizada uma placa para fazer prototipagem (as ligações ao ADC) com interface *PICtail*.

4.1.3.1 Microcontrolador PIC32MX

Os microcontroladores *PIC32MX* são uma família de microcontroladores desenvolvidos pela *Microchip* [31].

No *USB Starter Kit II* o microcontrolador que vem integrado é um *PIC32MX795F512L* (diagrama no anexo 4). A escolha do *USB Starter Kit II* foi feita de acordo com as necessidades do projeto. Um dos pontos chave do projeto era realizar a comunicação com o computador através de USB. Assim sendo, este *USB Starter Kit II* possui características ótimas para o desenvolvimento da comunicação USB por integrar os componentes necessários e as ligações necessárias para a comunicação USB. Para além disso, este microcontrolador *PIC32MX* tem como características principais e necessárias ao projeto [31]:

- CPU com 80MHz;
- Memória programável de 512 KB;
- Memória *RAM* de 128 KB;
- Entradas e Saídas Digitais e Analógicas;
- Módulos de comunicação em série SPI, I^2C, UART;
- Interrupções;
- Temporizadores;
- Comparador de Saída/PWM (*Pulse Width Modulation*);
- Conversor A/D;

Os microcontroladores *PIC32MX* possuem uma arquitetura diferente de todas os outros microcontroladores PIC anteriormente desenvolvidos [18].

Existem basicamente dois tipos de arquitetura: *Harvard* e *von-Neumann*. Estes dois tipos de arquitetura distinguem-se basicamente por a arquitetura de *Harvard* possuir duas memórias independentes, a memória de dados (*Data Memory*) e a memória onde estão as instruções (*Program Memory*), com ligações independentes

enquanto a arquitetura de *von-Neumann* tem uma só memória e consequentemente uma única ligação [17].

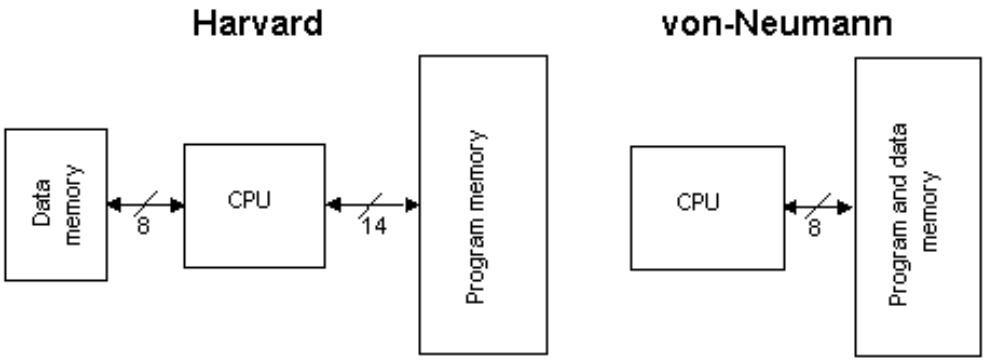


Figura 15 - Arquitetura de Harvard e von-Neumann [17].

No caso dos *PIC32MX* a arquitetura do microcontrolador é do tipo *Harvard* mas modificada.

4.1.4 Sistema de Alimentação

No desenvolvimento do hardware para este projeto foram utilizados diferentes componentes que necessitaram de diferentes tensões de alimentação para o seu funcionamento.

Para tal foi desenvolvido um sistema de alimentação próprio que satisfaça as necessidades de todos os elementos partindo de um transformador portátil AC-DC alimentado pela corrente alterna da rede pública. Foi escolhido o transformador da XP Power, modelo AED100US19 que fornece na saída +19V DC até 100W de potência [32].

Foi necessário criar um sistema de alimentação que fornecesse as seguintes tensões para os seguintes componentes:

- Câmara CCD HL5 da Proxitronic:
  - +12V ± 10%, consumo até 500mA.
- Câmara CCD C5809 da Hamamatsu:

- +5V, consumo até 2A;
- +15V, consumos até 100mA;
- -15V, consumo até 100mA;
- +24V, consumo até 10mA.

- ADC AD7680:
  - +5V ±0.5V, I_{DD}=5.2mA max.

Partindo dos +19V fornecidos pelo transformador foram utilizados os seguintes reguladores e conversores no sistema:

- Regulador L7812CV da *ST Microelectronics*;
- Regulador LM7815 da *Fairchild Semiconductor*;
- Regulador LD1085V50 da *ST Microelectronics*;
- Conversor DC/DC IE1224S-H da *XP Power*;
- Conversor DC/DC PT78NR115S da *Texas Instruments*.

O regulador L7812CV permite obter uma saída V_0 de 12V ± 0.5V com corrente até 1A [33]. Este regulador serve para alimentar a CCD da Proxitronic e será utilizado para alimentar outros componentes. O V_i é obtido do transformador e a montagem típica e construída no sistema é a seguinte:

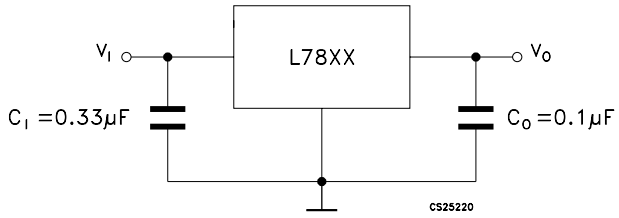


Figura 16 - Montagem Regulador L7812CV [33].

O regulador LM7815 fornece os +15V necessários para a CCD da Hamamatsu a partir dos +19V do transformador. Este regulador fornece uma corrente de saída até 1A. A montagem típica é igual à do regulador L7812CV e foi a montagem realizada no sistema [34]:

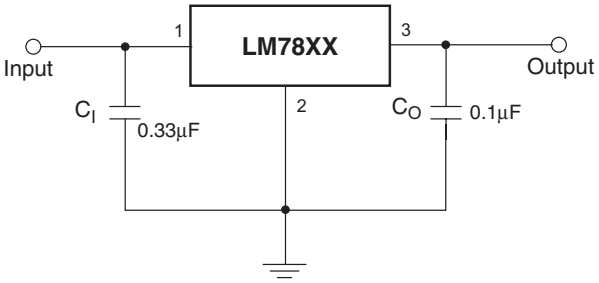


Figura 17 - Montagem Regulador LM7815 [34].

O regulador LD1085V50 tem um V_o de +5V ± 0.1V com um I_o até 3A. Os +5V são utilizados no arrefecimento da CCD da Hamamatsu. A montagem utilizada é a representada na imagem [35]:

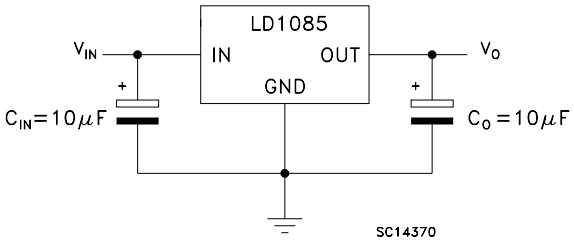


Figura 18 - Montagem LD1085V50 [35].

O conversor IE1224S-H é conversor DC-DC que fornece uma tensão de saída de +24V a partir de uma tensão de +12V (obtida a partir do regulador 7812CV). A corrente máxima na saída é de 42mA o que é mais do que suficiente para as necessidades da CCD Hamamatsu [36].

O conversor PT78NR115S cria uma tensão negativa a partir de uma tensão positiva. A partir dos +12V do regulador 7812, é obtida uma saída de -15V com uma corrente máxima de 300mA [37]. A montagem realizada foi a da imagem seguinte.

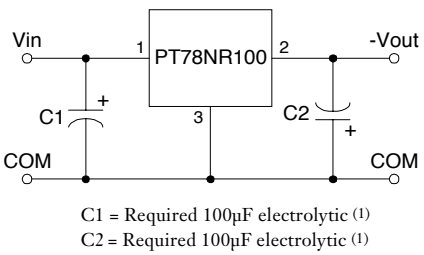


Figura 19 - Montagem do Conversor Pt78nr115s [37].

Para finalizar, na alimentação do ADC AD7680 foram necessários +5V. Para a alimentação do ADC, apesar de poder ser variável, é recomendado a utilização de uma tensão de referência para obter uma gama de entrada no ADC fixa (mais à frente será explicado) utilizando para tal o componente REF195 da *Analog Devices* [38]. Foi utilizado este componente alimentado a partir dos +12V do regulador 7812, o REF195 fornece uma tensão de saída de +5V ± 0.002V com uma corrente até 20mA [39]. A montagem utilizada foi a seguinte:

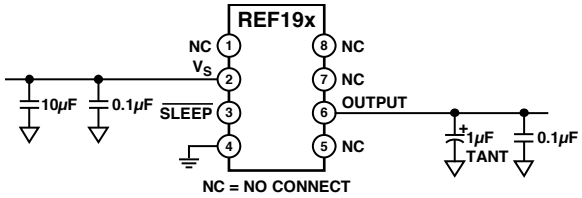


Figura 20 -Montagem para o REF195 [39].

Deste modo foi possível criar um sistema de alimentação para os três componentes enumerados, a partir de um transformador portátil.

4.1.5 Conversão Analógico-digital – ADC

O ADC selecionado no desenvolvimento do sistema de aquisição foi o AD7680 da *Analog Devices*. Este ADC tem como características principais [38]:

- Resolução de 16-bit;
- Taxa de amostragem: 100ksps (*samples per second*);
- Comunicação SPI (*Serial Peripheral Interface*);
- Tensão de alimentação de 2.5V a 5.5V;
- Tensão de referência obtida internamente a partir da alimentação;

Este ADC ao possuir uma resolução de 16-bit permite obter uma gama dinâmica de 2^{16}-1 (65535) na sua saída. Para o sistema desenvolvido, sabendo que a gama dinâmica da CCD da Hamamatsu é de 24000, a resolução do ADC é assim mais do que suficiente.

A alimentação deste ADC (V_{DD}) é variável podendo ir de 2.5V a 5.5V. No entanto a entrada analógica do ADC (V_{in}) é dependente da alimentação, correspondendo a uma gama de 0v a +V_{DD}. Assim se explica a razão de no sistema de alimentação se ter utilizado o REF195 recomendado, de maneira a obter uma gama de entrada fixa de 0v a 5V, como exemplificado na imagem seguinte [38].

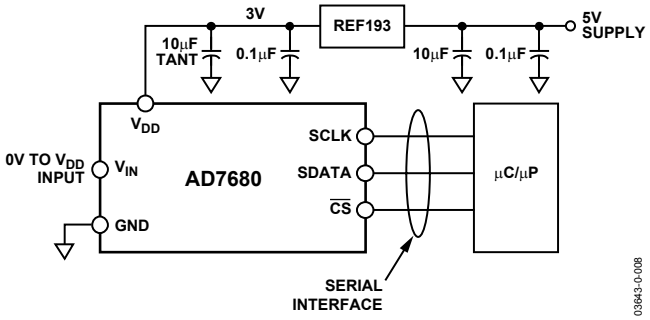


Figura 21 - Montagem do ADC com o REF195 [38].

Para o funcionamento deste ADC, para além da alimentação e da entrada analógica devidamente ligada ao correspondente pino, é necessária estabelecer a interface de comunicação SPI, através da qual se obterá o sinal digitalizado [38].

A interface SPI é realizada entre o ADC e o microcontrolador PIC32. São necessárias estabelecer 3 ligações físicas entre os dois componentes: CS, SCLK e SDATA.

Este ADC possui dois modos de funcionamento: *Normal Mode* e *Power-Down Mode*. O *Normal Mode* não tem questões de poupança energética associada e é o modo no qual se obtém uma maior taxa de amostragem, daí ter sido este modo de funcionamento o selecionado.

O CS (*Chip Select*) é um sinal lógico de entrada no ADC, sendo o inicio da amostragem, do sinal de entrada V_{IN}, realizada no baixar do CS. É igualmente no baixar do CS que a conversão se inicia.

O SCLK é um sinal de clock de entrada no ADC sendo o clock utilizado para o processo de conversão do ADC. O SCLK não necessita de ser contínuo podendo só ser ativo quando se baixa o CS. A taxa de amostragem depende da frequência do SCLK, para obter a taxa máxima de amostragem a frequência do SCLK é a frequência máxima deste admitida pelo ADC que é de 2.5MHz.

O SDATA é um sinal lógico de saída do ADC no qual vai o resultado da conversão do ADC. A conversão depende obviamente do CS e do SCLK, sendo iniciada no baixar do CS. Para atingir a taxa de amostragem de 100ksps são necessários 24 ciclos de clock antes de elevar novamente o CS. É possível obter o resultado de uma conversão apenas com 20 ciclos de clock, atingindo deste modo taxas superiores a 100ksps. Caso se utilizem menos de 20 ciclos de clock, o resultado da conversão não é válido.

O SDATA envia primeiro quatro zeros iniciais, contando o primeiro zero já durante o baixar do CS. Após os quatro zeros são enviados os 16 bits da conversão começando no bit mais significativo (MSB). No final são enviados quatro zeros finais antes do ADC entrar em suspensão.

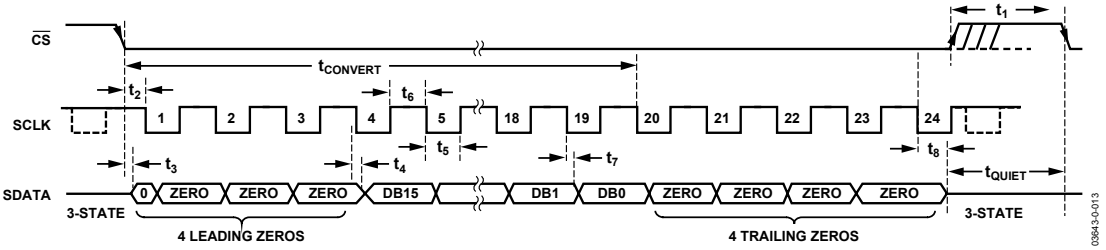


Figura 22 - Diagrama dos sinais do ADC [38].

A figura exemplifica todo o processo de conversão sendo apresentadas as três linhas da comunicação SPI e a relação entre os diferentes sinais. Na imagem é possível verificar como se processa uma conversão válida no ADC.

Todo este processo será controlado pelo PIC32, consequentemente, a taxa de amostragem será da responsabilidade do PIC32 e da frequência a que o SCLK é enviado, assim como do tempo entre sucessivos baixar de CS.

Na imagem seguinte é apresentado todo o hardware no qual esta incluído o sistema de alimentação, placas de desenvolvimento e ADC.

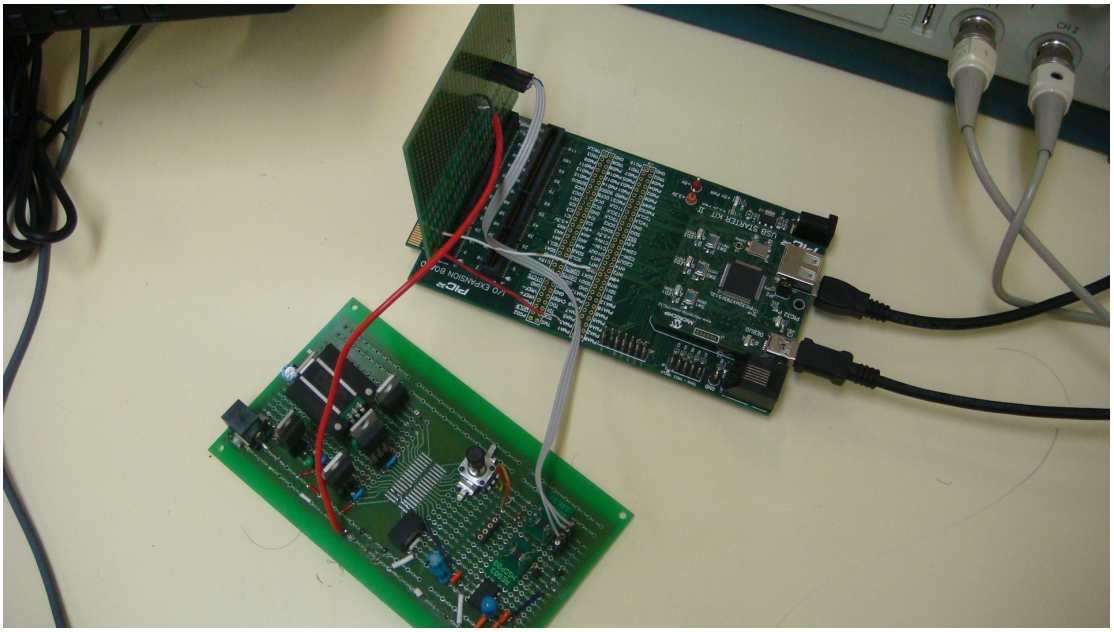


Figura 23 - Montagem final do Hardware.

Na secção seguinte, sobre o *firmware*, na comunicação SPI será demonstrado o trabalho desenvolvido acerca desta ligação entre PIC32 e o ADC.

4.2 *Firmware*

O desenvolvimento de *firmware* para controlo do microcontrolador PIC32 foi realizado por três etapas distintas. Em primeiro lugar foi necessário desenvolver *firmware* para obter um meio de comunicação entre o microcontrolador e a aplicação de interface do utilizador, de seguida procedeu-se ao desenvolvimento do protocolo de comunicação SPI para obter uma interface de comunicação entre o microcontrolador e o ADC. Finalmente foi desenvolvido *firmware* para gerar os sinais de controlo da CCD Hamamatsu (Start e clock) assim como desenvolver a receção de sinais de feedback da CCD (*Trigger* e *End-of-Scan*).

4.2.1 Comunicação USB

A comunicação USB é um meio de comunicação que foi desenhado originalmente partindo de três objetivos interrelacionados: Conexão entre telefone e PC (as

industrias de comunicações e computacionais desenvolvem-se independentemente uma da outra, o USB fornecia um meio de ligação), fácil utilização (falta de uniformidade na ligação de periféricos, surge a necessidade de um barramento comum a todos os periféricos com padrão *plug-and-play*) e expansão de portas (a existência de limitação no número de portas para conexão com periféricos, assim como a falta de comunicação bidirecional e baixo custo) [40].

O USB tem vindo a ser desenvolvido ao longo dos tempos, partindo de uma versão inicial USB 1.0, atualmente a versão mais comummente utilizada é o USB 2.0. A versão USB 3.0 já está lançada no mercado e começa agora a ser difundida. A principal diferença entre versões é a velocidade de transferência [41].

As velocidade de transferência (até ao USB 2.0) são *High Speed* (480Mbits/s), *Full Speed* (12Mbits/s) e *Low Speed* (1.5Mbits/s) [41].

A ligação por USB é controlada por um *Host* que é encarregue de todas as transações na linha entre este e o *Slave* (periférico). Só pode existir um *Host* por cada ligação USB e até 127 *Slaves* numa ligação ramificada. Atualmente existe o *USB On-The-Go* no qual um periférico pode atuar como *Host* ou como *Slave* [41].

O USB é composto por várias camadas de protocolos, no entanto somente são necessárias definir as camadas de alto nível. Cada transação por USB é composta por um *Token Packet*, um opcional *Data Packet* e um *Status packet*. O *Token Packet*, que é o inicial, é gerado pelo *Host* no qual é definido o endereço (dispositivo a ligar) assim como os *endpoint* nos quais se define o tipo de transferência: podem ser *Control, Interrrupt, Isochronous* ou *Bulk* [41].

No projeto o tipo de transferência pretendido foi uma *Bulk Transfer* que permite a transferência de grandes quantidades de dados. É um tipo de transferência que permite correção de erros, através de mecanismos próprios de correção de erros. Existe garantia de entrega de dados, no entanto não garante uma largura de banda mínima (não há uma taxa mínima à qual os dados são transferidos). As velocidades existentes para *Bulk Transfer* são em *High* ou *Full Speed*.

No desenvolvimento de *firmware* para comunicação USB, devido à elevada complexidade que este processo requer assim como uma elevada experiência na programação de microcontroladores, partiu-se de um exemplo fornecido nas librarias da *Microchip* (*Microchip Application Libraries* v2010-10-19) [42] denominada *USB Device – LibUSB – Generic Driver Demo*.

A LibUSB é uma libraria *open source* que permite o acesso a dispositivos USB em diferentes sistemas operativos. Esta libraria fornece uma API (*Application Programming Interface*) que estabelece rotinas de comunicação USB entre software e dispositivo de maneira ao utilizador necessitar somente de chamar certas funções no desenvolvimento de software para obter transferências de dados por USB.

A utilização deste exemplo permitiu obter uma comunicação USB entre *Host* (PC) e dispositivo (PIC). O exemplo utilizado fornecia um projeto em *Visual Studio C++,* o que permitiu um rápido desenvolvimento da interface gráfica com o utilizador. O exemplo fornecia também o controlador de dispositivo (*Device Driver*), e *firmware* preparado para comunicar com o PC (*Host*).

O tipo de comunicação definido possuía um *endpoint* de saída e outro de receção do tipo *bulk* e estava definido para transferências em *Full-Speed*. Em *Full-Speed*, as transferências *bulk* possuem até 64 bytes por cada pacote de dados [41].

Em relação à programação de *firmware*, na função main do *firmware* o sistema é iniciado pela função *InitializeSystem()* que ativa as portas como digitais e executa a função *USBDeviceInit()* para inicialização do módulo USB. Seguidamente a função entra num ciclo infinito *while(1)* onde executa a função *ProcessIO()*.

O *ProcessIO* tem implementada a função *USBGenRead* para leitura do pacote de dados enviado pelo software. Esta função tem como entrada um *array* de 64 bytes (*chars*) denominado INPacket. Este pacote de dados é recebido e, através do valor do primeiro byte (*char*) de cada pacote, é executada determinada função.

Para enviar dados por USB para o software a função utilizada é a *USBGenWrite*. Esta função tem como entrada, analogamente à função de receção, um *array*

denominado OUTPacket. Deste modo, é possível obter uma comunicação bidirecional entre microcontrolador (*Device*) e PC (*Host*).

Foi criada uma função *CheckConnection*() para verificação da comunicação USB. Esta altera um pacote de dados enviado pelo software. Se foi devidamente alterado, o software reconhece e indica que a comunicação USB está a funcionar corretamente.

4.2.2 Comunicação SPI

Após ser obtida a comunicação USB entre o firmware e o PC, o passo seguinte foi desenvolver um meio de obter um sinal digitalizado pelo ADC AD7680 e transferir o resultado da conversão para o software.

O ADC AD7680 suporta a comunicação SPI, um dos tipos de comunicação em série suportada igualmente pelo microcontrolador.

O módulo SPI possui diferentes configurações dependendo de diferentes fatores a ter em consideração:

- Modo *Normal* ou *Framed*

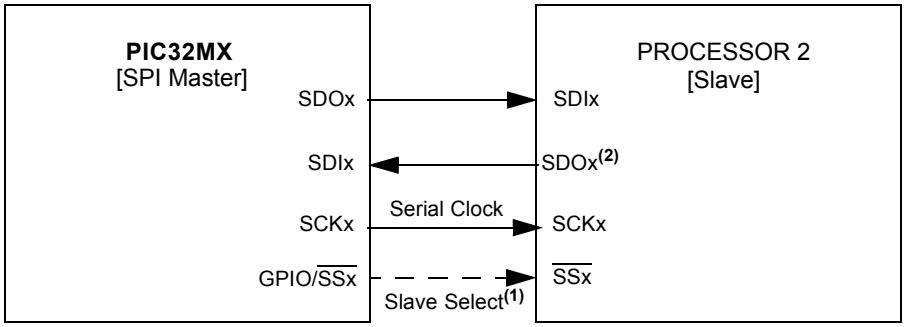
No modo *Normal* o dispositivo *Master* (Mestre – controla a comunicação SPI) é o responsável pelo envio dos clocks para o *Slave* (Escravo – responde aos sinais do *Master*) para sincronismo da transferência. É opcional o uso de um pino SS (*Slave Select* – para escolha do *Slave*). O envio de clocks é limitado e só se processa no momento da transferência de dados [43].

No modo *Framed* os clocks podem ser enviados ou pelo Master ou pelo Slave, no entanto tem que ser utilizado o pino SS. Neste tipo de transferência, caso o Master seja o PIC, os clocks são enviados continuamente, enquanto que no modo *Normal* só são enviados quando é dada ordem pelo Master [43].

O tipo selecionado para o projeto foi o *Normal Mode*.

- Modo Master ou Slave

O módulo SPI do microcontrolador PIC32 permite escolher se o PIC atua como Master ou como Slave. Neste caso o PIC irá atuar como Master e coordenar toda a transferência de dados por SPI. Na seguinte imagem temos a configuração utilizada em relação aos dois últimos modos na qual o *PROCESSOR 2* é o ADC [43].



Note 1: In Normal mode, the usage of the Slave Select pin ($\overline{SSx}$) is optional.
2: Control of the SDO pin can be disabled for Receive-Only modes.

Figura 24 - Comunicação SPI utilizada [43].

O pino SS foi desativado assim como o pino SDO (pino de saída de dados do PIC). Como já referido, a comunicação SPI é bidirecional, mas no caso em que se pretenda só receber dados do Slave, o pino de saída do Master pode ser inativo, como é o caso.

- Polaridade do Clock

Foi necessário escolher a polaridade do clock atendendo à configuração SPI do ADC demonstrada na figura 38. Existem quatro tipos diferentes de polaridade de clock [43].

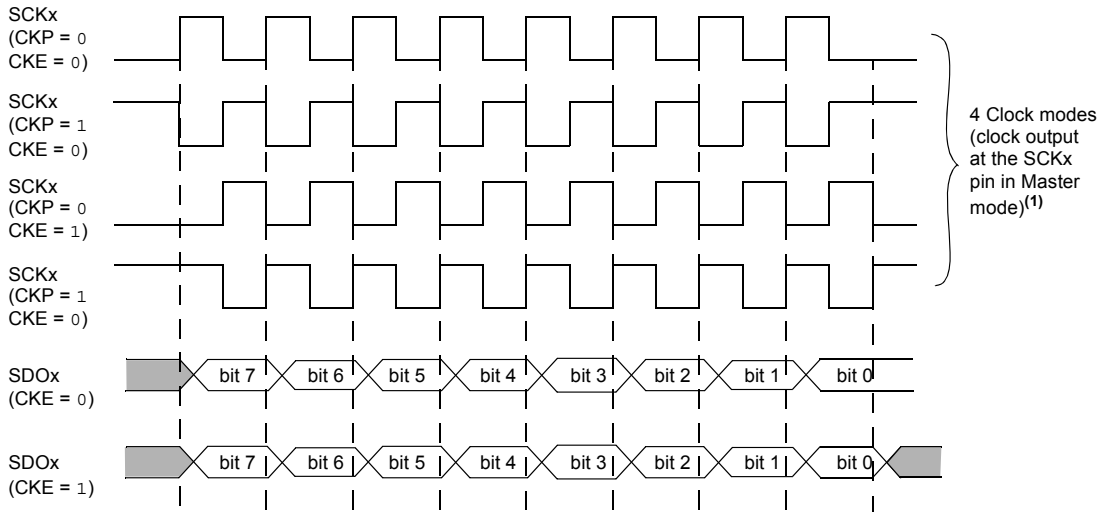


Figura 25 - 4 modos de polaridade de clock [43].

Assim sendo, a polaridade correta será começando alto, de tal modo que na configuração dos bits, o bit do *CKP* será 1 e o bit do *CKE* será 0. Deste modo foi possível obter uma correta contagem dos clocks enviados.

- Quantidade de Dados (*Data Width*)

Outro ponto a ter cuidado na definição da comunicação SPI foi a quantidade de dados por cada transação. Os *PIC32MX* permitem realizar esta operação em 8-bit, 16-bit ou 32-bit. Todas as transmissões ou receções são efetuadas com o mesmo número de bits selecionado.

Um ponto a destacar neste tipo de transferência é que o registo SPI de dados de receção/transmissão possuem internamente dois registos separados: o *buffer* de transmissão (SPIxTXB) e o de receção (SPIxRXB). No entanto eles possuem o mesmo endereço SPIxBUF. Assim sendo, para efetuar uma transação bidirecional de dados é necessário escrever algo no endereço SPIxBUF e posteriormente aceder ao endereço para obter os dados recebidos [43].

Para comunicar com o ADC foi utilizado um *Data Width* de 32-bit, pois o ADC necessita de pelo menos 20 clocks para uma conversão válida (4 zeros iniciais mais 16 bit de informação).

- Taxa de Transmissão (*Baud Rate*)

Para terminar, um ponto essencial a ter em conta foi a taxa a que a comunicação SPI se processa.

A taxa de amostragem máxima a que o ADC processa é de 2.5MHz, e como anteriormente referido, é a frequência de clock (SCLK) enviada para o ADC que determina a taxa de amostragem. A frequência de clock da comunicação SPI (F_{SCK}) relaciona-se com o registo SPIxBRG tendo em conta a frequência do *Peripheral Bus Clock* - F_{PB} (clock periférico que está relacionado com a frequência do sistema) [43]:

$$F_{SCK} = \frac{F_{PB}}{2 \cdot (SPIxBRG + 1)}$$

Sabendo que o F_{PB} é de 60MHz e que o F_{SCK} pretendido é de 2.5MHz obtém-se o valor de 11 para o SPIxBRG.

Após definição dos aspetos acima descritos na configuração SPI e da ativação do módulo SPI, o *firmware* fica preparado para realizar a comunicação com o ADC.

O ADC necessita de um sinal CS para iniciar o processo de conversão. Este é dado por um pino de saída do PIC. A instrução seguinte é escrever no *buffer* do SPI um inteiro. Apesar de o pino de saída da comunicação SPI estar desativado, o processo de comunicação SPI é bidirecional e processa-se deste modo: escrita no *buffer* seguido de leitura do *buffer* para obter o que é recebido, uma vez que partilham o mesmo endereço SPIxBUF. Os 32 clocks são enviados e o ADC envia o resultado da conversão. É lido o *buffer* com um inteiro que contém o resultado da conversão.

Uma vez que tenho o resultado (16 bits) no meio de um inteiro (32 bits) é necessário fazer um *shift* de bits para obter o resultado da conversão (16-bit) em dois *chars* separados, para envio no OUTPacket. Como o primeiro zero enviado pelo ADC conta no momento em que o CS vai abaixo, este bit não entra no inteiro com o resultado da conversão. Automaticamente, o *buffer* é preenchido com 3 bits a zero iniciais, seguido de 16 bits da conversão e 13 bits a zero finais perfazendo o 32 bits do *buffer*.

Para fazer a separação da conversão dos zeros, faz-se um *shift* à direita de 13 bits seguido de leitura do inteiro com um *short int* (16 bits) onde fica o resultado da conversão.

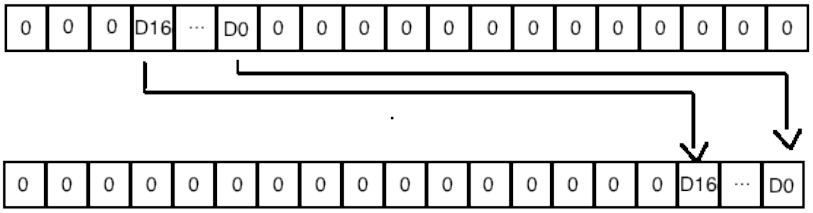


Figura 26 - Shift de bits necessário para obter resultado da conversão isolado.

Posteriormente é dividido o *short int* em dois *chars* para envio no OUTPacket.

Para fazer uma sequência de conversões, foi utilizado um temporizador para obter sucessivas conversões a uma frequência fixa.

Assim se processa a interface do microcontrolador com o ADC através da comunicação SPI e envio dos dados recebidos do ADC no lado do *firmware*.

4.2.3 Sinais de Controlo

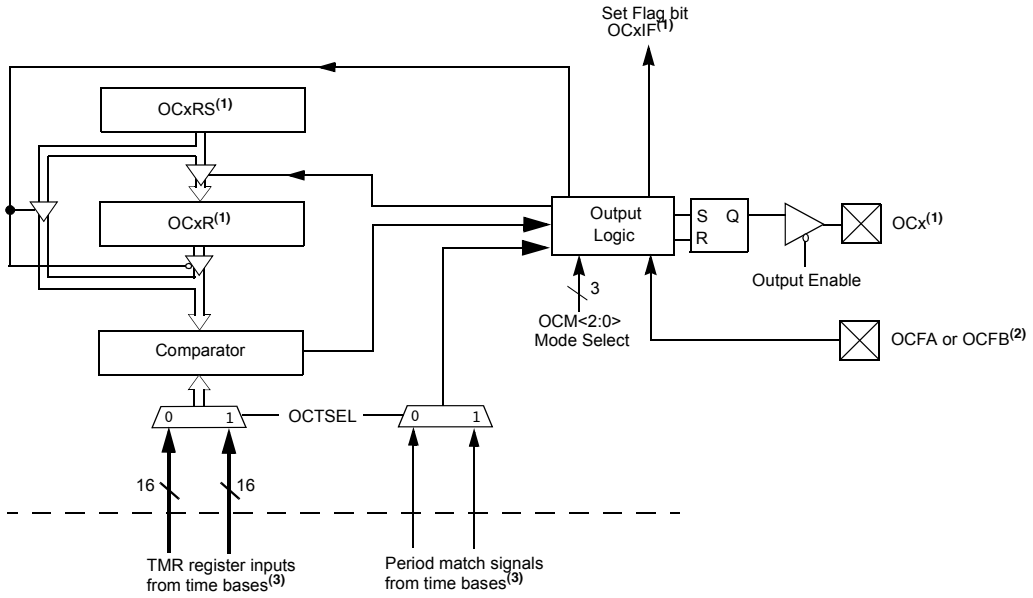
No desenvolvimento do *firmware* foi necessário gerar sinais de controlo para a CCD Hamamatsu: um start e um clock.

Para efetuar este processo foi utilizada uma ferramenta dos *PIC32MX* designada por *Output Compare* (OC). O módulo OC é utilizado para gerar um único impulso ou um conjunto de impulsos tendo como base um temporizador do microcontrolador. Este módulo server igualmente para fazer modulação de largura de impulsos (PWM – *Pulse Width Modulation*) [44].

O módulo OC para além de ter um temporizador como base permite ainda obter interrupções. O módulo pode ser definido em 16-bit ou 32-bit. Para este módulo, além de ser necessário definir o registos de controlo (definição da OC), é necessário

definir outros dois registos: o OCxR (registo do comparador) e OCxRS (registo do comparador secundário).

No diagrama de blocos do módulo é possível verificar os dois registos OCxR e OCxRS assim como um temporizador na entrada. A saída do impulso é dada no pino OCx.



Note 1: Where 'x' is shown, reference is made to the registers associated with the respective output compare channels 1 through 5.
2: OCFA pin controls OC1-OC4 channels. OCFB pin controls OC5 channels.
3: Each output compare channel can use one of two selectable 16-bit time bases or a single 32-bit time base. Refer to the device data sheet for the time bases associated with the module.

Figura 27 - Diagrama de blocos do módulo Output Compare [44].

Os sinais de controlo necessários ao funcionamento da CCD, start e clock, são portanto um único impulso (start) e um conjunto de 3216 clocks necessários para cada aquisição. Este dois sinais tem que estar devidamente sincronizados [26].

Para obter uma sincronização dos dois sinais é necessário utilizar o mesmo temporizador (*Timer* 3 neste caso). Para definir os dois OC foi utilizada uma função da libraria do microcontrolador:

OpenOCx(unsigned int config , unsigned int value1 , unsigned int value2);

O *config* contém os parâmetros relativos ao registo OCxCON no qual estão as definições da OC, o *value1* é o valor do registo secundário OCxRS e o *value2* é o valor do registo principal OCxR.

Para o sinal de start foi utilizada um OC de impulso único (*OC_SINGLE_PULSE*) com os dois registos OCxR e OCxRS a zero o que permite obter um impulso durante uma contagem do temporizador.

Para o sinal de clock foi utilizada um OC de impulsos contínuos (*OC_CONTINUE_PULSE*) com o registo primário (OCxR) a zero que irá permitir a sincronização com o start. O registo secundário tem o valor de metade do temporizador de maneira a obter o período de um clock enquanto o impulso do start está alto.

Deste modo é possível obter um sinal em que a largura do impulso do start é superior a um período de clock como requisito da Câmara CCD. Na imagem seguinte é possível ver o esquema de impulsos clock e start gerados pelo PIC.

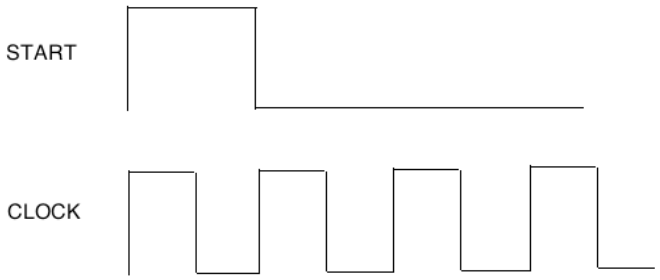


Figura 28 - Esquema de impulsos Start e Clock gerados pelo PIC32.

Outras alternativas para gerar os sinais de controlo foram experimentadas, no entanto, a utilização do módulo de OC foi o que obteve melhor sincronismo nos sinais.

4.3 *Software*

Para o projeto foi necessário desenvolver uma GUI para obtermos um meio de interação entre o utilizador e o hardware.

A GUI desenvolvida tem que criar um ambiente integrado para programar, visualizar e processar as aquisições, e numa posterior fase, efetuar a função de osciloscópio digital.

Seguidamente será apresentado o trabalho realizado para obter essa interface gráfica, como se obteve a comunicação com o hardware, o processamento dos dados enviados e recebidos e como foi obtida a representação gráfica da imagem linear.

4.3.1 *Visual Studio C++ Windows Forms*

O desenvolvimento de software para interface gráfica com o utilizador foi realizado em *Visual Studio C++*. A escolha desta ferramenta de programação para o desenvolvimento do software partiu do exemplo utilizado na comunicação USB *LibUSB*. O exemplo utilizado já possuía uma GUI desenvolvida nesta ferramenta, consequentemente o trabalho desenvolvido no software à posteriori foi realizado a partir do exemplo fornecido.

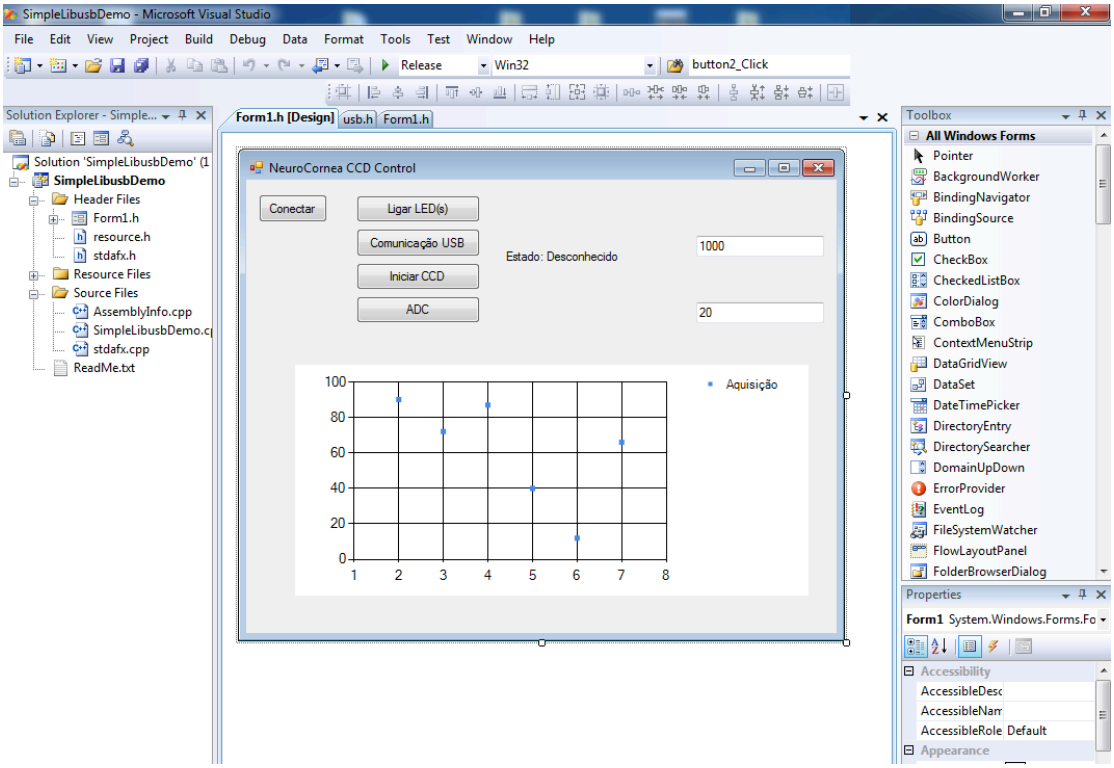


Figura 29 - Ambiente de desenvolvimento da GUI em Visual Studio C++.

Para criação de diferentes objetos existe uma *toolboox* com múltiplas ferramentas. Os objetos são introduzidos e automaticamente é gerado um código que podemos personalizar para obter determinada ação ao interagir com esses objetos.

4.3.2 Receção de Dados e Armazenamento

A comunicação por USB implementada a partir do exemplo inicia-se no botão conectar. Ao conectar é iniciado o módulo USB com a função *usb_init(),* a seguir são encontrados todos os pontos de conexão assim como todos os dispositivos conectados com as funções *usb_find_busses()* e *usb_find_devices()* respetivamente.

O ponto seguinte passa por encontrar o dispositivo por VID/PID (*Vendor e Product Identification*). Após identificação do dispositivo, este é aberto, colocado ativo e é requerida uma interface com o sistema operativo.

A partir deste ponto a comunicação fica ativa e através de funções semelhantes às usadas no firmware, é possível obter uma comunicação bidirecional entre dispositivo e computador.

As funções de envios de dados é dada por:

usb_bulk_write(MyLibusbDeviceHandle, 0x01, &OutputPacketBuffer[0], 64, 5000)

O *MyLibusbDeviceHandle* está relacionado com o aceder à ligação USB; o *0x01* é o número do endpoint; *&OutputPacketBuffer[0]* é o *char* inicial do *array* de *chars* (OutputPacketBuffer) a ser enviado; o valor de *64* é o número de *chars* a ser enviado; o valor *5000* é o valor limite de tempo para a comunicação se processar (5 segundos).

A função de receção de dados é análoga à de envio e é dada por:

usb_bulk_read(MyLibusbDeviceHandle, 0x81, &InputPacketBuffer[0], 64, 5000).

No desenvolvimento da GUI foram criados outros botões, para além do botão conectar já referenciado, com diferentes finalidades.

Em relação à receção de dados da conversão do ADC, foi criado um botão no qual é dada ordem ao *firmware* para obter um conjunto de 512 conversões seguidas do ADC. Como a comunicação estabelecida não suporta mais do que 64 bytes (64 *chars*) por pacote de transferência, as 512 conversões tem que ser separadas por pacotes de dados.

O resultado de uma conversão ocupa 2 bytes, automaticamente por cada pacote de transferência são enviadas 32 conversões, terão que ser enviados 16 pacotes de dados para obter o resultado das 512 conversões (32 x 16).

No momento em que é recebido um pacote de dados é necessário concatenar os bytes 2 a 2 para obter o resultado das conversões num só inteiro. Assim sendo, o primeiro byte, que contem os bits menos significativos da conversão, é guardado numa variável *Data0a8*, enquanto que o segundo byte, que contem os bits mais significativos da conversão, é guardado numa variável *Data8a16*. À segunda variável

é necessário fazer um *shift* agora à esquerda de 8 bits (fica na variável *Data8a16s*), para posterior concatenação das duas variáveis de maneira a obter o resultado na variável *Data*:

Data = (Data0a8|Data8a16s)

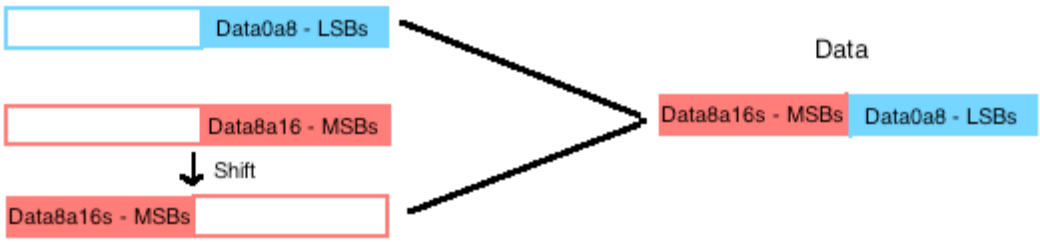


Figura 30 - Esquema de concatenação de variáveis para obter resultado final da conversão.

Deste modo é possível obter todos os resultados das conversões que vão sendo guardados para um ficheiro de texto através da função:

fprintf(pFile,"%d \n",Data)

Deste modo o resultado das 512 conversões são guardados num ficheiro. O tempo necessário para realizar cada aquisição é de 10.24 ms uma vez que cada conversão é obtida em 20 μs.

4.3.3 Amostragem de Dados

Para fazer a representação gráfica dos dados foi utilizado um *Add-on* do *Microssoft Visual Studio 2008*, o *Microssoft Chart Controls*. Este *Add-on* é um conjunto de controlos que torna possível a apresentação de gráficos em *Windows Forms* [45].

Esta ferramenta permite facilmente construir gráficos a partir da *toolboox* do projeto em *Windows Forms*. Permite a introdução de legendas, títulos, séries de dados, múltiplos tipos de gráficos, manipulação de dados, entre outras possibilidades [45].

Partindo desta ferramenta, foi introduzido um gráfico de dispersão ao qual se associou uma série denominada "*series1*".

Para amostragem de dados no gráfico tem que se introduzir obrigatoriamente dois *arrays* correspondentes aos valores X e Y:

ArrayList ^ xvals = gcnew ArrayList(); ArrayList ^ yvals = gcnew ArrayList();

Depois só temos que introduzir os valores pretendidos para cada um dos *arrays*, tendo que ser introduzidos no mesmo número senão ocorrerá erro na execução do programa. Para introduzir valores basta utilizar o comando *xvals->Add(valor*), neste caso no *arrays* do X.

Para realizar a amostragem temos que ligar os valores X e Y à série ("*series1*" neste caso) utilizando a função:

chart1->Series["Aquisição"]->Points->DataBindXY(xvals, yvals);

Deste modo é apresentado graficamente os valores introduzidos. Este conjunto de passos, para representação gráfica, podem ser introduzidos em qualquer evento que ocorra no programa.

Para o software desenvolvido, a apresentação gráfica de dados ocorreu em dois eventos (dois clicar de botões). No botão "ADC" é apresentado graficamente o valor de 512 conversões sucessivas a uma taxa fixa de aquisição. No botão "*Iniciar CCD*" a apresentação gráfica obtida, são de valores de conversão obtidos quando um conjunto de sinais *trigger* (sinal enviado pela CCD Hamamatsu – 512 *triggers* correspondentes aos 512 pixeis da CCD) são enviados para o microcontrolador.

5. Resultados

Após a realização de todo o trabalho experimental descrito no ultimo capítulo foi possível obter a digitalização de qualquer sinal analógico utilizando o botão correspondente ao ADC.

A taxa máxima de amostragem à qual o ADC pode funcionar é de 100ksps. No entanto devido ao facto de ser utilizada uma comunicação SPI de 32 bits, automaticamente são gerados 32 clocks, quando eram somente necessários 24 (para obter uma taxa de 100ksps). Assim sendo, vai haver pelo menos um decréscimo de 25% o que corresponderá a 75ksps no máximo.

Para obter uma periodicidade na amostragem das 512 conversões foi utilizado o módulo de OC, anteriormente já descrito para gerar os sinais de controlo. A utilização do módulo OC, neste caso, vai servir para simular o CS que o ADC necessita. Outra maneira igualmente testada para obtenção de periodicidade na amostragem foi a utilização de um temporizador.

A utilização da OC vai permitir obter um tempo fixo entre descidas do pino CS (aquisições), sendo agora a periodicidade da OC a responsável pela taxa de amostragem. Foi utilizada uma OC em que o tempo de duração é de 20 μs o que dá uma taxa de amostragem de 50ksps. Posteriormente utilizando a mesma taxa de amostragem na CCD, a taxa de 50ksps é suficiente uma vez que a gama dinâmica desta é de 24k.

Realizando cada conversão em 20 μs, o tempo de aquisição de 512 conversões será realizado em 10.24 ms. No entanto, entre envio de pacotes de dados por USB foi implementado um atraso de 1 ms para funcionar corretamente. São enviados os dados em 16 pacotes o que significa um atraso de 16 ms adicionais. Assim sendo, cada 512 conversões são realizadas em 26.24 ms (valor teórico, na prática não foi medido).

Deste modo o resultado final da ação do botão ADC no programa desenvolvido é um conjunto de 512 conversões analógico-digitais de 16 bits de resolução, a uma taxa de

50ksps, para uma entrada analógica de 0v a 5v. Na imagem seguinte é demonstrado um sinal sinusoidal obtido com o sistema desenvolvido, a partir de um sinal dado por um gerador de sinais.

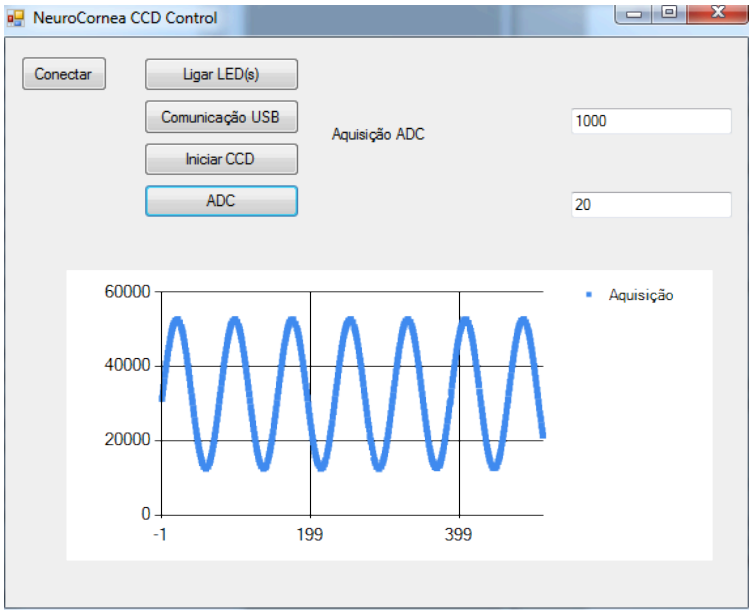


Figura 31 - Aquisição de um sinal sinusoidal.

Não foi possível obter ainda a aquisição em modo contínuo porque no momento final de desenvolvimento em que se procedia a este ajuste, a aquisição contínua estava a ser realizada implementando ciclos contínuos de programação dentro do evento de clicar no botão ADC no *Microssoft Visual Studio C++*. Esta solução não é possível, para contornar o problema terá que ser usado um modo diferente de programação no qual terão que ser usados dois botões no qual um inicia e realiza a amostragem enquanto que o outro finaliza, emitindo uma *flag* de interrupção para o processo de aquisição.

Em relação ao botão Iniciar CCD, o resultado final foi a obtenção de 2 sinais na saída de 2 pinos do microcontrolador OC1 e OC2 (vêr apêndice do esquemático do microcontrolador) respetivamente os sinais de clock e start para a CCD Hamamatsu, assim como a obtenção da resposta a 512 *triggers* fornecidos ao microcontrolador, durante o envio dos clocks.

Foi utilizada uma fonte de sinal para simular o envio de *triggers* para o microcontrolador como resposta da CCD. Por cada *trigger* recebido na CCD o

microcontrolador faz um conjunto de n aquisições (número a ser defino pelo utilizador) do ADC e realiza uma média aritmética deste conjunto de aquisições. É realizada uma contagem de 512 *triggers* que corresponderá aos *triggers* enviados pela CCD. Na entrada do ADC estava um sinal fixo de tensão ajustável por um potenciómetro. A imagem seguinte demostra o resultado obtido desta aquisição.

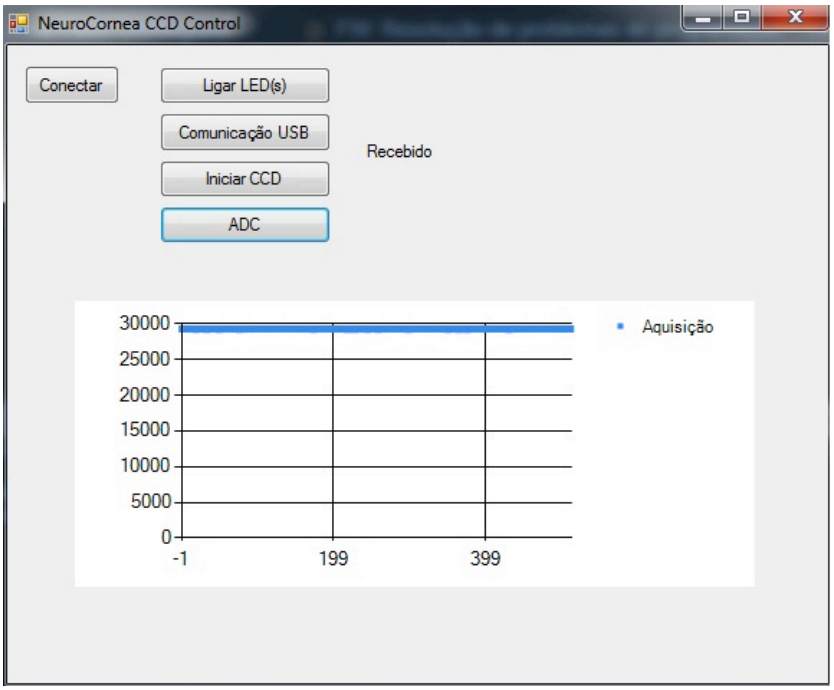


Figura 32 - Resultado obtido de interface CCD com trigger simulado, e amostragem na saída do potenciómetro.

Apesar de não ter sido feita a ligação do sistema à CCD Hamamatsu, com esta simulação do *trigger*, em principio, o sistema está preparado para ligar à câmara e obter desta o resultado de imagens lineares.

Em relação à ligação da Câmara CCD Proxitronic, utilizando o *frame grabber*, foi possível obter imagens provenientes da câmara CCD.

Na saída dos diferentes reguladores e conversores foi possível obter as diferentes tensões de alimentação que todos os componentes necessitam.

6. Conclusão

A utilização da microscopia confocal da córnea como meio de diagnóstico precoce para pacientes que sejam diabéticos e que possam sofrer da condição de neuropatia diabética periférica, é um meio de atingir esse fim de uma maneira não-invasiva.

O desenvolvimento de um módulo confocal permitirá que este meio de diagnóstico seja amplamente utilizado no diagnóstico de neuropatia diabética.

A eletrónica desenvolvida foi um passo para obter essa finalidade. Resumidamente os objetivos seguintes foram atingidos:

- Construção de um sistema de alimentação para o sistema de aquisição e para as câmaras CCD utilizadas;
- Criação de um sistema que faça o interface com a Câmara CCD Hamamatsu C5809;
- Interface entre o sistema desenvolvido e utilizador final através de comunicação USB;

Apesar de não ter sido feita a ligação entre a Câmara CCD da Hamamatsu e o sistema, o funcionamento da câmara foi simulado e testado com bons resultados.

O desenvolvimento do circuito de controlo e leitura para a câmara FFT-CCD da Hamamatsu foi realizado no sentido de manutenção e possibilidade de utilização do PAF com o sistema desenvolvido, atualizando assim o seu hardware.

6.1 Trabalho Futuro

A ligação da Câmara CCD Hamamatsu ao sistema possibilitará verificar se esta funciona corretamente. Com a ligação da câmara será possível fazer testes de linearidade e reprodutibilidade da câmara.

Em relação ao desenvolvimento de *firmware* e software será necessário fazer com que o sistema seja capaz de receber diferentes parâmetros tais como o tempo o

exposição ou o número de aquisições de maneira a tirar todas as potencialidades da câmara CCD, assim como fazer com que o sistema faça aquisições do ADC em modo contínuo.

A construção de todo o sistema num PCB (*Printed Circuit Board*) permitirá obter um sistema mais compacto e funcional, podendo deste modo reduzir eventual ruído de sinal que possa surgir no sistema desenvolvido.

Bibliografia

[1] Observatório da Diabetes. *Diabetes: Factos e Números. Relatório Anual do Observatório Nacional da Diabetes.* 2010.

[2] International Diabetes Federation. *Diabetes In Europe – Towards a European Framework for Diabetes Prevention and Care*. EU Workshop Proceedings, 2004

[3] Centers for Disease Control and Prevention. *National diabetes fact sheet: national estimates and general information on diabetes and prediabetes in the United States, 2011*. Atlanta, GA: U.S. Department of Health and Human Services, Centers for Disease Control and Prevention, 2011.

[4] Fauci, A., Braunwald, E., Kasper, D, Hauser, S, Longo, D., Jameson, J., Loscalzo, J. *Harrison - Manual de Medicina.* 17ª Edição. McGraw-Hill, 2010.

[5] [Internet] http://diabetes.niddk.nih.gov/dm/pubs/neuropathies/ Acedido a 05 de agosto de 2011.

[6] Chamberlin, S.L., Narnins, B. *The Gale Encyclopedia of Neurological Disorders*. Thomson Gale, 2005

[7] Boucek, P. *Advanced Diabetic Neuropathy: A Point of no Return?* Rev Diabet Stud. **3**(3), pp.143-151. 2006

[8] Masters, B. *Noninvasive Diagnostic Techniques in Ophthalmology*, Springer, 1990.

[9] Wnek, G., Bowlin, G. *Encyclopedia of Biomaterials and Biomedical Engineering*. Second edition. Informa Healthcare, 2008.

[10] Vo-Dinh, T. *Biomedical Photonics Handbook.* CRC Press, 2003.

[11] Masters, B., Böhnke, M. *Confocal Microscopy of the human cornea in vivo. International Ophthalmology*. 23: pp.199-206, 2001.

[12] Masters, B. Ch.17 - Three-dimensional Confocal Microscopy Of The Living Human Cornea In: *Handbook of Optics Volume III: Vision and Vision Optics*. Third Edition. McGraw-Hill, 2010.

[13] Ferreira, A., Lamas, J., Gomes, L., Silva, S., Loureiro, L., Domingues, J.P., Silva, J.S., Morgado M. *NeuroCórnea - Diabetic peripheral neuropathy early diagnosis and follow-up through in vivo automatic analysis of corneal nerves morphology.* 2011.

[14] [Internet] http://groups.engin.umd.umich.edu/CIS/course.des/cis400/c/c.html Acedido a 10 de agosto de 2011.

[15] Damas, L. *Linguagem C*. 10ª Edição. LTC Editora, 2007

[16] Horton, I. *Beginning C: From Novice to Professional*. Apress, 2006.

[17] Matic, N. *PIC microcontrollers, for beginners too.* Mansong, 2000.

[18] di Jasio, L. *Programming 32-bit Microcontrollers in C, Explore the PIC 32.* Elsevier, 2008.

[19] Randolph, N., Gardner, D. *Professional Visual Studio 2008*. Wiley Publishing Inc, 2008.

[20] Horton, I. *Ivor Horton's Beginning Visual C++ 2008*. Wiley Publishing Inc, 2008.

[21] Janesick, J. *Scientific Charge-coupled Devices*. SPIE Press, 2001.

[22] [Internet] http://learn.hamamatsu.com/articles/moscapacitor.html Acedido a 20 de agosto de 2011

[23] Morgado, M. *Radiometria e Fotometria.* Slides das Aulas de Intrumentação Optoelectronica, 2008.

[24] Vitor, M. *Fluorómetro Ocular – Upgrade.* Tese de Mestrado, 2009.

[25] Proxitronic. *Proxicam HL5. High Resolution Low Light Camera.*

[26] Hamamatsu Photonics K.K. *FFT-CCD Image Sensor Multichannel Detetor Head, C5809 series, Instruction Manual.* 1995.

[27] Hamamatsu. *Technical Information – Characteristics and use of NMOS linear image sensors.* 2007.

[28] Microchip Technology Inc. *PIC32 USB Starter Kit II User's Guide*. 2009.

[29] [Internet] http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2615&dDocName=en535536 Acedido a 20 de agosto de 2011

[30] [Internet] http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2615&dDocName=en535444 Acedido a 20 de agosto de 2011

[31] [Internet] http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en545660 Acedido a 20 de agosto de 2001

[32] XP Power. *36-100 Watts AED Serieis.* 2007.

[33] ST Microelectronics. *L78xx - L78xxC L78xxAB - L78xxAC Positive Voltage Regulators.* 2010.

[34] Fairchild Semiconductor. *LM78XX/LM78XXA 3-Terminal 1A Positive Voltage Regulator.* 2010.

[35] ST Microelectronics. *LD1085xx 3A low drop positive voltage regulator adjustable and fixed.* 2008.

[36] XP Power. *1Watt IE Series*. 2011.

[37] Texas Instruments. *PT78NR100 Series 1 Amp Plus to Minus Voltage Integrated Switching Regulator.* 2000.

[38] Analog Devices. *AD7680.* 2004.

[39] Analog Devices. *REF 19x Series*.

[40] Hewlett-Packard Company, Intel Corporation, Lucent Technologies Inc, Microsoft Corporation, NEC Corporation, Koninklijke Philips Electronics N.V. *Universal Serial Bus Specification.* 2000.

[41] Peacock, C. *USB in a Nutshell*. Disponivel em: http://www.beyondlogic.org/usbnutshell/usb-in-a-nutshell.pdf 2000.

[42] [Internet] http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2896 Acedido a 22 de agosto de 2011

[43] Microchip Technology Inc. Section 23. Serial Peripheral Interface (SPI). In: *PIC32 Family Reference Manual*. 2009.

[44] Microchip Technology Inc. Section 16. Output Compare. In: *PIC32 Family Reference Manual*. 2009.

[45] [Internet] http://archive.msdn.microsoft.com/mschart Acedido a 24 de agosto de 2011

Anexos

1 - Cálculos de Sensibilidades das Câmaras CCD

- Orca – Flash 2.8

Readout noise: $3e^-$ Ef. Quântica (500nm): 0.66

A_{pixel} = 3.63 (µm) x 3.63 (µm) = $1.32 \times 10^{-11} m^2$ T_{exp} = 0.02 s

n = 9.09 E = 3.29×10^{-18} J $\phi = 1.64 \times 10^{-16}$ W Ee = 1.24×10^{-5} W/m^2

EV = 2.80 mLx

- Orca – R2

Readout noise: $10e^-$ Ef. Quântica (500nm): 0.70

A_{pixel} = 6.45 (µm) x 6.45 (µm) = $4.16 \times 10^{-11} m^2$ T_{exp} = 0.02 s

n = 28.57 E = 1.03×10^{-17} J $\phi = 5.17 \times 10^{-15}$ W Ee = 1.24×10^{-5} W/m^2

EV = 2.80 mLx

- Orca – 03G

Readout noise: $8e^-$ Ef. Quântica (500nm): 0.70

A_{pixel} = 6.45 (µm) x 6.45 (µm) = $4.16 \times 10^{-11} m^2$ T_{exp} = 0.02 s

n = 22.86 E = 8.26×10^{-18} J $\phi = 4.13 \times 10^{-16}$ W Ee = 9.93×10^{-6} W/m^2

EV = 2.24 mLx

- Orca – 05G

Readout noise: $10e^-$ Ef. Quântica (500nm): 0.70

A_{pixel} = 6.45 (µm) x 6.45 (µm) = $4.16 \times 10^{-11} m^2$ T_{exp} = 0.02 s

n = 28.57 E = 1.03×10^{-17} J $\phi = 5.17 \times 10^{-15}$ W Ee = 1.24×10^{-5} W/m^2

EV = 2.80 mLx

- Hamamatsu C9300-221

Readout noise: $20e^-$ Ef. Quântica (500nm): 0.56

A_{pixel} = 7.40 (µm) x 7.40 (µm) = $5.48x10^{-11}m^2$ T_{exp} = 0.02 s

n = 71.43 E = 2.58×10^{-17} J $\phi = 1.29 \times 10^{-15}$ W $Ee = 2.36 \times 10^{-5}$ W/m^2

EV = 5.30 mLx

- Pixelfly Qe

Readout noise: $7e^-$ Ef. Quântica (500nm): 0.62

A_{pixel} = 6.45 (µm) x 6.45 (µm) = $4.16x10^{-11}m^2$ T_{exp} = 0.02 s

n = 22.58 E = 8.16×10^{-18} J $\phi = 4.08 \times 10^{-16}$ W $Ee = 9.81 \times 10^{-6}$ W/m^2

EV = 2.21 mLx

1.2 PIC32 FUNCTIONALITY AND FEATURES

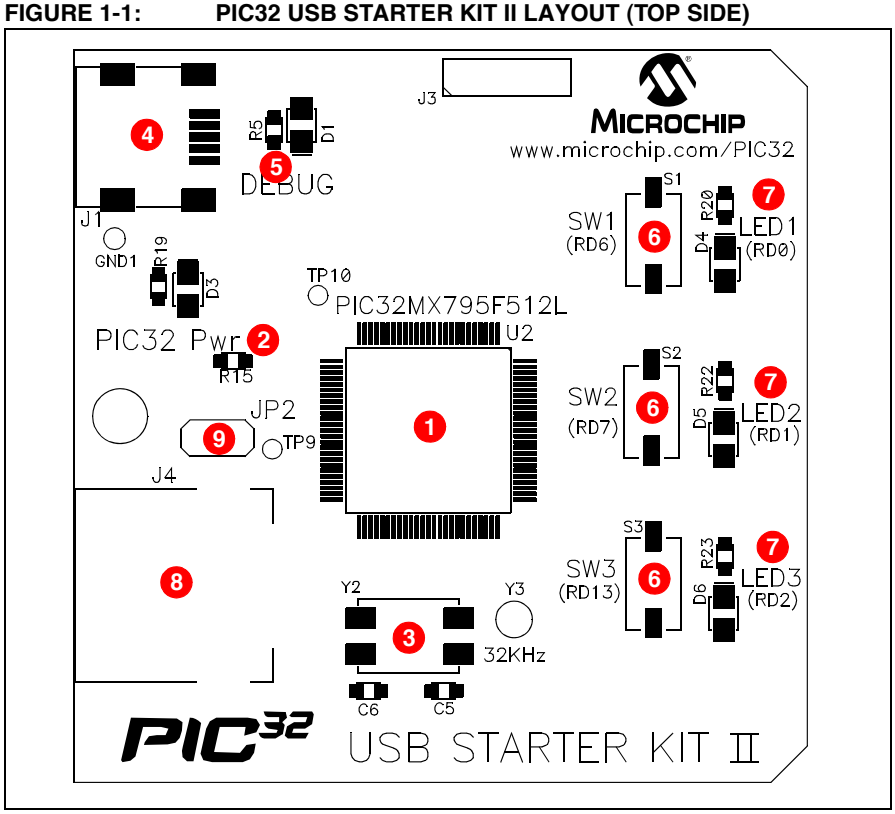
Representations of the layout of the PIC32 USB Starter Kit II are shown in Figure 1-1 and Figure 1-2.

The top assembly of the board includes these key features, as indicated in Figure 1-1:

1. PIC32MX795F512L 32-bit microcontroller.
2. Green power-indicator LED.
3. On-board crystal for precision microcontroller clocking (8 MHz).
4. USB connectivity for on-board debugger communications.
5. Orange debug indicator LED.
6. Three push button switches for user-defined inputs.
7. Three user-defined indicator LEDs.
8. USB Type A receptacle connectivity for PIC32 Host-based applications.
9. HOST mode power jumper.

Note: When running USB device applications, open the jumper JP2 to prevent possibly back-feeding voltage onto the VBUS from one port on the host to another (or from one host to another).

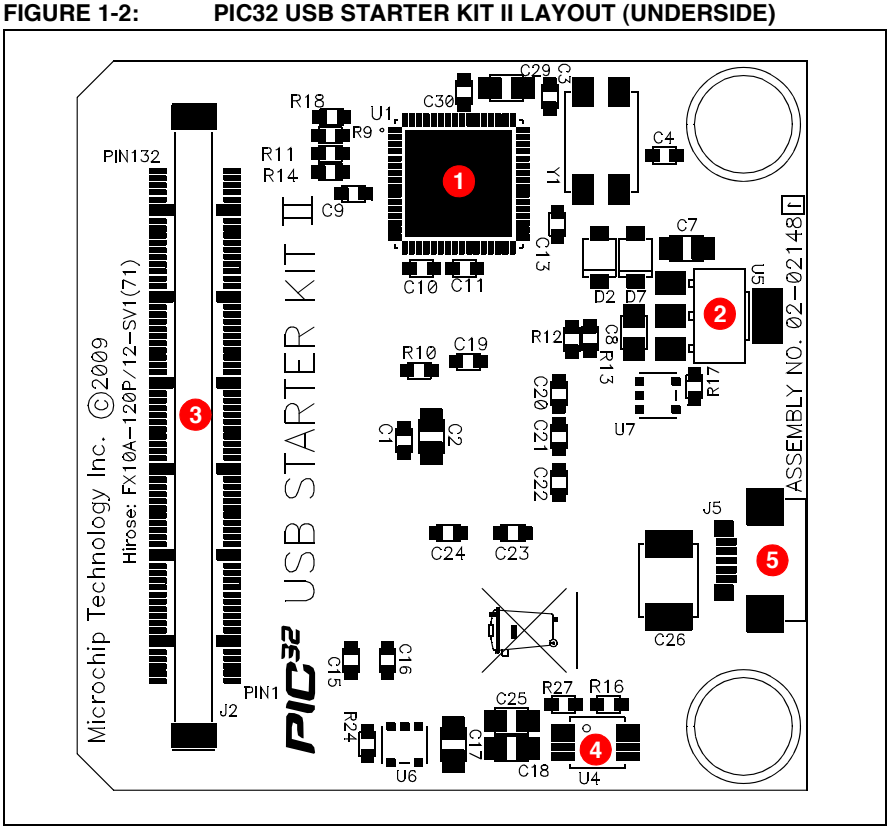
FIGURE 1-1: PIC32 USB STARTER KIT II LAYOUT (TOP SIDE)



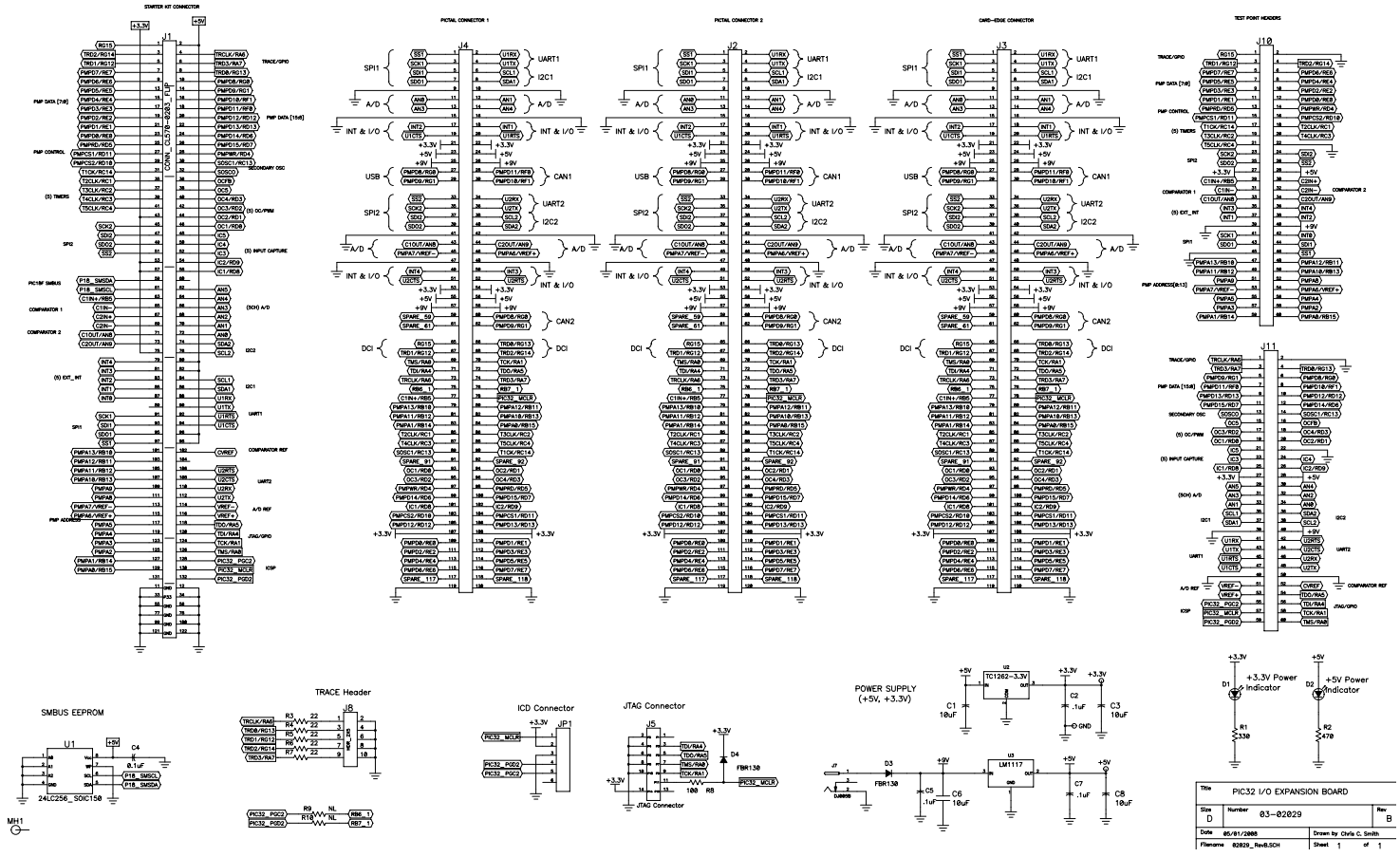
The bottom assembly of the board includes these key features, as indicated in Figure 1-2:

1. PIC32MX440F512H USB microcontroller for on-board debugging.
2. Regulated +3.3V power supply for powering the starter kit via USB or expansion board.
3. Connector for various expansion boards.
4. USB Host and OTG power supply for powering PIC32 USB applications.
5. USB Type Micro-AB receptacle for OTG and USB device connectivity for PIC32 OTG/device-based applications.

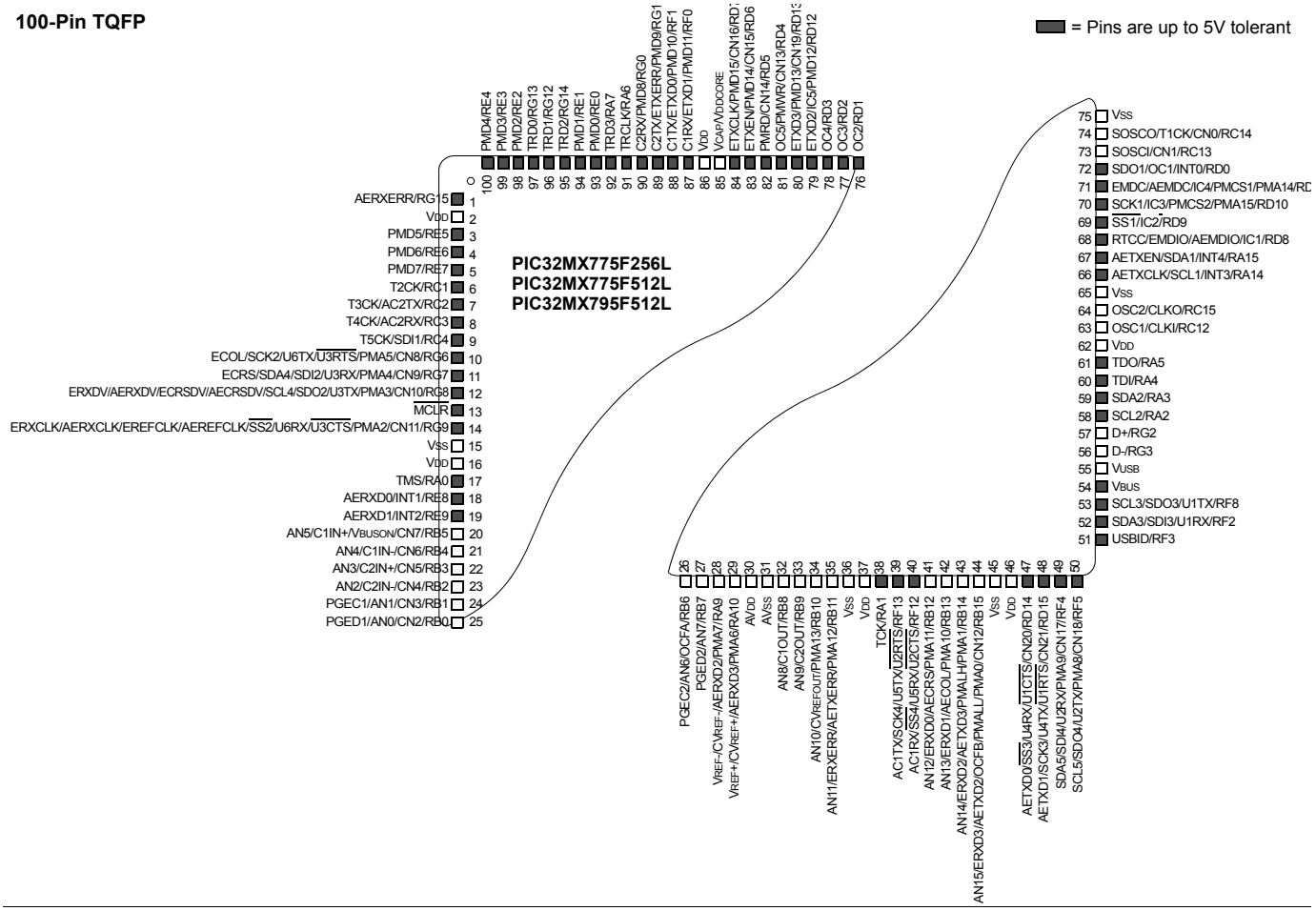
FIGURE 1-2: PIC32 USB STARTER KIT II LAYOUT (UNDERSIDE)



3 - Diagrama de Pinos - Expansion Board IO



4 - Diagrama de Pinos – PIC32MX795F512L



5 - Código – Firmware

O código apresentado não inclui a maioria das funções para a comunicação USB pela sua extensão.

```
#include <p32xxxx.h>
#include "plib.h"
#include "USB/usb.h"
#include "HardwareProfile.h"
#include "USB/usb_function_generic.h"

#include "CheckConnection.h"
#include "LigDesLED.h"

#include <cp0defs.h>
#include "delay.h"
/** CONFIGURATION **************************************************/
#if defined(EXPLORER_16)
  #if defined(__32MX460F512L__) || defined(__32MX795F512L__)
    #pragma config UPLLEN   = ON        // USB PLL Enabled
    #pragma config FPLLMUL  = MUL_15        // PLL Multiplier
    #pragma config UPLLIDIV = DIV_2         // USB PLL Input Divider
    #pragma config FPLLIDIV = DIV_2         // PLL Input Divider
    #pragma config FPLLODIV = DIV_1         // PLL Output Divider
    #pragma config FPBDIV   = DIV_1         // Peripheral Clock divisor
    #pragma config FWDTEN   = OFF           // Watchdog Timer
    #pragma config WDTPS    = PS1           // Watchdog Timer Postscale
    #pragma config FCKSM    = CSDCMD        // Clock Switching & Fail Safe Clock Monitor
    #pragma config OSCIOFNC = OFF           // CLKO Enable
    #pragma config POSCMOD  = HS            // Primary Oscillator
    #pragma config IESO     = OFF           // Internal/External Switch-over
    #pragma config FSOSCEN  = OFF           // Secondary Oscillator Enable (KLO was off)
    #pragma config FNOSC    = PRIPLL        // Oscillator Selection
    #pragma config CP       = OFF           // Code Protect
    #pragma config BWP      = OFF           // Boot Flash Write Protect
    #pragma config PWP      = OFF           // Program Flash Write Protect
    #pragma config ICESEL   = ICS_PGx2      // ICE/ICD Comm Channel Select
    #pragma config DEBUG    = ON            // Background Debugger Enable
  #else
    #error No hardware board defined, see "HardwareProfile.h" and __FILE__
  #endif

#elif defined(PIC32_USB_STARTER_KIT)
  #pragma config UPLLEN   = ON                  // USB PLL Enabled
  #pragma config FPLLMUL  = MUL_15        // PLL Multiplier
                            //Code Timer: =MUL_20
  #pragma config UPLLIDIV = DIV_2         // USB PLL Input Divider
  #pragma config FPLLIDIV = DIV_2         // PLL Input Divider
  #pragma config FPLLODIV = DIV_1         // PLL Output Divider
  #pragma config FPBDIV   = DIV_1         // Peripheral Clock divisor
  #pragma config FWDTEN   = OFF           // Watchdog Timer
  #pragma config WDTPS    = PS1           // Watchdog Timer Postscale
  #pragma config FCKSM    = CSDCMD        // Clock Switching & Fail Safe Clock Monitor
  #pragma config OSCIOFNC = OFF           // CLKO Enable
  #pragma config POSCMOD  = HS            // Primary Oscillator (HS = High Speed Crystal) //Code Timer:
=XT (Resonator, crystal)
```

```
  #pragma config IESO     = OFF          // Internal/External Switch-over
  #pragma config FSOSCEN  = OFF          // Secondary Oscillator Enable (KLO was off)
  #pragma config FNOSC    = PRIPLL       // Oscillator Selection
  #pragma config CP       = OFF          // Code Protect
  #pragma config BWP      = OFF          // Boot Flash Write Protect
  #pragma config PWP      = OFF          // Program Flash Write Protect
  #pragma config ICESEL   = ICS_PGx2     // ICE/ICD Comm Channel Select
  #pragma config DEBUG    = ON           // Background Debugger Enable
                      //Code Timer: =OFF
#else
  #error No hardware board defined, see "HardwareProfile.h"
#endif

/** VARIABLES ******************************************************/

unsigned char OUTPacket[64];        //User application buffer for receiving and holding OUT packets
sent from the host

unsigned char INPacket[64];         //User application buffer for sending IN packets to the host
#pragma udata
BOOL blinkStatusValid;
USB_HANDLE USBGenericOutHandle;
USB_HANDLE USBGenericInHandle;
#pragma udata

/** PRIVATE PROTOTYPES *********************************************/
static void InitializeSystem(void);
void USBDeviceTasks(void);
void YourHighPriorityISRCode(void);
void YourLowPriorityISRCode(void);
void USBCBSendResume(void);
void UserInit(void);
void ProcessIO(void);
void BlinkUSBStatus(void);

/** DECLARATIONS ***************************************************/
#pragma code
#define Trigger PORTBbits.RB2       // select line for Trigger
#define EOS PORTBbits.RB3           // select line for EOS End Of Scan
#define SPI_BAUD  11                //Clock divider Fclk = Fpb/(2 * (SPI_BAUD+1))  Valor mais baixo de
Baud = 11 para obter Fclk = 2.5 MHz

unsigned int mais;                  //Inicia variavel para o contador
unsigned short int DataToTransfer[512];      //Inicia array com shorts(16bits) com 512 aquisiÁıes
unsigned short int DataToTransferCCD[512];
unsigned int pix;                   //pix = PR3
unsigned int naq=20;                //Number of acquist
//unsigned int PRTexp=((pix/3216)*60000000);

//.Interrupt Service Routines

void __ISR (_TIMER_2_VETOR, ipl3) Timer2Handler(void)
{
        mais++;                     //Incrementa contador
        IFS0 &= 0xFFFFFEFF;         //Limpa Timer 2 interrup flag
}
unsigned int mais3;
```

```
unsigned int mais2;

void __ISR( _OUTPUT_COMPARE_3_VETOR, ipl3) OC3Handler(void)
{
	static unsigned int spidata;
	SPI1BUF = 0x00000000;
	while( !SPI1STATbits.SPIRBF);
	spidata=SPI1BUF;
	unsigned short int spidata2 = spidata >> 13; //Shift de 13 bits para direita(short int spidata2
contem 16 bits da aquisiÁ„o)
	DataToTransfer[mais2]=spidata2;		//Guarda aquisiÁ„o
	mais2++;
	IFS0 &= 0xFFFFBFFF;
}

void __ISR( _OUTPUT_COMPARE_2_VETOR, ipl7) OC2Handler(void)
{
	IFS0 &= 0xFFFFFBFF;
}

void __ISR( _OUTPUT_COMPARE_1_VETOR, ipl3) OC1Handler(void)
{
	mais3++;
	IFS0 &= 0xFFFFFFBF;
}

void __ISR (_TIMER_3_VETOR, ipl3) Timer3Handler(void)
{
	IFS0 &= 0xFFEFFFFF;					//Limpa Timer 3 interrup flag
}

//.SPI Initialization

void IniSPI (void)
{
	// 1. inicia SPI
	int spidata;		//Introduz vari·vel SPI
	SPI1CON = 0; // Stops and resets the SPI1.
	spidata=SPI1BUF; // clears the receive buffer
	IFS0CLR=0x03800000; // clear any existing event
	//IPC5CLR|=0x1f000000;
	//IPC5SET|=0x0d000000;
	//IEC0SET|=0x03800000;
	// 2. Define SPI, Baud Rate
	OpenSPI1(SPI_MODE32_ON|MASTER_ENABLE_ON|DISABLE_SDO_PIN|CLK_POL_ATIVE_LO
W , SPI_ENABLE);
	SPI1BRG = SPI_BAUD;
}

//.Timer Initialization & Resets

void SetTimer2(void)
{
	T2CON   = 0;	//Disable Timer 2.
	TMR2    = 0;	//Clear Timer 2
	PR2     = 1200;  // 900 para PS=1 obter 15us entre aquisiÁoes
	IFS0    &= 0xFFFFFEFF;   //Clear Timer 2 Flag
```

```
        IEC0       |= 0x00000100;   //Set Timer 2 Interrupt
        IPC2       |= 0x0000000D;   // Priority = 3, Sub-Priority = 1
        T2CONSET = 0x8000; // Start timer with PreScaler 0. To PS:256->0x8070
}

void SetTimer3(void)
{
        T3CON   = 0;       //Disable Timer 2. Write bits configurations
        TMR3    = 0;
        PR3     = 19375;// 94 para Texp=1s ps=256
        IFS0    &= 0xFFFFEFFF;    //Clear Timer 3 Flag
        IEC0    |= 0x00001000;   //Set Timer 3 Interrupt
        IPC2    |= 0x0000000D;   // Priority = 3, Sub-Priority = 1
        T3CONSET = 0x8000; // Start timer PS 256-> 0x8070
}

void Timer2Reset(void)
{
        T2CON= 0;
        TMR2=0;
        PR2=0;
}

void Timer3Reset(void)
{
        T3CON=0;
        TMR3=0;
        PR3=0;
}

//.Main
#if defined(__18CXX)
void main(void)
#else
//MAIN
int main(void)
#endif
{
        SYSTEMConfigPerformance(80000000);
        unsigned int temp = 0;
        temp = _CP0_GET_EBASE();
        _CP0_SET_EBASE(temp | 0x80000000);
        temp = _CP0_GET_INTCTL();
        _CP0_SET_INTCTL(temp | 0x00000020);//(VS = 1)
        temp = _CP0_GET_CAUSE();
        _CP0_SET_CAUSE(temp | 0x00800000);//(IV = 1)
        temp = _CP0_GET_STATUS();
        _CP0_SET_STATUS(temp & 0xFFBFFFFD);//(EXL = 0, BVL = 0)
        INTCON  |= 0x00001000;   //(MEVC = 1)

        INTEnableSystemMultiVectoredInt();
        InitializeSystem();
  #if defined(USB_INTERRUPT)
    USBDeviceAttach();
  #endif
  while(1)
  {
```

```
    #if defined(USB_POLLING)
                // Check bus status and service USB interrupts.
    USBDeviceTasks(); // Interrupt or polling method.  If using polling, must call
    #endif
                // Application-specific tasks.
                // Application related code may be added here, or in the ProcessIO() function.
    ProcessIO();
  }//end while
}//end main

static void InitializeSystem(void)
{
  #if (defined(__18CXX) & !defined(PIC18F87J50_PIM))
    ADCON1 |= 0x0F;             // Default all pins to digital
  #elif defined(__C30__)
    #if defined(__PIC24FJ256GB110__) || defined(__PIC24FJ256GB106__)
      AD1PCFGL = 0xFFFF;
     #elif defined(__dsPIC33EP512MU810__)
        ANSELA = 0x0000;
        ANSELB = 0x0000;
        ANSELC = 0x0000;
        ANSELD = 0x0000;
        ANSELE = 0x0000;
        ANSELG = 0x0000;
       RPINR19 = 0;
       RPINR19 = 0x64;
       RPOR9bits.RP101R = 0x3;
    #endif
  #elif defined(__C32__)
    AD1PCFG = 0xFFFF;
  #endif
  #if defined(PIC18F87J50_PIM) || defined(PIC18F46J50_PIM) || defined(PIC18F_STARTER_KIT_1) ||
defined(PIC18F47J53_PIM)
  {
    unsigned int pll_startup_counter = 600;
    OSCTUNEbits.PLLEN = 1;  //Enable the PLL and wait 2+ms until the PLL locks before enabling USB
module
    while(pll_startup_counter--);
  }
  //Device switches over automatically to PLL output after PLL is locked and ready.
  #endif
  #if defined(PIC18F87J50_PIM)
  WDTCONbits.ADSHR = 1;                         // Select alternate SFR location to access
ANCONx registers
  ANCON0 = 0xFF;            // Default all pins to digital
  ANCON1 = 0xFF;            // Default all pins to digital

  WDTCONbits.ADSHR = 0;                         // Select normal SFR locations
  #endif
  #if defined(PIC18F46J50_PIM) || defined(PIC18F_STARTER_KIT_1) || defined(PIC18F47J53_PIM)
  ANCON0 = 0xFF;            // Default all pins to digital
  ANCON1 = 0xFF;            // Default all pins to digital
  #endif
  #if defined(DSPIC33EP512MU810_PIM)
        PLLFBD = 58;                            /* M  = 60          */
        CLKDIVbits.PLLPOST = 0;             /* N1 = 2           */
        CLKDIVbits.PLLPRE = 0;              /* N2 = 2           */
```

```
        OSCTUN = 0;
  /*    Initiate Clock Switch to Primary
   *    Oscillator with PLL (NOSC= 0x3)*/
  __builtin_write_OSCCONH(0x03);
        __builtin_write_OSCCONL(0x01);
        while (OSCCONbits.COSC != 0x3);
  ACLKCON3 = 0x24C1;
  ACLKDIV3 = 0x7;
  ACLKCON3bits.ENAPLL = 1;
  while(ACLKCON3bits.APLLCK != 1);
  #endif
  #if defined(PIC24FJ64GB004_PIM) || defined(PIC24FJ256DA210_DEV_BOARD)
  {
     unsigned int pll_startup_counter = 600;
     CLKDIVbits.PLLEN = 1;
     while(pll_startup_counter--);
  }
  //Device switches over automatically to PLL output after PLL is locked and ready.
  #endif
  #if defined(USE_USB_BUS_SENSE_IO)
  tris_usb_bus_sense = INPUT_PIN; // See HardwareProfile.h
  #endif
  #if defined(USE_SELF_POWER_SENSE_IO)
  tris_self_power = INPUT_PIN;     // See HardwareProfile.h
  #endif
        USBGenericOutHandle = 0;
        USBGenericInHandle = 0;
  UserInit();                          //Application related initialization.  See user.c
  USBDeviceInit();          //usb_device.c.  Initializes USB module SFRs and firmware
                                       //variables to known states.
}//end InitializeSystem

void UserInit(void)
{
  mInitAllLEDs();
  mInitAllSwitches();
  blinkStatusValid = TRUE;          //Blink the normal USB state on the LEDs.
}//end UserInit

void ProcessIO(void)
{
  //Blink the LEDs according to the USB device status, but only do so if the PC application isn't
connected and controlling the LEDs.
  if(blinkStatusValid)
  {
     BlinkUSBStatus();
  }
  if((USBDeviceState < CONFIGURED_STATE)||(USBSuspendControl==1)) return;
  if(!USBHandleBusy(USBGenericOutHandle))          //Check if the endpoint has received any data
from the host.
  {
     switch(OUTPacket[0]) //Chegada de Data por USB verifica/lÍ byte 0 dos 64 bytes do datapacket
     {
        case 0x80:             //Button Ligar LEDs in Softaware
           blinkStatusValid = FALSE;   //Desliga piscar de LEDs, Liga os dois LEDS (1 e 2)
           LigDesLED():
           break;
```

```
case 0x81:              //Button ComunicaÁ„o USB in Software
  blinkStatusValid = FALSE;
  CheckConnection();
  Break;
case 0x82:              //Button Iniciar CCD in Software
  pix=OUTPacket[1];
  blinkStatusValid = FALSE;
  IniSPI();             //Init SPI
  //Set Trigger and EOS pins
  AD1PCFGSET=0x000C;    //Set RB2, RB3 as digital (Trigger and EOS)
  TRISBSET=0x000C;      //Set RB2, RB3 as input    (Trigger and EOS)
  PORTBSET=0x000C;      //Set RB2, RB3 line
  mais3=0;              //Clear counter OC1
  //OC1 enable Interrupt and Flag Clear
  IFS0    &= 0xFFFFFFBF;                //Clear OC1 Flag
  IEC0    |= 0x00000040;                //Set OC1 Interrupt
  IPC1    |= 0x000C0000;                // Priority = 3
  //Init OC1 OC2 (Cam Clock and Cam Start
  SetTimer3();
  OpenOC1( OC_ON | OC_TIMER3_SRC | OC_CONTINUE_PULSE ,(PR3/2) , 0);     //Clock
  OpenOC2( OC_ON | OC_TIMER3_SRC | OC_SINGLE_PULSE , 0 ,0 );            //Start
  unsigned int index_ccd;
  while(mais3<3216);        //3216 clocks (to perform 512 pixel aquis)
  {
        for(index_ccd=0;index_ccd<512;index_ccd++)      //Count pixel
        {
              while(Trigger==0);        //Wait for trigger and Data Video in
              //Use of timer2 to count between CS signals
              mais2=0;
              T2CON   = 0;      //Disable Timer 2.
              TMR2    = 0;      //Clear Timer 2
              PR2     = 1200;           // 900 para PS=1 obter 15us entre aquisiÁoes
              T2CONSET = 0x8000; // Start timer with PreScaler 0. To PS:256->0x8070
              //Interrupt for OC3 enable (In this interrupt aquisi will be made)
              IFS0    &= 0xFFFFBFFF;                //Clear OC3 Flag
              IEC0    |= 0x00004000;                //Set OC3 Interrupt
              IPC3    |= 0x000C0000;                // Priority = 3
              OpenOC3(OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE , 100 ,0 );
              while(mais2<naq);         //mais2 counter of number of acquisitions
              unsigned int t;
              unsigned int soma=0;      //soma to perform mean of acq
              for (t=0;t<naq;t++)
              {
                    soma=soma+DataToTransfer[t];
              }
              soma=soma/naq;            //arythmetic mean of acq
              DataToTransfer[0]=soma;
              DataToTransferCCD[index_ccd]=DataToTransfer[0];//just one acquis
              Timer2Reset();
              CloseOC3();
              while(Trigger==1);        //wait if Trigger is still high
        }
  }
  OpenOC2( OC_ON | OC_TIMER3_SRC | OC_SINGLE_PULSE , 0 ,0 );//New Start at the end
  CloseOC1();
  //Transfer Data throw usb
  unsigned int a;
```

```
        for(a=0;a<16;a++)
        {
                unsigned short int e;
                for(e=0;e<32;e++)
                {
                        DataToTransferCCD[e]=DataToTransferCCD[e+(a*32)];
                }
                unsigned int p;
                for(p=0;p<1;p++){
                        unsigned int j,i;
                        for(j=0, i=0;j<64;j++)
                        {
                                INPacket[j]=DataToTransferCCD[i];
                                j++;
                                INPacket[j]=DataToTransferCCD[i]>>8;
                                i++;
                        }
                        if(!USBHandleBusy(USBGenericInHandle))
                        {
                                USBGenericInHandle =
USBGenWrite(USBGEN_EP_NUM,(BYTE*)&INPacket,USBGEN_EP_SIZE);
                        }
                        DelayMs(1);
                }
        }
        break;
      case 0x83:            //Button ADC in Software
        blinkStatusValid = FALSE;
        mais=0;             //Counter in Timer2 interrupt
        IniSPI();           //Initialization of SPI module
        mais2=0;
        T2CON   = 0;        //Disable Timer 2.
        TMR2    = 0;        //Clear Timer 2
        PR2     = 1200;  // 900 para PS=1 obter 15us entre aquisiÁoes
        // If Acqusiition is made on timer2 interrupt, activate timer2 interrupt
        //IFS0    &= 0xFFFFFEFF;   //Clear Timer 2 Flag
        //IEC0    |= 0x00000100;   //Set Timer 2 Interrupt
        //IPC2    |= 0x0000000D;   // Priority = 3, Sub-Priority = 1
        T2CONSET = 0x8000; // Start timer with PreScaler 0. To PS:256->0x8070
        //OC3 interrupt enable (in the interrupt the acquisition will be made)
        IFS0    &= 0xFFFFBFFF;                  //Clear OC3 Flag
        IEC0    |= 0x00004000;                  //Set OC3 Interrupt
        IPC3    |= 0x000C0000;                  //Priority = 3
        OpenOC3(OC_ON | OC_TIMER2_SRC | OC_CONTINUE_PULSE , 100 ,0 );          //using a PR2
of 1200, RS=100
        while(mais2<512);                   //Counter of 512 acquisitions
        //Reset OC3 and Timer2
        CloseOC3();
        Timer2Reset();
        //Data transfer throw usb
        unsigned int b;
        for(b=0;b<16;b++)
        {
                unsigned short int t;
                for(t=0;t<32;t++)
                {
                        DataToTransfer[t]=DataToTransfer[t+(b*32)];
```

```
                    }
                    unsigned int j,i;
                    for(j=0, i=0;j<64;j++)
                    {
                            INPacket[j]=DataToTransfer[i];
                            j++;
                            INPacket[j]=DataToTransfer[i]>>8;
                            i++;
                    }
                    if(!USBHandleBusy(USBGenericInHandle))
                    {
                            USBGenericInHandle =
USBGenWrite(USBGEN_EP_NUM,(BYTE*)&INPacket,USBGEN_EP_SIZE);
                    }
                    DelayMs(1);
            }
            break;
      }
            //Rearm the OUT endpoint for the next packet:
      USBGenericOutHandle =
USBGenRead(USBGEN_EP_NUM,(BYTE*)&OUTPacket,USBGEN_EP_SIZE);
   }

}//end ProcessIO
```

6 - Código – Software

O código apresentado exclui as funções da comunicação USB e de inicialização de componentes pela sua extensão. O código apresentado é do ficheiro forms.h (é neste ficheiro que se gere a GUI e os eventos)

```
#pragma once
//Includes
#include <windows.h>
#include <errno.h>
#include "usb.h"
#include <fstream>
#include <stdio.h>
//VID e PID no USB device descriptor.
#define MY_VID   0x04D8 //0x04D8
#define MY_PID   0x0204   //0x0204
#define CONNECTED 1
// Global Variables
unsigned char Connection_Status;
usb_dev_handle *MyLibusbDeviceHandle = NULL; /* the device handle */

//INICIALIZAÇÃO DE COMPONENTES OMITIDA

private: System::Void Connect_btn_Click(System::Object^  sender, System::EventArgs^  e)  //Bot„o
Conectar
 {
// Primeiro ponto passa por encontrar o nosso dispositivo USB por VID e PID
          if ( Connection_Status != CONNECTED)  // Se ainda n„o conectado
          {
                    usb_init();                              //inicia a libraria usb
                    usb_find_busses();                  //encontra todos os pontos de conecÁao
                    usb_find_devices();         //encontra dispositivos conectados
           //Da lista de dispositivos encontrados, procura se algum corresponde ao nosso VID/PID
                    struct usb_bus *bus;
                    struct usb_device *dev;
                    for(bus = usb_get_busses(); bus; bus = bus->next)
                    {
                              for(dev = bus->devices; dev; dev = dev->next)
                              {
                                        if(dev->descriptor.idVendor == MY_VID && dev-
>descriptor.idProduct == MY_PID)
                                        {
                                                  MyLibusbDeviceHandle = usb_open(dev);     //Abre o
nosso dispositivo USB
                                                  break;
                                        }
                              }
                     }
                    if(!MyLibusbDeviceHandle)
                    {
                              return;
                    }
                    if(usb_set_configuration(MyLibusbDeviceHandle, 1) < 0) //Coloca dispositivo ativo
                    {
                              usb_close(MyLibusbDeviceHandle);
                              return;
```

```
                }
                if(usb_claim_interface(MyLibusbDeviceHandle, 0) < 0)  //Pede interface com
sistema operativo
                {
                        //Se pedido de interface com SO falhar, fecha dispositivo aberto
                        usb_close(MyLibusbDeviceHandle);
                        return ;
                }
                ToggleLED_btn->Enabled = true;    //Coloca bot„o disponivel
                GetPushbuttonState_btn->Enabled = true;  //Coloca bot„o disponivel
                StateLabel->Enabled = true;          //Coloca label disponivel
                label1->Enabled = true;              //Coloca label disponivel
                button1->Enabled = true;             //Coloca bot„o disponivel
                button2->Enabled = true;             //Coloca bot„o disponivel
                Connection_Status = CONNECTED;  //Status È agora conectado
        }
}

private: System::Void ToggleLED_btn_Click(System::Object^  sender, System::EventArgs^  e)
{
        char OutputPacketBuffer[64];       //Aloca memoria do buffer que contem data que enviar·
para o dispositivo USB
        OutputPacketBuffer[0] = 0x80;      //0x80 È o comando "Toggle LEDs" no firmware
        //Escreve data para um endpoint bulk. A funÁao chamada ir· enviar 64b de data.
        if(usb_bulk_write(MyLibusbDeviceHandle, 0x01, &OutputPacketBuffer[0], 64, 5000) != 64)
        {
                return;
        }
 }

private: System::Void GetPushbuttonState_btn_Click(System::Object^  sender, System::EventArgs^  e)
{
        char OutputPacketBuffer[64];       //Aloca memoria do buffer que contem data que enviar·
para o dispositivo USB
        char InputPacketBuffer[64];        //Aloca memoria do buffer que contem data que receber·
do dispositivo USB
        OutputPacketBuffer[0] = 0x81;      //0x81 È o comando "Get Pushbutton State" no firmware
        OutputPacketBuffer[1] = 25;
       //Escreve numeros de 2 a 63 nos restantes bytes disponiveis dos 64, para teste.
        for(unsigned int i = 2; i < 64; i++)
        {
                OutputPacketBuffer[i] = (unsigned char)i;
        }
       //Para obter o estado enviamos byte com o comando "Get Pushbutton State" nele.
       //usb_bulk_write() envia 64 bytes de data para o dispositivo USB
        if(usb_bulk_write(MyLibusbDeviceHandle, 0x01, &OutputPacketBuffer[0], 64, 5000) != 64)
        {
                return;
        }
        //Verifica a resposta do firmware
        //usb_bulk_read() recebe 64 bytes de data do dispositivo USB
        if(usb_bulk_read(MyLibusbDeviceHandle, 0x81, &InputPacketBuffer[0], 64, 5000) != 64)
        {
                return;
        }
        //Verifica se os comando no byte 2 È o valor 3 como enviado para teste
        if (InputPacketBuffer[2]==3)
```

```
		{
			label1->Text = "ComunicaÁ„o Efetuada";
		}
		else
		{
			label1->Text = "Falha de LigaÁ„o";
		}
		//InputPacketBuffer[0] is an echo back of the command.
		//InputPacketBuffer[1] contains the I/O port pin value for the pushbutton.
		if (InputPacketBuffer[1] == 0x01)
		{
			StateLabel->Text = "Estado: N„o pressionado";
		}
		else
		{
			StateLabel->Text = "Estado: Pressionado";
		}
}

private: System::Void Form1_FormClosed(System::Object^  sender,
System::Windows::Forms::FormClosedEventArgs^  e)
{
		if ( Connection_Status == CONNECTED)
		{
			//The following functiom releases a previously claimed interface
			usb_release_interface(MyLibusbDeviceHandle, 0);
			//closes a device opened
			usb_close(MyLibusbDeviceHandle);
		}
		return;
}

private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
{
		char OutputPacketBuffer[64];	//Allocate a memory buffer which will contain data to
send to the USB device
		OutputPacketBuffer[0] = 0x82;	//0x82  command in the firmware
		int pix = 19000;
		char pix1=pix;
		OutputPacketBuffer[1] = 19000;				//pix duration
		int naq = 20;
		OutputPacketBuffer[2] = 20;
		char InputPacketBuffer[64];
		unsigned char dados2[64];
		unsigned short Data0a8, Data8a16, Data;
		unsigned int Data8a16s;
		ArrayList ^ xvals = gcnew ArrayList();
		ArrayList ^ yvals = gcnew ArrayList();
		FILE * pFile;
		pFile = fopen ("Dados.txt","w");
		//Writes data to a bulk endpoint. The Function call will send out 64 bytes to the USB Device.
		if(usb_bulk_write(MyLibusbDeviceHandle, 0x01, &OutputPacketBuffer[0], 64, 5000) != 64)
		{
			return;
		}
		//usb_bulk_read() recebe 64 bytes de data do dispositivo USB
		for(unsigned int l=0;l<16;l++)
```

```
		{
			if(usb_bulk_read(MyLibusbDeviceHandle, 0x81, &InputPacketBuffer[0], 64, 5000)
!= 64)
			{
				return;
			}
			memcpy( dados2, InputPacketBuffer, sizeof( dados2 ) );		//Transforma signed
char em unsigned char
			//Leitura de dados, concatenaÁ„o de chars e 32 conversıes como resultado
			for (unsigned int i = 0; i < 63; i=i+2)
			{
				Data0a8 = dados2[i];
				Data8a16 = dados2[i+1];
				Data8a16s = Data8a16<<8;
				Data = (Data0a8|Data8a16s);
				int Data_int=(int)Data;
				fprintf(pFile,"%d %x \n",Data_int, Data_int  );
				yvals->Add(Data_int);
			}
		}
		for (unsigned int q=0; q<512; q++)   //q<512
		{
			xvals->Add(q);
		}
		chart1->Series["AquisiÁ„o"]->Points->DataBindXY(xvals, yvals);
		fclose(pFile);
		StateLabel->Text = "AquisiÁ„o CCD";
}

private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e)
{
		char OutputPacketBuffer[64];		//Allocate a memory buffer which will contain data to
send to the USB device
		OutputPacketBuffer[0] = 0x83;		//0x83  command in the firmware
		char InputPacketBuffer[64];
		unsigned char dados[64];
		unsigned short Data0a8, Data8a16, Data;
		unsigned int Data8a16s;
		ArrayList ^ xvals = gcnew ArrayList();
		ArrayList ^ yvals = gcnew ArrayList();
		FILE * pFile;
		pFile = fopen ("Dados.txt","w");
		//Writes data to a bulk endpoint. The Function call will send out 64 bytes to the USB Device.
		if(usb_bulk_write(MyLibusbDeviceHandle, 0x01, &OutputPacketBuffer[0], 64, 5000) != 64)
		{
			return;
		}
		//usb_bulk_read() recebe 64 bytes de data do dispositivo USB
		for(unsigned int l=0;l<16;l++)
		{
			if(usb_bulk_read(MyLibusbDeviceHandle, 0x81, &InputPacketBuffer[0], 64, 5000)
!= 64)
			{
				return;
			}
			memcpy( dados, InputPacketBuffer, sizeof( dados ) );		//Transforma signed
char em unsigned char
```

```
                //Leitura de dados, concatenaÁ„o de chars e 32 conversıes como resultado
                for (unsigned int i = 0; i < 63; i=i+2)
                {
                        Data0a8 = dados[i];
                        Data8a16 = dados[i+1];
                        Data8a16s = Data8a16<<8;
                        Data = (Data0a8|Data8a16s);
                        int Data_int=(int)Data;
                        fprintf(pFile,"%d %x \n",Data_int, Data_int  );
                        yvals->Add(Data_int);
                }
        }
        for (unsigned int q=0; q<512; q++)
        {
                xvals->Add(q);
        }
        chart1->Series["AquisiÁ„o"]->Points->DataBindXY(xvals, yvals);
        fclose(pFile);
        StateLabel->Text = "AquisiÁ„o ADC";
}
```