UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA

PEDRO HENRIQUE SILVA E OLIVEIRA

IMPLEMENTAÇÃO DE SISTEMA PARA CONTAGEM MÚLTIPLA DE PULSOS UTILIZANDO FPGA

LIMEIRA

2019

PEDRO HENRIQUE SILVA E OLIVEIRA

IMPLEMENTAÇÃO DE SISTEMA PARA CONTAGEM MÚLTIPLA DE PULSOS UTILIZANDO FPGA

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia da Universidade de Campinas, como requisito parcial para obtenção do título de Engenheiro de Telecomunicações.

ORIENTADOR: LUIS FERNANDO DE AVILA

LIMEIRA

2019

Autor: Pedro Henrique Silva e Oliveira

Título: Implementação de sistema para contagem múltipla de pulsos utilizando FPGA

Natureza: Trabalho de Conclusão de Curso em Engenharia de Telecomunicações

Instituição: Faculdade de Tecnologia, Universidade Estadual de Campinas

Aprovado em: ___ / ___ / _____

BANCA EXAMINADORA

_____

Prof. Dr. Luis Fernando de Avila – Presidente

Faculdade de Tecnologia (FT/UNICAMP)

_____

Prof. Dr. Rangel Arthur - Avaliador

_____

Prof. Dr. Leandro Ronchini Ximenes - Avaliador

Dedico este trabalho:

À minha mãe Cleide e ao meu pai Israel, que sempre trabalharam para que nada faltasse para mim e minhas irmãs, e desde cedo me ensinaram que a educação era o caminho libertador, proporcionando-me a oportunidade de construir um futuro melhor.

À minha amada namorada e companheira Rafaela pelo amor incondicional e todo apoio e carinho que encontro em seus abraços.

Aos meus amigos, tanto os que estiveram presentes durante as incontáveis horas de estudos ao longo desta graduação, quanto aqueles que deixei de compartilhar bons momentos nestes últimos anos.

AGRADECIMENTOS

Agradeço ao Prof.° Dr. Cristiano Gallep, por ter sido meu mentor e orientador durante três anos da graduação. Além de um ser um bom professor, sua visão de mundo e determinação em fazer as coisas acontecerem foram realmente instigantes. Também agradeço aos membros do Laboratório de Fotônica Aplicada (LaFA) que contribuíram para a realização deste trabalho: Sr. João, Daniel Vaz e Rafaela Nogueira.

Agradeço ao Prof.° Dr. Luis Fernando de Avila, excelente educador, que mesmo enfrentando adversidades pessoais, se dispôs a me orientar nesta etapa final da minha graduação.

Agradeço ao meu amigo e colega de turma Guilherme Paulino, pela ajuda e dedicação no desenvolvimento deste trabalho, e por compartilhar tantos ensinamentos, sempre com boa vontade e paciência. Serei eternamente grato pela oportunidade que me deu de hoje ser estagiário na empresa em que trabalhamos juntos, Pi Tecnologia, à qual tenho grande satisfação em fazer parte.

Agradeço à minha família, fonte de amor e inspiração. Também agradeço à Deus, pela minha vida, saúde e tudo que conquistei até aqui.

RESUMO

Os avanços atingidos pela tecnologia nos campos da eletrônica, computação, instrumentação e fotônica, têm oferecido amplo desenvolvimento em áreas como medições científicas, espectroscopia, biotecnologia, diagnóstico e tratamento médico, física de alta energia, entre outros, demandando o desenvolvimento de detectores e sistemas com alta performance e precisão. Válvulas fotomultiplicadoras são dispositivos mais viáveis em sua relação custo/benefício para detecção de luz de intensidade ultra fraca, sendo em geral muito utilizadas na construção de câmaras escuras para pesquisas e testes com micro-organismos diversos. No Laboratório de Fotônica Aplicada - LaFA (Faculdade de Tecnologia - UNICAMP), a detecção de emissão de luz ultra-fraca em sementes tem sido explorada em estudos toxicológicos e cronobiológicos, utilizando as PMTs para detecção de luz neste espectro. Neste trabalho, foi implementado um sistema digital com suporte para contagem de pulsos de fóton-contagem de até 8 canais independentes utilizando tecnologia baseada em FPGA, um dispositivo lógico programável. Utilizar FPGA demonstrou ser uma alternativa economicamente viável e de desempenho confiável, fornecendo boa precisão de contagem em taxas de até 100 MS/s. O resultado deste projeto é uma unidade de contagem de pulsos chamada de “Multichannel Counting Unit (MCU)” implementada em uma placa de prototipagem e desenvolvimento Altera DE2-115. Na MCU, fotodetectores enviam pulsos elétricos ao FPGA, onde são contados ao longo do tempo por 8 contadores de 32 bits e enviados à um computador via interface serial. Um script em MatLab é responsável por ler, armazenar e exibir em uma interface gráfica os valores de contagem enviados ao PC.

Palavras chave: *FPGAs, sistema digital, fóton-contagem.*

ABTRACT

Technology's advances in the fields of electronics, computing, instrumentation, and photonics have offered broad development in areas such as scientific measurement, spectroscopy, biotechnology, medical diagnosis and treatment, high energy physics, among others, requiring the development of detectors and systems with high performance and accuracy. Photomultiplier tubes (PMT) are the most cost-effective devices for ultra-weak light emission detecting and are generally widely used in the construction of dark chambers for research and testing with various microorganisms. At the Applied Photonics Laboratory - LaFA (Faculty of Technology - UNICAMP), detection of ultra-weak light emission in seeds has been explored in toxicological and chronobiological studies using PMTs for light detection in this spectrum. Through this work, a digital system with support for up to 8 independent photon-counting pulse counts was implemented using FPGA-based technology, a programmable logic device. Working using FPGA has proven to be an economically viable and reliable performance alternative, delivering good counting accuracy at rates up to 100 MS / s. The result of this project is a pulse counting unit called "Multichannel Counting Unit (MCU)" implemented on an Altera DE2-115 prototyping and development board. Photodetectors sends electrical pulses to the FPGA, where they are counted over time using 8 32-bit counters and sent to a computer via serial interface. A MatLab script is responsible for reading, storing and displaying in a graphical interface the count values sent to the PC.

Keywords: *FPGAs, digital system, photon counting.*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

ASIC - Application Specific Integrated Circuit

BNC - Bayonet Neill Concelman

CAD - Computer-Aided Design

CI – Circuito Integrado

CLB – Configurable Logic Block

CMOS - Complementary Metal-Oxide Semiconductor

CODEC - Codificador/Decodificador

CPLD - Complex Programmable Logic Device

DSP - Digital Signal Processing

EEPROM - Electrically-Erasable Programmable Read Only Memory

EPLD - Erasable Programmable Logic Device

FIFO - First-In First Out

FPGA – Field Programmable Gate Array

FSM - Finite State Machine

GAL - Generic Array Logic

GPIO - General Purpose Input/Output

HCPLD - High Complexity Programmable Logic Device

HDL – Hardware Description Language

HSMC - High Speed Mezzanine Card

IEEE - Institute of Electrical and Electronics Engineers

IOB – Input/Output Block

IP - Intelectual Property

IS - Interconnect Switch

LAB - Logic Array Block

LaFA - Laboratório de Fotônica Aplicada

LCD - Liquid Crystal Display

LE - Logic Element

LED - Light-Emitting Diode

LUT - LookUp Table

MCU - Multichannel Counting Unit

OLMC - Output Logic Macrocell

PAL - Programmable Array Logic

PAR - Place and Rout

PC - Personal Computer

PCI - Placa de Circuito Impresso

PDS - Processamento Digital de Sinais

PLA - Programmable Logic Array

PLD – Programmable Logic Device

PLL - Phase-Locked Loops

PMT – Photomultiplier Tube

RAM - Random Access Memory

SD - Secure Digital Card

SoC - System on Chip

SPLD - Simple Programmable Logic Device

TTL – Transistor Transistor Logic

UART - Universal Assynchronous Receiver/Transmitter

USB - Universal Serial Bus

UV - Ultravioleta

VHDL – VHSIC Hardware Description Language

VHSIC - Very High Speed Integrated Circuit

VLSI – Very Large Scale Integration

VLSI – Very Large Scale of Integration

1 INTRODUÇÃO

Um sistema digital é composto por dispositivos interconectados que tomam sinais digitais em suas entradas, manipulam, armazenam, processam e transmitem informações em forma de sinais digitais em suas saídas. Essencialmente, estes dispositivos são eletrônicos, mas também podem ser mecânicos, pneumáticos ou magnéticos [1]. Sistemas digitais são amplamente utilizados em áreas de processamento de sinais (i.e., áudio, imagem, voz), telecomunicações, computação, armazenamento de dados, entre outros, sendo tão comuns na rotina das pessoas que a maioria delas nem os percebem. A maioria dos sistemas eletrônicos são parcialmente ou totalmente digitais e, características como a simplicidade do projeto, versatilidade e capacidade de produzir sistemas grandes e complexos, fazem destes excelentes para processamento e armazenamento de informação [2].

A invenção do transistor bipolar de junção em 1948 pelos físicos John Bardeen, Walter Brattain e Willian Shockley revolucionou a forma como computadores e outros sistemas eletrônicos foram construídos. O transistor é um dispositivo semicondutor utilizado como uma chave elétrica que permite a passagem de corrente entre dois de seus terminais à partir de uma corrente injetada em um terceiro terminal. Inicialmente eram utilizados como componentes discretos, mas com o avanço da tecnologia, engenheiros foram capazes de criar circuitos eletrônicos contendo dispositivos cada vez menores e agrupá-los em um único chip, com quantidades que ultrapassam a ordem de centenas de milhares de unidades. Esta tecnologia é nomeada de circuito integrado (CI), e graças ao constante desenvolvimento e aprimoramento dos recursos que são empregados, a indústria eletrônica deu saltos gigantescos com relação às demais, pois os CIs são

expressivamente baratos quando produzidos em larga escala, confiáveis e consomem muito pouca potência. Publicada em uma revista como uma observação, em meados dos anos 60, Gordon E. Moore previu que o número de transistores dos circuitos integrados teria um aumento de 100% pelo mesmo custo, a cada 18 meses [3]. Essa observação tornou-se realidade e ficou conhecida como *Lei de Moore*, sendo estabelecida como uma premissa para a indústria de semicondutores do século XX, que investiram fortemente em pesquisa e desenvolvimento, proporcionando evolução contínua do desempenho e produção em larga escala com custos cada vez mais acessíveis.

Essa evolução permitiu que empresas de engenharia criassem CIs de diversas arquiteturas com grande capacidade de processamento e robustez quanto ao seu funcionamento, no qual podemos citar o FPGA, um dispositivo lógicos programável determinístico que pode ser projetado para implementar da mais simples função da eletrônica digital até circuitos extremamente complexos, sendo o foco deste estudo.

Organização deste trabalho:

No capítulo 2 são apresentados os fundamentos teóricos relativos aos dispositivos lógicos programáveis, os principais tipos de dispositivos, arquiteturas e princípio de funcionamento.

No capítulo 3 é abordado a plataforma de desenvolvimento utilizada, contemplando o kit de prototipagem, o FPGA e os detectores utilizados. Também é apresentado a descrição funcional do projeto, as entidades e suas arquiteturas, bem como a elaboração do quadro de dados para transmissão ao computador.

No capítulo 4 são discutidos os resultados do trabalho, tais como a quantidade de recursos utilizados no dispositivo, a interface gráfica implementada através do Matlab e a caracterização dos detectores utilizando o sistema implementado.

Também são apresentadas funcionalidades e melhorias que podem ser implementadas em versões futuras do projeto.

1.1 Objetivos

O objetivo deste trabalho é apresentar um sistema digital para contagem de pulsos de até 8 canais independentes ao longo do tempo, utilizando tecnologia baseada em FPGA e uma placa de desenvolvimento e prototipagem.

1.1.1 Objetivos específicos

- Escrita da lógica programável utilizando linguagem de descrição de hardware;
- Escrita de um programa em Matlab para aquisição, leitura, armazenamento e visualização em tempo real da contagem obtida nos canais disponíveis.

2 DISPOSITIVOS LÓGICOS PROGRAMÁVEIS

Com a introdução dos Circuitos Integrados na década de 60, a crescente demanda por dispositivos com maior capacidade de processamento e em tamanhos reduzidos, no qual o circuito lógico seria definida apenas pelo usuário final era propícia para o surgimento dos PLDs. Assim, PLDs podem ser definidos como CI's que são configurados pelo usuário, eliminando esta etapa da fabricação do CI e consequentemente, facilitando mudanças posteriores no projeto [4].

O progresso de tecnologias de integração em escalas muito grandes (VLSI, Very Large Scale Integration) possibilitou o projeto de PLDs. PLDs contém uma arquitetura geral baseada em chaves programáveis que permitem configurar as rotas dos circuitos internos para realizarem a função desejada. O programador ou usuário final consegue mudar a configuração destas chaves ao escrever um programa em uma linguagem de descrição de hardware (HDL, Hardware Description Language), tais como VHDL, Verilog ou SystemVerilog, e gravando este programa no chip. Como PLDs são em geral reprogramáveis, eles tornam-se ótimos para prototipagem, uma vez que o projetista pode programar o PLD para realizar uma tarefa, fazer mudanças na descrição de hardware e reprogramá-lo para testar novamente no mesmo chip. Outra vantagem oferecida pelo fato do dispositivo ser reprogramado em hardware é a redução de custos para prototipagem. Uma desvantagem está no fato de que a performance pode ser limitada quando comparada à um ASIC (Application Specific Integrated Circuit) ou à um CI, isto porque as funções são realizadas à partir de alguns blocos existentes dentro do PLDs [5]. Alguns dos exemplos de PLDs que podem ser encontrados no mercado incluem:

- Dispositivos Lógicos Programáveis Simples (SPLD)

- Arranjo Lógico Programável (PLA)
- Lógica de Arranjo Programável (PAL)
- Lógica de Arranjo Genérico (GAL)

- Dispositivos Lógicos de Alta Complexidade (HCPLD)
  - Dispositivo Lógico Programáveis Complexo (CPLD)
  - Arranjo de Portas Programáveis em Campo (FPGA)

Estes diferentes tipos de PLDs apresentam variações quanto à sua estrutura interna, sendo que os fabricantes de PLDs escolhem diferentes arquiteturas para implementação de blocos lógicos e chaves de interconexão. A quantidade de portas lógicas evidencia as diferenças entre os SPLDs e HCPLDs. Enquanto SPLDs possuem geralmente menos de 600 portas, os HCPLDs chegam a centenas de milhares. Os FPGAs possuem maior quantidade de portas dentre os vários PLDs, o que permite acomodar designs maiores e mais complexos em seu chip.

2.1 Arranjo Lógico Programável (PLA)

Os PLAs foram os primeiros PLDs disponíveis no mercado. Sua estrutura interna é baseada na ideia de que as funções lógicas podem ser realizadas na forma de soma de produtos e, por isso, uma PLA consiste em um conjunto de portas lógicas AND e OR. As entradas de um PLA passam através de uma série de buffers, que fornecem o valor verdadeiro da entrada ou seu complemento para um bloco chamado de plano, ou arranjo AND, que realiza os produtos das entradas. Estes produtos servem como entradas para o plano OR, que faz a soma dos produtos da entrada, gerando assim, uma soma de produtos na saída da PLA.

O PLA é eficiente quanto à área necessária para implementação em CI e, por isso, é frequentemente empregado como parte de chips maiores, como

microprocessadores, e neste caso, um PLA é feito de tal forma que as conexões às portas AND e OR são fixas, mais do que programáveis [6]. A Figura 1 mostra o diagrama esquemático de portas lógicas de um PLA. As marcações em "*X*" representam os pontos de conexão.

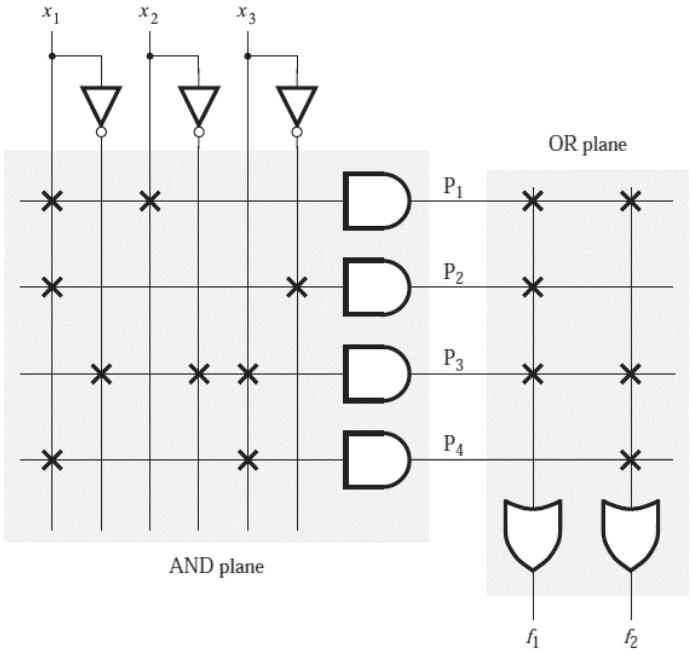


Figura 1: Diagrama de portas de um PLA. Fonte: [6]

2.2 Lógica de Arranjo Programável (PAL)

Em uma PLA tanto o plano AND e OR são programáveis, o que trazia dificuldades para os fabricantes fazê-los corretamente e com uma performance de velocidade reduzida. Esta ocasião favoreceu o desenvolvimento de um dispositivo semelhante no qual o plano AND é programável, porém, o plano OR é fixo. Este chip é chamado de Lógica de Arranjo Programável (PAL), e possui uma fabricação mais simples e uma melhor performance, o que fez com que estes dispositivos se tornassem populares. Em comparação com PLA, estes dispositivos são menos flexíveis e, para

compensar, geralmente são fabricados em uma faixa de tamanhos, com alto número de entradas e saídas [6].

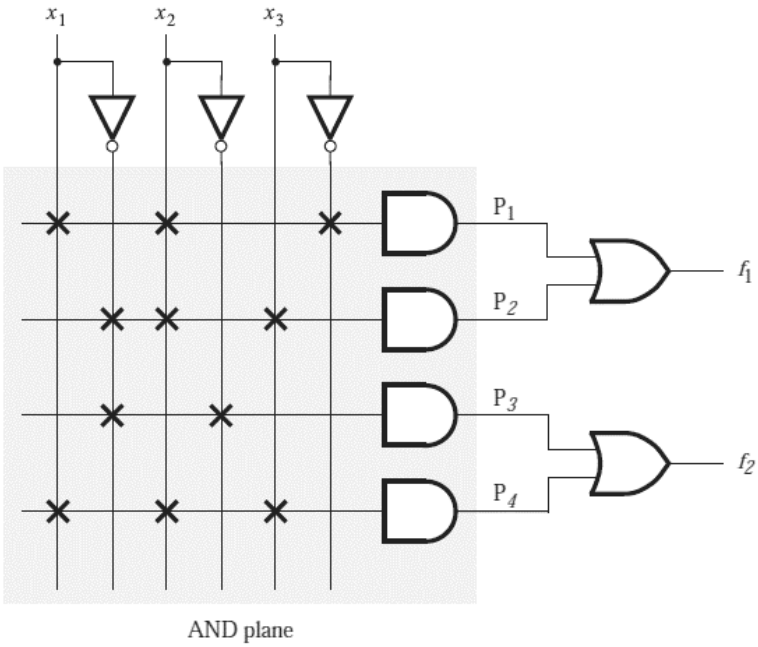


Figura 2: Exemplo de PAL. Fonte: [6]

2.3 Arranjo de Lógica Genérico (GAL)

Em dispositivos do tipo GAL, há uma evolução na arquitetura com relação aos PLDs citados anteriormente, pois eles contêm uma macro célula de lógica de saída programável (OLMC , O*utput Logic Macrocell*), que é formada por flip-flops que podem implementar a função de registrador e/ou contador, buffers para as saídas e multiplexadores para seleção de modos de operação. Outra vantagem, é que CIs GAL usam matriz programável EEPROM para realizar as conexões das portas nos planos AND/OR, podendo ser programadas ao menos cem vezes. Posições específicas na matriz EEPROM são usadas para controlar as conexões programáveis [1].

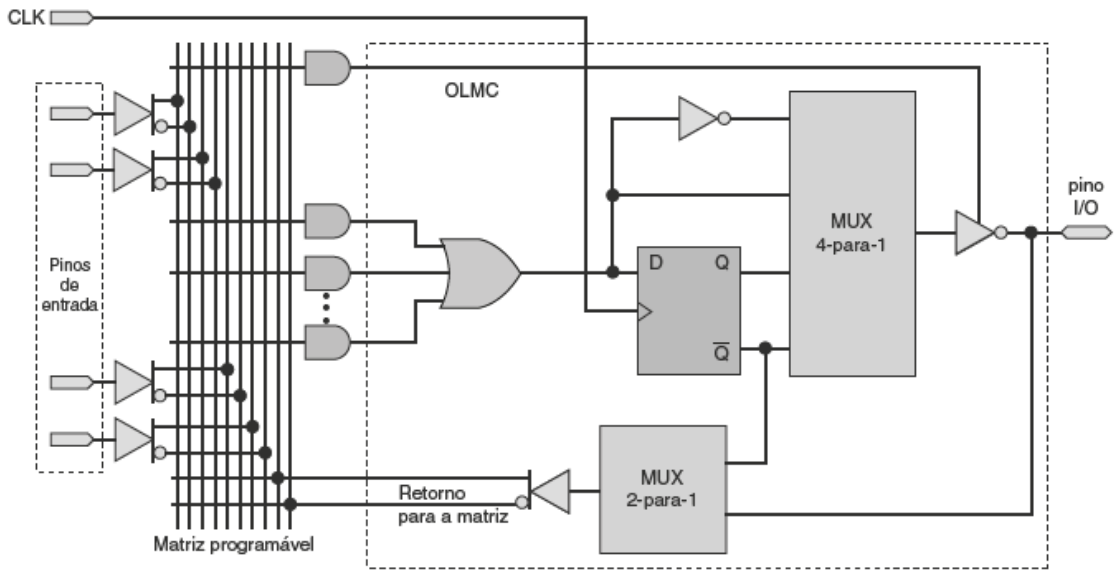


Figura 3: Estrutura Interna de um GAL. Fonte: [1]

2.4 Dispositivo Lógico Programável Complexo (CPLD)

Os SPLDs vistos são úteis para implementação de diversos circuitos digitais que não requerem muito mais que do que 32 entradas e saídas quando combinadas. Quando é necessário um circuito com mais entradas e saídas, são empregados múltiplos PLAs ou PALs, podendo comprometer a área da PCI ou até mesmo a performance do projeto [5]. Nestes casos, pode ser utilizado um CI mais sofisticado, chamado Dispositivo Lógico Programável Complexo, CPLD. Os CPLDs foram introduzidos no mercado pela Altera Corporation em 1983, sendo inicialmente chamado de Dispositivo Lógico Programável Apagável (EPLD, *Erasable Programmable Logic Device*) e, posteriormente, como CPLD [4].

Os CPLDs podem ser programados e reprogramados pelo usuário, possuem um alto desempenho, baixo custo e uma alta capacidade de integração, podendo ser utilizado para aplicações como Máquinas de Estado Finitas (FSMs, *Finite State Machines*) decodificadores de sinais, entre outros. Algumas vantagens com relação aos ASICs tradicionais são a reprogramabilidade, o uso da tecnologia CMOS

proporcionando menor consumo de energia, integração em larga escala e tempo de desenvolvimento reduzido [4].

Um CPLD pode conter de 2 à 100 blocos de circuitos semelhantes aos PLA ou PAL em um único chip. Os blocos são interconectados por uma Matriz de Chaves Programáveis ou Arranjo de Interconexão, permitindo a conexão de todos os blocos de CPLDs [2]. A Figura 4 mostra a estrutura interna de um CPLD.

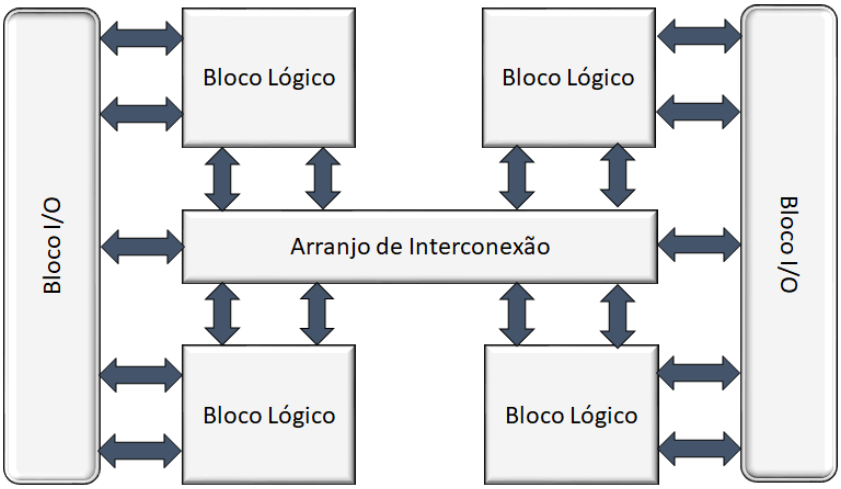


Figura 4: Estrutura Interna de um CPLD. Fonte: O Autor

2.5 Field Programmable Gate Array (FPGA)

Um FPGA (*Field Programmable Gate Array*, em português “Arranjo de Portas Programáveis em Campo”) é um dispositivo semicondutor que executar uma função que pode ser programada pelo designer ou consumidor após a sua fabricação e que suporta implementação de circuitos relativamente grandes devido sua alta capacidade lógica, destacando-se com relação aos outros PLDs [2]. É uma tecnologia de ponta quando falamos de implementação de circuitos digitais. Foi lançado em 1985 pela empresa Xilinx Inc., e possui ampla versatilidade para implementação, devido uma arquitetura multinível com grande número de portas lógicas, proporcionando maior integração entre os blocos lógicos configuráveis [7].

Alguns outros fabricantes são a Altera (comprada recentemente pela Intel), Cypress, QuickLogic, Actel, e Lattice Semiconductor. Sua arquitetura reconfigurável distingue-se dos processadores usuais por não processar funções com tarefas realizadas em processamento sequencial ao longo do tempo de execução. Os FPGAs executam processamento em paralelo de diversas unidades funcionais, reduzindo o tempo de resposta, aumentando o desempenho de execução das instruções e permitindo que a capacidade computacional da máquina seja customizada de acordo com a aplicação [8].

2.5.1 Arquitetura Interna

Sua estrutura básica é composta por circuitos de dispositivos semicondutores que se dividem em três componentes principais: Blocos lógicos configuráveis (CLBs), Blocos de entrada e saída (IOBs), e Chaves de interconexão (IS) programáveis, que são estabelecidos entre as linhas e colunas que passam ao redor do blocos, como mostra a Figura 5. Os blocos I/O fazem a interface de controle entre os pinos de entrada e saída e os sinais internos, e as chaves de interconexão fazem arranjos de caminhos corretos para conectar as entradas e saídas dos CLBs e IOBs [2]. Geralmente, a lógica combinacional de células lógicas, pode ser implementada através de uma memória chamada *LookUp Table* (LUT). Uma LUT opera como uma tabela- verdade, no qual a saída é programada para armazenar apenas um valor lógico, 0 ou 1 de acordo com cada combinação da entrada.

Um FPGA típico pode conter dezenas de milhares de CLBs e um número maior ainda de flip-flops. Projetos complexos são criados através da combinação destes blocos básicos para criar o circuito desejado.

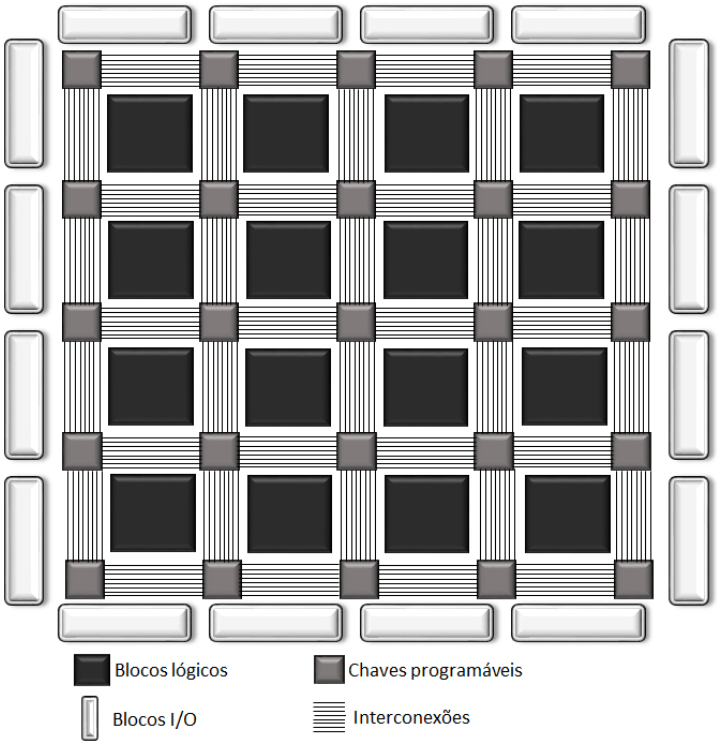


Figura 5: Estrutura Interna de um FPGA. Fonte: O Autor

Atualmente, há quatro categorias principais de FPGAs disponíveis comercialmente: Arranjo simétrico (*symmetrical array)*, Baseado-em-linhas (*row-based*), hierarquia PLD (*PLD hierarchical)*, e mar de portas (*sea of gates*) [2]. Estas categorias podem ser vistas na Figura 6. Presentemente, há também quatro tecnologias em uso, células RAM estáticas, antifusíveis, transistores EPROM e transistores EEPROM. FPGAs de RAM estática são os mais comuns. Eles perdem sua programação assim que são desenergizados, por não conter uma memória no CI. FPGAs programáveis baseados em EPROM, não podem ser reprogramados e necessitam que sua memória seja limpa com apagamento via ultravioleta (UV). Já os chips baseados em EEPROM podem ser apagados eletricamente, mas no geral, não são reprogramados em circuito.

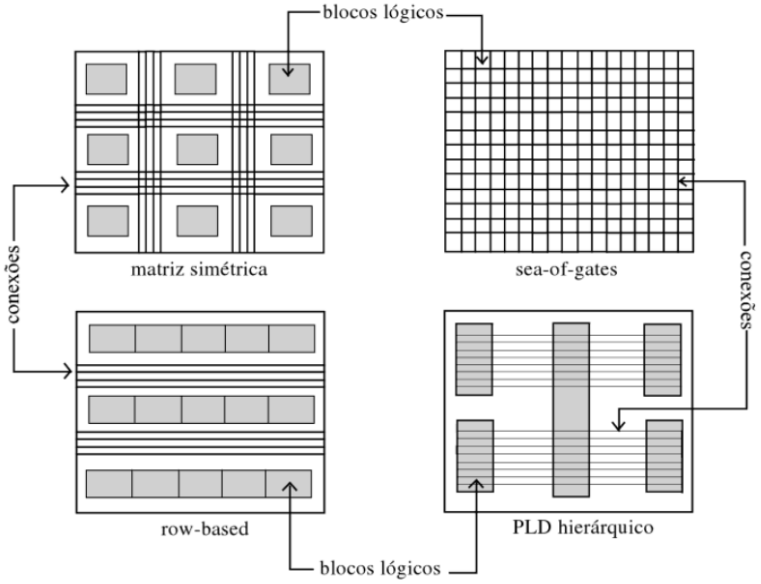


Figura 6: Categorias de FPGAs disponíveis. Fonte: [9]

Um fator importante que deve ser levado em consideração, é a existência de blocos de propriedade intelectual (IP, *intelectual property*), que se refere à projetos pré-construídos de blocos digitais, fornecidos pelos fabricantes ou por empresas do ramo. Estes blocos podem executar diversas funções e blocos de sistemas digitais de diversas aplicações, poupando tempo de desenvolvimento ao possibilitar a reutilização da lógica. Alguns tipos de blocos IPs disponíveis para uso em FPGAs incluem processadores embutidos, blocos de construção de DSP (*Digital Signal Processing*), blocos de streaming de dados, memórias RAM, memórias FIFO (*First-In First-Out*) circuitos de interfaceamento, PLLs (*Phase-Locked Loops,* malhas de fase-fechada), entre outros.

2.5.2 Linguagem de Descrição de Hardware

Uma Linguagem de Descrição de Hardware (HDL) é um código que descreve a estrutura ou o comportamento de um circuito eletrônico, no qual a conformidade

física do circuito pode ser inferida através de um compilador. Esta característica difere uma HDL de uma linguagem de programação de computadores. Suas aplicações incluem síntese de circuitos digitais em chips CPLD/FPGA e layout e geração de máscara para fabricação de ASIC [10]. Atualmente, há diferentes linguagens HDL no mercado, mas apenas algumas delas são padronizadas pela IEEE como VHDL, Verilog (IEEE 1364) sendo utilizadas em meios industriais e acadêmicos, cada uma com suas particularidades e diferentes níveis de abstração, mas ambas oferecem funcionalidades semelhantes.

2.5.3 Linguagem VHDL

VHDL é uma sigla para VHSIC (Very High Speed Integrated Circuits) Hardware Description Language e foi criada como iniciativa do departamento de defesa dos Estados Unidos nos anos 1980. A primeira versão foi o VHDL 87, sendo posteriormente atualizada para VHDL 93, VHDL 2002 e por último VHDL 2008. Foi a primeira linguagem HDL padronizada pela IEEE pelas normas 1076 e 1164.

O VHDL permite a síntese e simulação de circuitos. Síntese é a tradução do código fonte em uma estrutura de hardware que implementa a funcionalidade desejada, e a simulação é um procedimento de teste para garantir que essa funcionalidade seja realmente alcançada pelo circuito sintetizado [10]. Como é suportado pela maioria das organizações que oferecem tecnologia de hardware digital, o VHDL fornece uma portabilidade e reuso do design [6]. A portabilidade é uma importante vantagem a ser considerada ao permitir que a tecnologia do circuito digital mude rapidamente.

Um código VHDL consiste em 3 partes principais. LIBRARY, ENTITY e ARCHITECTURE. A LIBRARY é a declaração das bibliotecas que serão utilizadas

no projeto. ENTITY contém a descrição das portas (pinos) do circuito e a ARCHITECTURE, descreve como o circuito deve funcionar.

2.5.4 Processo de Design

O processo de design em FPGA consiste em algumas etapas que são realizadas pelo projetista com o auxílio de um software de design assistido por computador, CAD (*computer-aided design*). O Quartus Prime da Intel, e o Vivado da Xilinx são exemplos de CAD para prototipagem em FPGA. Estes softwares necessitam de licenças para que todas as etapas necessárias sejam realizadas, sendo elas:

a. **Entrada do Design**. É o primeiro passo no projeto utilizando o CAD. Nesta etapa o projetista descreve o circuito que será implementado de acordo com as especificações e requisitos do projeto da forma que acha conveniente, porém a mais comum é através da escrita de um programa utilizando uma Linguagem de Descrição de Hardware (HDL), em VHDL ou Verilog, por exemplo, no qual o software pode ser compilado usando os recursos do software para simular, sintetizar, otimizar realizar a implementação do circuito. Outros métodos podem ser utilizados nesta etapa, tais como entrar com o design através de um desenho esquemático, ou utilizando tabelas-verdade.

b. **Simulação**. Nesta etapa o design tem suas funcionalidades testadas em um ambiente de simulação. Para essa etapa, o projetista insere no ambiente de simulação as variáveis de entradas e saídas do projeto, verificando, geralmente através de formas de ondas, se as saídas estão se comportando corretamente, em quesito de lógica, mas também em restrições de tempo que ocorrem no circuito.

c. **Síntese**. O processo de síntese consiste em traduzir a descrição de hardware programada na entrada do design em um circuito fisicamente realizável. Esta

mesma ferramenta é utilizada para otimização do circuito em termos de tempo e potência.

d. **Design Físico.** Esta é a última etapa do processo de design antes da implementação em hardware propriamente dita do CI digital. Também é conhecida como *place and route* (PAR), é nesta etapa que portas lógicas são colocadas e interconectadas à fim de completar o circuito. Uma simulação é feita após o PAR para garantir que o circuito não viola restrições temporais quando capacitâncias parasitas são adicionadas devido aos transistores e interconexões de fios são adicionadas. A Figura 7 mostra o fluxograma do processo de design descrito.

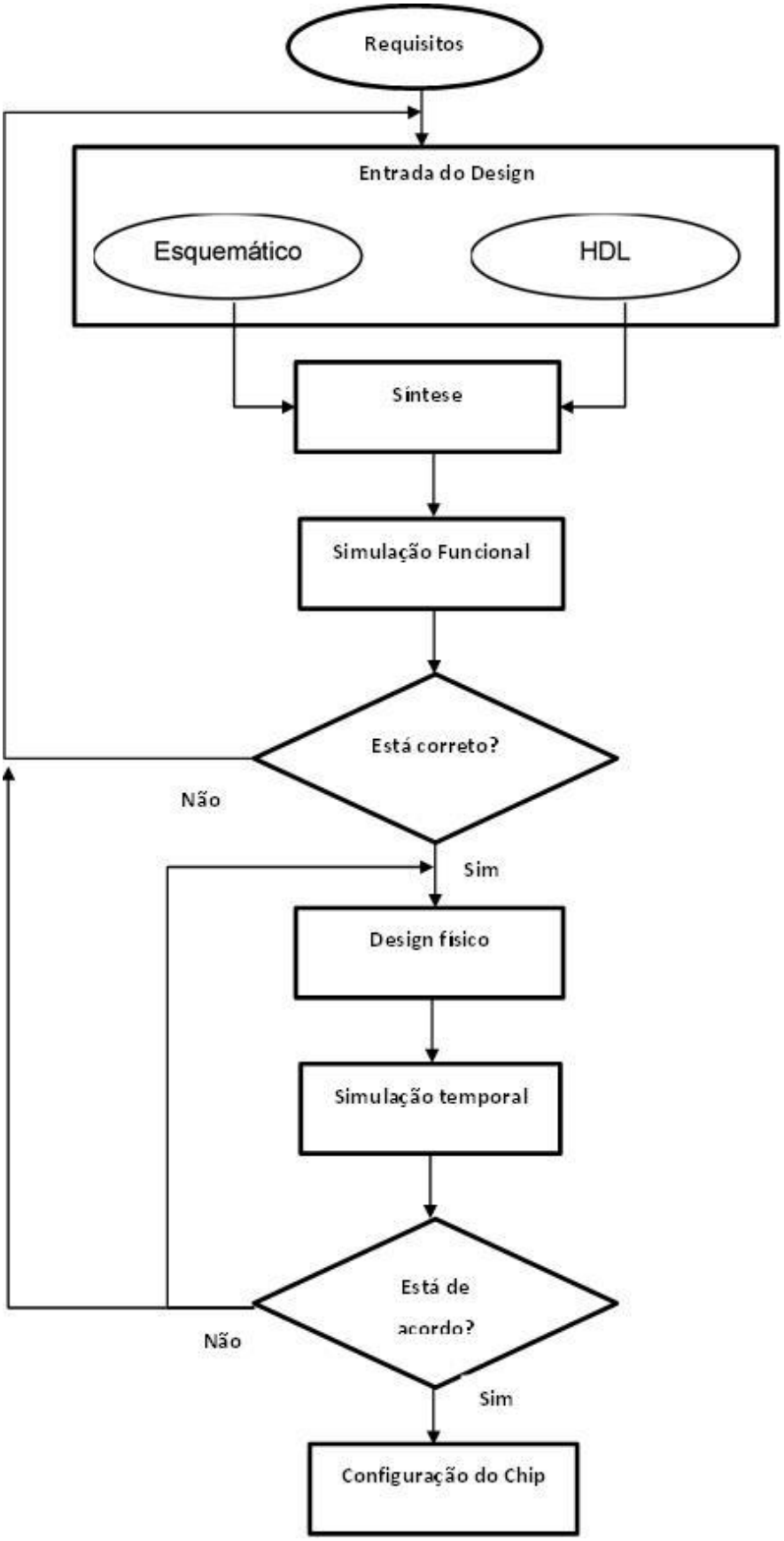


Figura 7: Fluxograma do Design em FPGA. Fonte: O Autor

2.6 Aplicações

Atualmente as áreas de aplicação para o uso de FPGAs são muito abrangentes, pois graças à sua capacidade de reconfiguração e outras vantagens como velocidade, robustez, confiabilidade, eles podem ser empregados desde pequenos circuitos digitais, como a implementação de algumas unidades de portas lógias, até a implementação de sistemas extremamente complexos, extrapolando o uso em territórios do planeta Terra, como é o caso do uso de FPGAs no Programa de Exploração da NASA em Marte [12].

Os FPGAs modernos combinam portas lógicas com processadores em único chip, conhecido como *System on Chip* (SoC) para o aumento da capacidade de computadores [13]. Como é uma tecnlogia de estado-da-arte novas implementações são desenvolvidas a cada ano, e seria muito extenso citar todas elas, embora podemos listar algumas áreas de destaque:

- Automotiva;
- Broadcasting;
- Consumo;
- Militar;
- Industrial;
- Telecomunicações;
- Redes de Comunicação;
- Computadores;
- Transportes.

3 PLATAFORMA DE DESENVOLVIMENTO

Tendo apresentado os fundamentos teóricos relativos à tecnologia utilizada neste trabalho, este capítulo aborda a plataforma e metodologia utilizada para a implementação prática dele.

À fim de obter uma alternativa economicamente viável de hardware e software para medição de pulsos de emissão espontânea de luz ultra fraca em testes realizados no LaFA (Laboratório de Fotônica Aplicada, Faculdade de Tecnologia), foi implementada a unidade de contagem foi batizada de “Multichannel Counting Unit” (MCU, Unidade de Contagem Multicanal, em português). A MCU toma como entrada 8 PMTs, fotodetectores que convertem a incidência de fótons na sua área útil em pulsos elétricos no padrão TTL (Transistor-Transistor Logic) ao longo do tempo, que são contados através de 8 contadores independentes com resolução de 32 bits cada. Os dados são enviados da MCU para um computador hospedeiro via interface serial por meio de um adaptador RS232-USB. Por fim, um script em Matlab é responsável por ler, armazenar e mostrar por meio de interface gráfica os valores de contagem enviados ao PC.

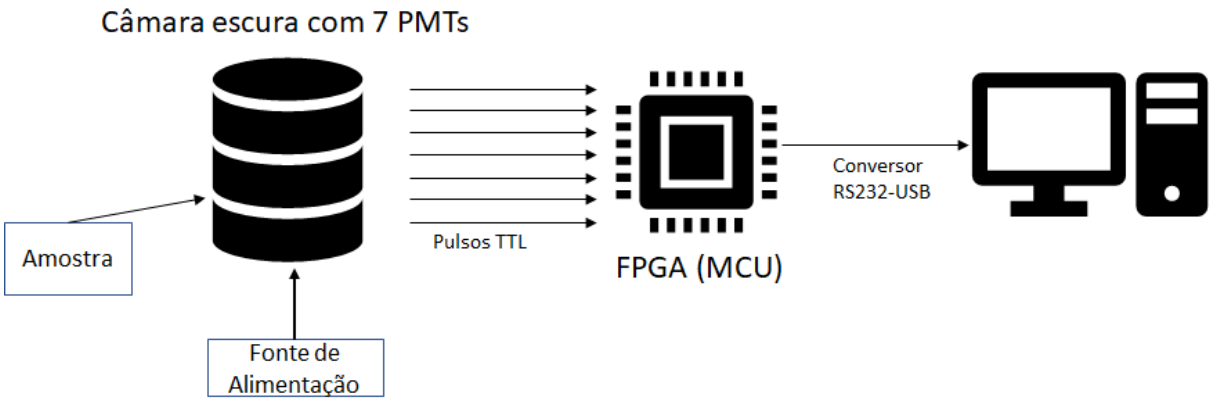


Figura 8: Diagrama esquemático do sistema. Fonte: O Autor

3.1 Placa de Prototipagem Altera DE2-115

A placa de prototipagem Altera DE2-115 contém uma ampla variedade de recursos que permitem que ela seja utilizada tanto para fins educacionais como para prototipagem de projetos, antes de que o design seja implementado em uma placa de hardware próprio. Esta placa conta principalmente com um dispositivo FPGA Cyclone IV E da Altera, que pode ser utilizado para controle de diversos periféricos e interfaces de comunicação, respondendo às necessidades de versatilidade, baixo custo, baixo consumo de energia e atendendo a demanda de vídeo, voz, dados, memória e recursos de PDS (Processamento Digital de Sinais) [13].

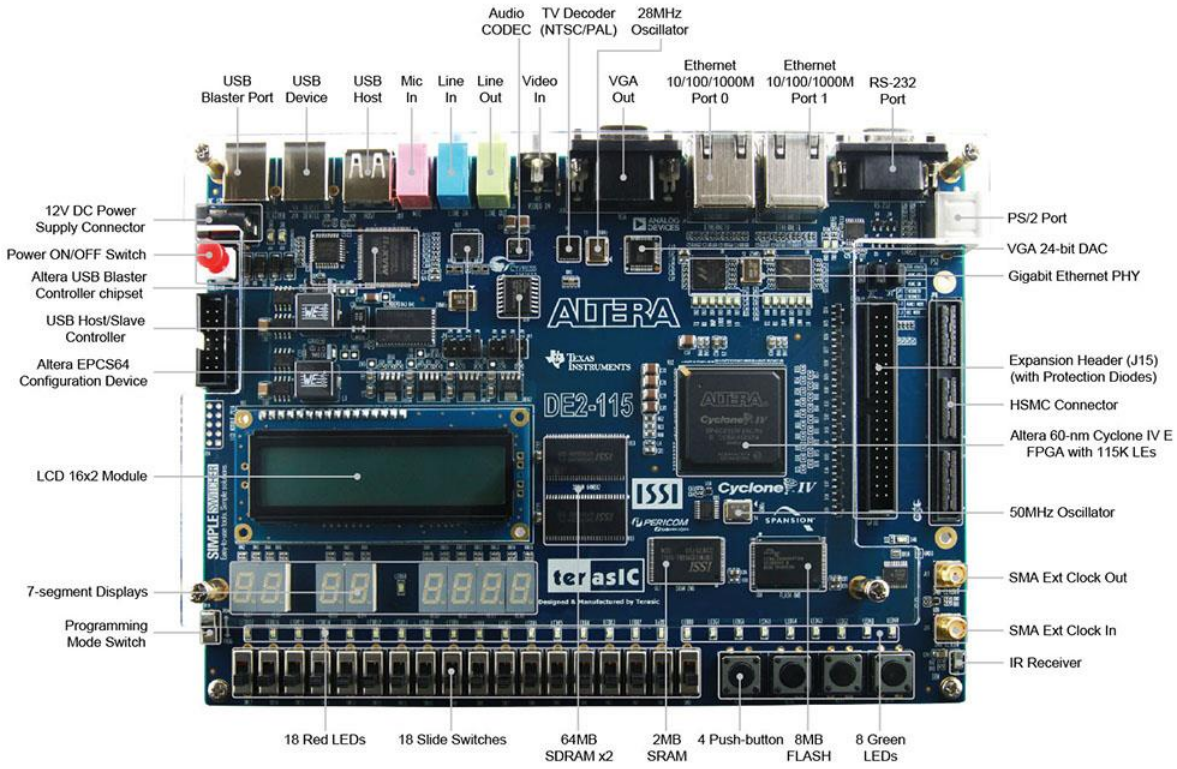


Figura 9: Layout superior da placa Altera DE2-115. Fonte: [14]

Entre os recursos presentes na Altera DE2-115, podemos listar:

- FPGA Cyclone IV EP4CE115
- Dispositivos de Memória

  - SDRAM 128 MB (32 M x 32 bits)
  - SRAM 2 MB (1 M x 16 bits)
  - Flash 8 MB (4M x 16 bits) com modo 8-bits
  - EEPROM 32 Kbits
- Chaves e indicadores
  - 18 chaves e 4 push-buttons
  - 18 LEDs vermelhos e 9 LEDs verdes
  - 8 displays de 7-segmentos
- Áudio
  - Codificador/Decodificador (CODEC) de 24-bits
  - Jacks de microfone e entrada/saída de linha
- Display
  - Módulo LCD 16x2
- Circuitos de clock
  - Três osciladores de clock de 50 MHz
  - Conectores SMA para entrada/saída de clock externo
- Suporte para cartão SD
  - Acesso ao cartão por SPI ou modo 4-bits
- Duas portas Gigabit Ethernet
  - Integração com 10/100/1000 Gigabit Ethernet
  - Suporte para IPs Ethernet industrial
- High Speed Mezzanine Card (HSMC) de 172 pinos
  - Padrões I/O configuráveis (níveis de tensão : 1.5 V/ 1.8 V/ 2.5 V/ 3.3 V)
- USB tipo A e B
  - Controlador host/device em conformidade com o padrão USB 2.0

    - Suporte a transferência em velocidade máxima e baixa velocidade
- Porta de expansão de 40-pinos (GPIO)
    - Padrões I/O configuráveis (níveis de tensão : 1.5 V/ 1.8 V/ 2.5 V/ 3.3 V)
- Conector de saída VGA
- Conector serial DB-9
    - Porta RS-232 com controle de fluxo
- Controle remoto
    - Módulo receptor Infravermelho
- Conector de Entrada de TV
    - Decodificador de TV dos padrões (NTSC/PAL/SECAM)
- Dispositivo de configuração e circuito USB-Blaster
    - Dispositivo de configuração serial EPCS64
    - Circuito na placa USB-Blaster
    - Suporte ao modo de configuração AS e JTAG
- Alimentação
    - Entrada de alimentação 12 V DC
    - Conversor regulador para 5 V/ 3.3 V/ 2.5 V/ 1.8 V/ 1.5 V/ e 1.2 V

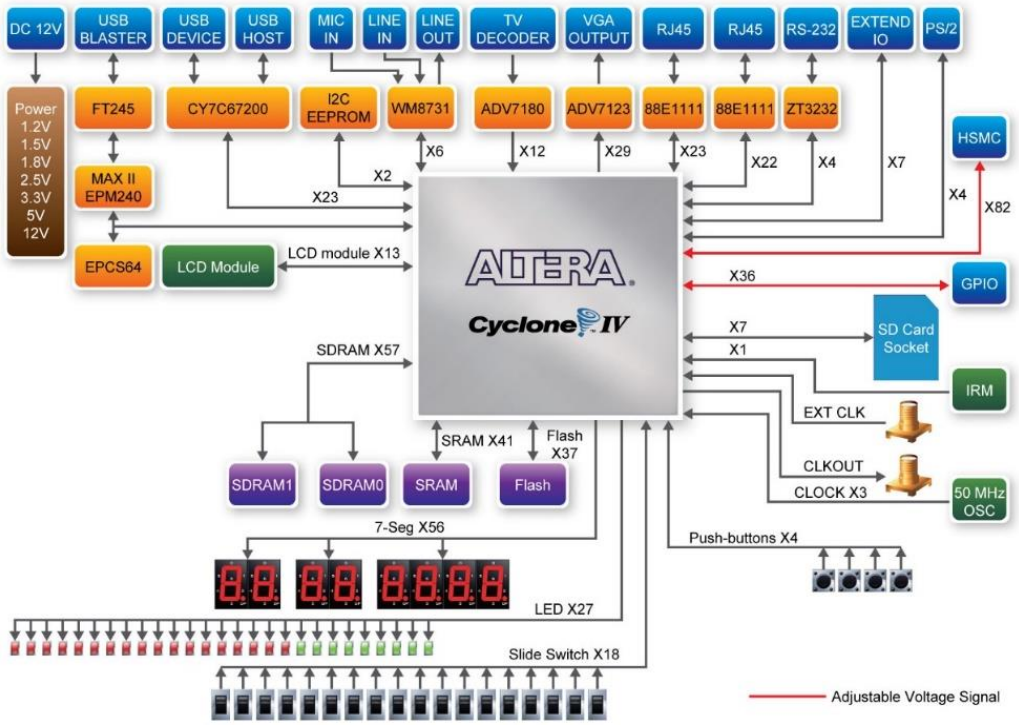


Figura 10: Diagrama de blocos da placa Altera DE2-115. Fonte: [14].

3.1.1 Família de FPGAs Cyclone da Intel

A Intel, possui uma ampla variedade de FPGA e SoCs, tendo desenvolvido e melhorado cinco séries principais: *Agilex* a mais recente série com grande velocidade e performance, *Stratix* com FPGAs mais rápidos e robustos, *Arria*, uma série que equilibra custo, potência e performance, *Cyclone*, uma série de baixo custo e boa performance e a série *MAX*, com chips únicos não-voláteis [16]. A Intel, possuí 6 famílias de FPGAs *Cyclone* até o momento, sendo: *Cyclone, Cyclone II, Cyclone III, Cyclone IV, Cyclone V* e *Cyclone 10.* Cada nova família proporciona aos projetistas maior integração, melhor desempenho e consumo e custos menores.

Em dispositivos *Cyclone*, as funções lógicas são implementadas em elementos lógicos (LE, *logic elements*), que contêm uma LUT (*Look-Up Table*) baseada em

SRAM de quatro entradas e um registrador programável, flip-flops. OS LEs são agrupados em LABs (bloco de arranjo lógico) e recursos de roteamento de sinais.

A família possui diferentes dispositivos que permitem aos projetistas escolherem entre 3 mil e 150 mil LEs. Como foi utilizado a placa Altera DE2-115, que contém o FPGA Cyclone IV EP4CE115, este é o dispositivo alvo deste trabalho. A Tabela 1 mostra a comparação entre a quantidade de recursos deste FPGA com outros da mesma família, *Cyclone IV E*.

Tabela 1: Família de FPGAs IV E da ALTERA. Fonte: []

Recursos	**Família Cyclone IV E**							
	EP4CE6	EP4CE10	EP4CE15	EP4CE30	EP4CE40	EP4CE55	EP4CE75	EP4CE115
LEs	6.272	10.320	15.208	28.848	39.600	55.856	75.408	114.800
Blocos RAM M9k	30	46	56	66	126	260	305	432
RAM total (Kbits)	270	414	504	594	1.134	2.340	2.745	3.888
Multiplicadores Embutidos	15	23	56	66	116	154	200	266
PLLs	2	2	4	4	4	4	4	4
Número máx. de I/Os	182	182	346	535	535	377	429	531

3.2 Descrição Funcional do Sistema

As PMTs são a fonte dos sinais. Foram utilizados estes detectores com aplicação destinada à detecção em uma faixa espectral específica estudada pela biofotônica, mas qualquer fonte de pulsos elétricos pode ser utilizada com esta implementação. Até oito PMTs podem ser conectadas ao GPIO. Como o padrão é LVDS, são utilizados um pino para o sinal positivo e outro para o aterramento e integridade do

sinal. Uma interface adaptadora conecta os cabos das PMTs, que possuí conectores BNC (Bayonet Neill Concelman), à um cabo flat de 40 vias para que possa ser plugado no GPIO da placa DE2-115. Oito chaves permitem que o usuário ligue ou desligue o sinal das PMTs à oito canais independentes (A à H) presentes na lógica.

Devido à natureza da largura de pulso das PMTs, fez se necessário elevar à frequência do clock para que todos os pulsos pudessem ser amostrados sem perdas. De acordo com o critério de Nyquist para amostragem de sinais, temos:

$$f_{max} = 2B \tag{1}$$

Onde f_{max} é a frequência máxima de amostragem do sinal, e B é a largura de banda do sinal. Considerando o período dos pulsos das PMTs utilizadas, temos:

$$B = \frac{1}{T} = \frac{1}{9\ x\ 10^{-9}\ s} = \ 50\ x\ 10^{6}\ s^{-1} \tag{2}$$

Logo a frequência necessária para fazer a amostragem de todos os pulsos corretamente:

$$f_{max} = 100\ x\ 10^{6}\ s^{-1} = 100\ MHz$$

Como esta placa contém dois osciladores de 50 MHz como fontes de clock, é necessário a utilização de uma das PLLs presentes para elevar a frequência do relógio. Como já existem alguns blocos IPs proprietários da Intel que podem ser utilizados através de designs com o Quartus, dentre eles o bloco de PLLs. Estas PLLs permitem obter um sinal de frequência de saída que seja um múltiplo inteiro da frequência do sinal de entrada. Para garantir mais robustez na amostragem e prover suporte a fontes de sinais ainda mais rápidos, *e.g.,* modelos mais recentes de PMTs, com período de pulsos de 10 ns, a frequência de saída escolhida foi de 200 MHz.

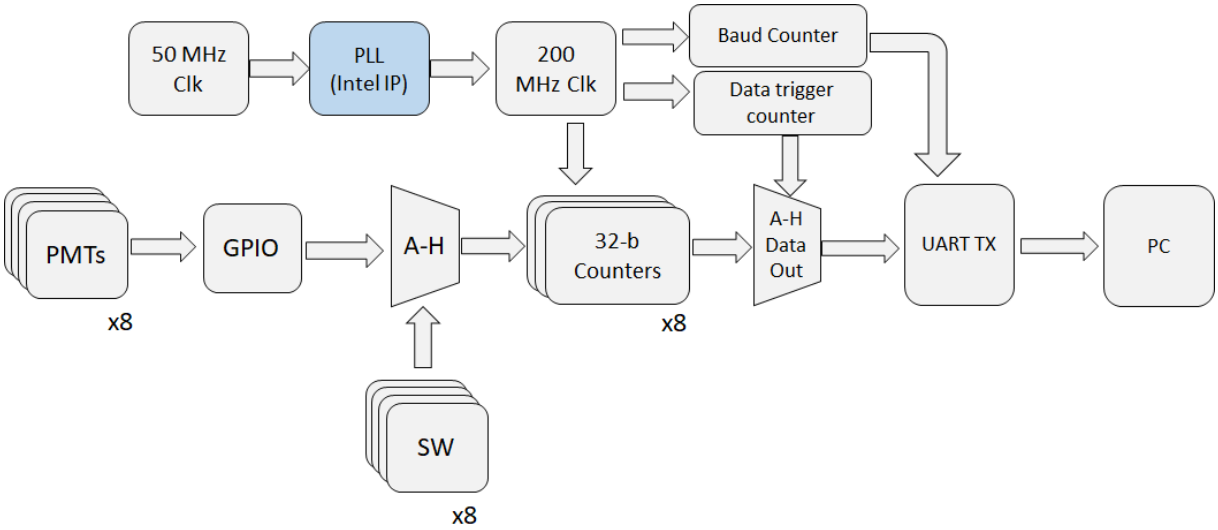


Figura 11: Diagrama de blocos da Unidade de Contagem Multicanal. Fonte: O Autor

Um parâmetro importante para circuitos contadores, é a definição do *tempo de janela*, um intervalo de tempo pré-definido no qual o mesmo irá realizar a contagem de pulsos e então limpar o contador para a próxima contagem. Aqui, é utilizado um tempo de janela fixo de 100 ms. A **Error! Reference source not found.** mostra o diagrama de blocos da MCU.

São utilizados dois contadores auxiliares: O contador de baud, responsável pela contagem de 19200 ciclos de clock, e o contador de trigger de dados.

Como o relógio do sistema possui a frequência de 200 MHz, este clock trabalha com o seguinte período:

$$T = \frac{1}{f} \tag{3}$$

$$T = \frac{1}{2 \cdot 10^8\ s^{-1}} = 5\ ns$$

Para a taxa de 19200 bits/s, temos o seguinte período de bit:

$$T_{baud} = \frac{1}{19200} = 52,08\ \mu s$$

Dividindo este resultado, pelo período de clock obtido a partir da equação (3), estimamos o valor de contagem no qual o contador de baud deve chegar:

$$\frac{52,08\ \mu s}{5\ ns} = 10416$$

Assim, o contador de baud é um contador de 14 bits que dispara um sinal de trigger ao atingir o valor de 10416, para que os dados sejam enviados na taxa de baud especificada.

O contador de trigger de dados é um contador de 15 bits responsável por a cada um décimo do baud rate escolhido de 19200 baud/s, ou seja, quando este contador alcança a contagem de 1920, um sinal de trigger torna-se ativo alto os contadores de 32 bits (essa quantidade é mais do que suficiente para contagem de pulsos na janela de 100 ms na aplicação deste trabalho) realizam a contagem a cada pulso de clock que foi detectado uma borda positiva nos sinais A-H.

Na entidade Data-Out é montado o quadro de dados contém os valores de contagem de pulsos dos oito canais e envia para a entidade UART TX para que possam ser enviados pela porta RS232. O stream de dados é iniciado a cada 100 ms e a taxa de streaming é controlada pelo baud clock de 19200 bits/s. Como é feita uma transmissão serial, é utilizado um controle de fluxo, ou *flowcontrol*, para que o computador, saiba o início e o fim de cada quadro, bem como distinguir os valores de contagem cada um dos oito canais. Para o controle de fluxo, são utilizados os seguintes parâmetros:

- Bits de dados: 8 bits, sendo 7 de dados efetivos e 1 bit de terminação;
- Stop bits: Utilizado 1 bit de start e 1 bit de stop

- Início e fim de quadro: 1 bit de *throw away*, ‘1’ que informa o início de cada quadro. Ao final do quadro, é enviado um byte de terminação para detecção do fim de quadro, sendo 8’b1.

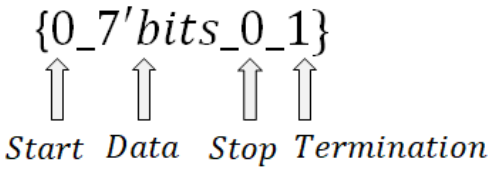


Figura 12: Divisão de palavras de um canal. Fonte: O Autor

Com essas informações podemos estimar a quantidade de bits enviados por quadro, para que possa ser projetado um buffer de entrada com tamanho suficiente para receber todos os bits de cada quadro:

- Para cada canal, são enviados 32-bits de dados;
- A cada 7 bits são enviados 3 bits de controle, totalizando 15 bits de controle por canal, conforme mostra a **Error! Reference source not found.**;
- Três bits ‘0’ são utilizados para completar a última palavra de cada canal, de forma que o canal seja enviado em 5 bytes + controle de fluxo.

  Assim temos:

$$8 \cdot \{32 + 15 + 3\} = 400\ bits$$

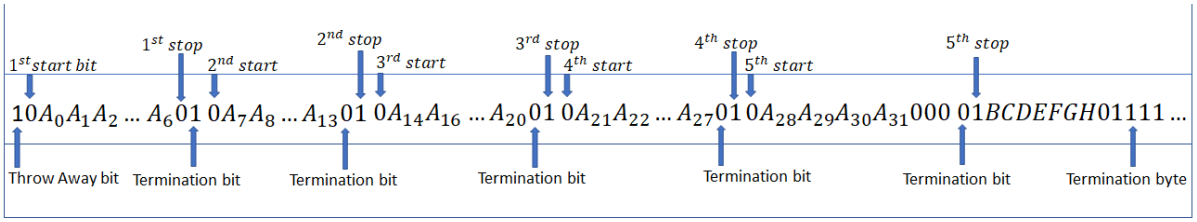


Figura 13: Representação do quadro de transmissão. Fonte: O Autor

Uma vez que a entidade DataOut é monta um quadro, eles são transferidos para a saída UART TX (Universal Asynchronous Receiver/Transmitter), que envia este streaming para o computador, à uma taxa de 10 Hz pela interface serial RS232.

O Anexo A deste trabalho contém a descrição VHDL completa deste sistema.

3.3 Script em Matlab

Foi implementado um script em Matlab para ler e armazenar os dados da MCU durante as aquisições de pulsos, proporcionando a opção de o usuário definir o tempo de aquisição e a quantidade de *loops,* ou repetições de contagem.

No script, é preciso configurar manualmente a porta COM do computador no qual a placa DE2-115 está conectada. Para cada 100 ms de tempo de janela, o programa faz o “parse”, separando os valores de leitura dos 8 canais em 8 colunas diferentes compondo uma matriz.

Ao executar o script através do botão “Run”, é solicitado para o usuário digitar o tempo de aquisição (em segundos) e em seguida, os loops que serão realizados. Por exemplo, caso o usuário escolha um tempo de 3600 s, e 2 loops, o programa irá realizar duas aquisições de 36000 amostras cada (devido ao tempo de janela de 100 ms), e irá salvar todos os valores em um arquivo de texto no formato .txt em uma pasta no diretório do arquivo. O Anexo C deste trabalho contém o programa completo escrito em matlab.

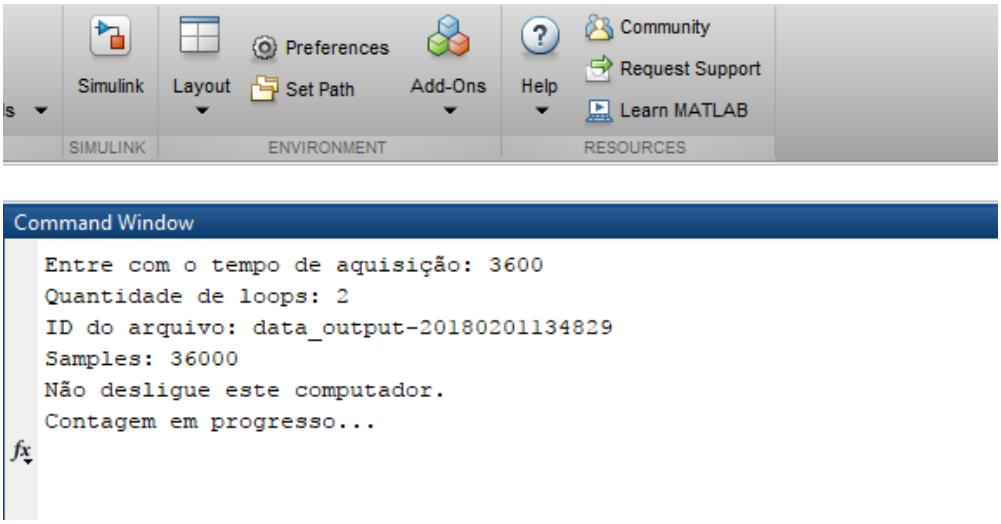


Figura 14: Janela de comandos do Matlab. Fonte: O Autor

3.4 Detectores utilizados

Fotodetectores ou sensores de luz, são dispositivos responsáveis pela conversão de energia luminosa em uma corrente elétrica mensurável. Eles podem ser classificados de acordo com seu princípio de operação, estando em três categorias principais: de efeito fotoelétrico externo, efeito fotoelétrico interno e dos tipos térmicos. O efeito fotoelétrico externo é um fenômeno no qual quando a luz incide em um metal ou semicondutor colocado no vácuo, elétrons são emitidos de sua superfície para o vácuo. Válvulas fotomultiplicadoras (*Photomultiplier tubes*, regularmente abreviadas como PMTs), fazem uso deste fenômeno e possuem elevada velocidade de resposta e sensibilidade, permitindo detecção de níveis de luz muito baixos sendo amplamente utilizadas em equipamentos médicos, instrumentos analíticos e sistemas de medições industriais [18].

Assim, uma PMT é uma válvula que consiste em uma janela de entrada, um foto-catodo, eletrodos de foco, um multiplicador de elétrons e um anodo usualmente selado em um tubo de vidro à vácuo. A Figura 15 mostra a construção de uma PMT.

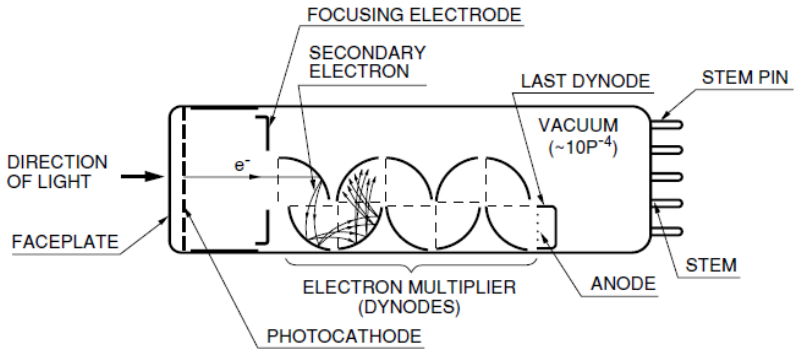


Figura 15: Construção de uma PMT. Fonte: [18]

O seguinte processo caracteriza a concepção de um sinal de saída à partir de um feixe de luz que incide na PMT: A luz passa pela janela de entrada e excita elétrons no foto-catodo, assim fotoelétrons são emitidos no vácuo, caracterizando o efeito fotoelétrico externo. Os fotoelétrons são acelerados e focalizados pelo eletrodo no primeiro dínodo, onde são multiplicados por emissão elétrica secundária. Esta emissão secundária é repetida em cada um dos sucessivos dínodos, formando o multiplicador de elétrons, até que os elétrons secundários emitidos pelo último dínodo são coletados pelo anodo [18].

3.4.1 A PMT Hamamatsu H7360

O fotodetector utilizado para realizar a contagem de pulsos deste trabalho é um módulo PMT integrado Hamamatsu H7360. Esta é uma série de dispositivos de contagem de fótons de área larga sensível com diâmetro de 25 mm, um circuito de alimentação de alta tensão, um discriminador de baixo nível (*low level discriminator*, LLD), e um modelador de pulso. Este módulo é chamado de *"Photon Counting Head".* A Figura 16 mostra o diagrama de blocos do módulo PMT utilizado.

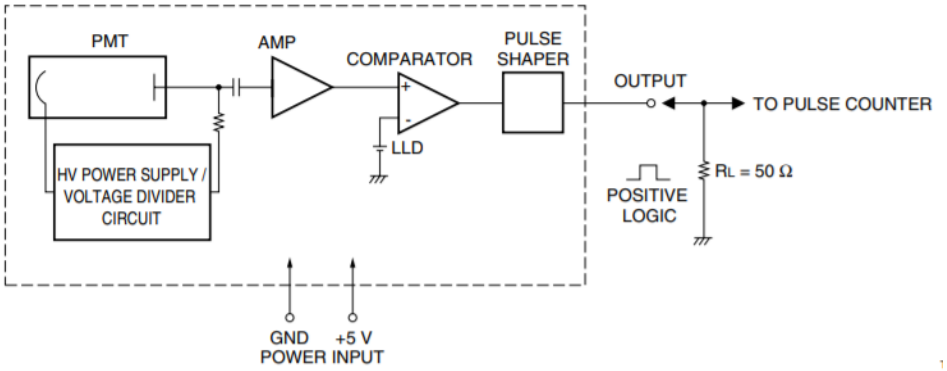


Figura 16: Diagrama de blocos da PMT Photon Counting Head. Fonte: [18]

O sistema implementado neste trabalho, cumpre o papel de substituir o contador de pulsos convencional entregue junto à este módulo, suprindo esta necessidade com mesma precisão e robustez.

Pulsos de corrente oriundos do fotomultiplicador, são amplificados no estágio amplificador, e apenas os pulsos que passam um certo valor de limiar são discriminados pelo comparador e convertidos em pulsos de tensão pelo modelador de pulsos para a saída.

A impedância de saída é projetada para aproximadamente $50\ \Omega$. Uma impedância de entrada do circuito contador de pulsos externo que não esteja compatível com este valor, causará reflexões dos pulsos, resultando em contagens errôneas. Para uma impedância de entrada que esteja casada com este valor de $50\ \Omega$, faz com que a amplitude do sinal seja metade da terminação. Como a saída é configurada para o nível lógico TTL (*Transistor-transistor logic*, 0 – 5 V), em terminações casadas, temos aproximadamente 2,5 V de tensão.

Este dispositivo é capaz de operar em uma alta taxa de contagem. A faixa de resposta espectral abrange comprimentos de onda entre 300 nm e 650 nm, conforme mostra Figura 17 .

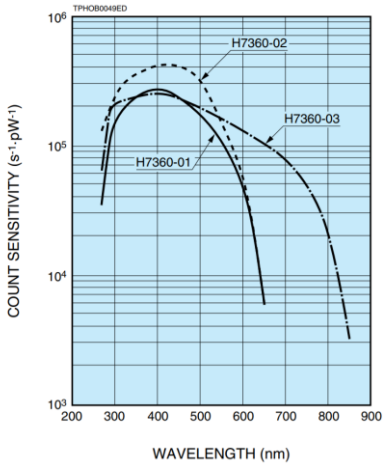


Figura 17: Resposta espectral PMTs H7360. Fonte: [17]

Neste trabalho foi utilizado o modelo H7360-01, por ser de baixo ruído, mas há também os modelos H7360-02 para detecção de alta eficiência, e H7360-03 para detecção no espectro visível e próximo ao infravermelho [17]. A Figura 18 mostra a PMT Hamamatsu H7360.

Figura 18: PMT Hamamatsu H7360. Fonte: [17]

3.5 Interface Adaptadora

Para que as PMTs pudessem ser conectadas à placa DE2-115, foi necessário a construção de uma pequena estrutura adaptadora, para que os cabos de sinal das PMTs permanecessem fixos em uma estrutura evitando interferências e ocasionando ruídos.

Materiais necessários:

- Um cabo “flat” de 40 vias com um conector fêmea para terminação na placa DE2-115. Vinte centímetros de cabo são suficientes, evitando perdas por distância;
- 8 terminações BNC fêmea com base redonda;
- 1 placa de acrílico (cerca de 150 cm²);
- 1 cabo RS232, com conversor USB para conexão da placa DE2-115 ao computador;
- Tesoura
- Furadeira.

Em uma das extremidades do cabo flat, com o auxílio da tesoura, os fios foram separados individualmente e desencapados para que fossem conectados às terminações BNC;

Utilizando a furadeira, foram feitos 8 furos de 1 cm de diâmetro na placa de acrílico. As terminações BNC são inseridas nestes furos. Soldar os fios do cabo flat nas terminações BNC. Feito isso às PMTs podem ser conectadas à DE2-115.

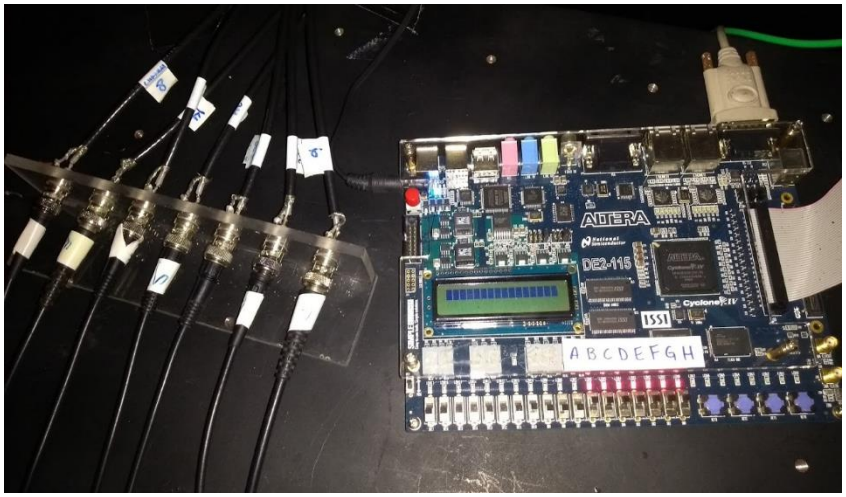


Figura 19: Placa Altera DE2-115 e interface adaptadora construída. Fonte: O Autor

4 RESULTADOS E DISCUSSÃO

Uma forma de avaliar os resultados referentes à implementação de sistemas digitais utilizando FPGAs pode ser realizada através da análise da quantidade de recursos utilizados após a implementação do circuito. Para a MCU tivemos os seguintes recursos utilizados:

- 326 registradores;
- 46 dos 529 pinos físicos disponíveis (8,70%)
- 634 elementos lógicos (< 1% da quantidade de LE disponíveis na Cyclone IV E115);

Percebe-se que foram consumidos recursos mínimos em relação à quantidade disponível no FPGA deste kit de desenvolvimento. Isso implica que o sistema pode receber melhorias, e muitas outras funcionalidades futuras podem ser incorporadas ao projeto, proporcionando maior versatilidade. Dentre as funcionalidades que podem ser incluídas podemos citar:

- Disponibilização de múltiplas fontes de clock: Como são disponíveis 4 recursos de PLL’s neste FPGA, poderiam ser incluídos saídas de clock em 200 MHz, 300 MHz e 400 MHz (limite de frequência), permitindo que o conte com precisão pulsos com largura de até 5 ns;
- Possibilidade de escolha de diferentes tempos de janelas: Atualmente, o sistema trabalha com um tempo de janela fixo em 100 ms. Através de um registrador, ou utilizando chaves físicas, ou alguma implementação diferente, o sistema pode permitir ao usuário que ele escolha o tempo de janela variável, em diferentes intervalos, como 10 ms, 100 ms, 1 s, 10 s, e entre outros, tornando o intervalo de contagem mais flexível;

- Controle de obturador: Não foram utilizados obturadores neste projeto, mas os mesmos poderiam ser incluídos assim como lógica de controle, permitindo contagem múltipla sequencial (com escolha de tempo de obturador aberto e um tempo de intervalo entre aberturas do obturador) ou contínua (através de lógica que permita o chaveamento de contadores);
- Comunicação com sensores: As PMTs ficam em um ambiente controlado, como uma sala escura, diferente da sala de controle onde o usuário permanece durante a aquisição. Podem ser incluídos sensores de temperatura, umidade e iluminação permitindo ao FPGA receber informações do ambiente e que seja programada lógica para que tome decisões e atue sobre os dispositivos conectados à ele. Sensores de tensão e corrente elétrica também podem medir essas grandezas permitindo um controle de segurança sobre as PMTs.

O script do Matlab foi utilizado para configuração das aquisições. Uma interface gráfica mostra a aquisição de até 8 canais em sub-janelas. Para facilitar a compreensão, o gráfico é atualizado uma vez por segundo, realizando a integração de 10 pontos do tempo de janela de 100 ms, facilitando o monitoramento da aquisição por parte do usuário. No próprio Matlab, ou em outro software também é possível incluir diversas outras funcionalidades, como implementação de cálculos estatísticos, como média e variância por exemplo.

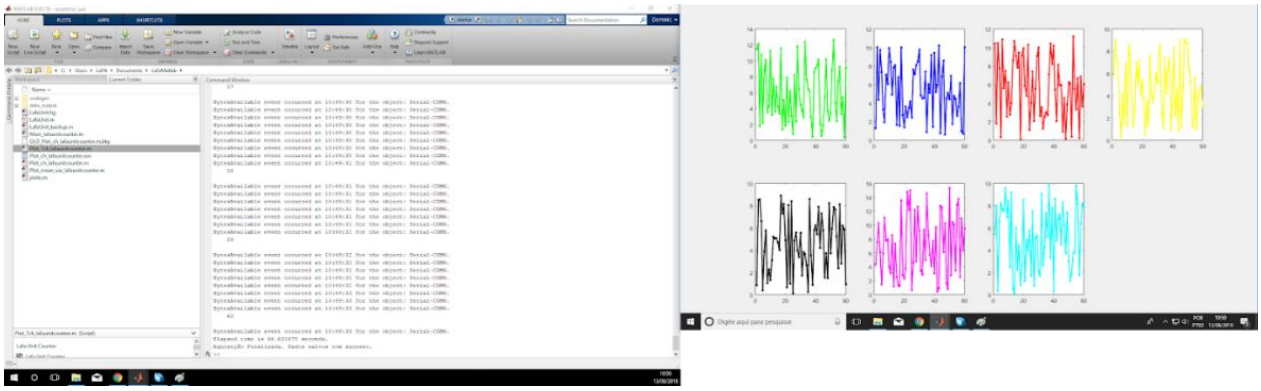


Figura 20: Interface com subjanelas mostrando contagem de cada canal. Fonte: O Autor.

O desempenho do sistema pôde ser validado realizando aquisições de contagem escura. Essa é uma medida que distingue fotodetectores dos demais detectores. Essa taxa de contagem escura é a taxa média de contagens registradas sem uma fonte de luz incidente. Assim é determinado a taxa de contagem mínima na qual o sinal é predominantemente causado por fótons reais. Essa detecção tem sua principal origem no ruído térmico, e este efeito pode ser amenizado através de resfriamento ou utilizando detectores arrefecidos.

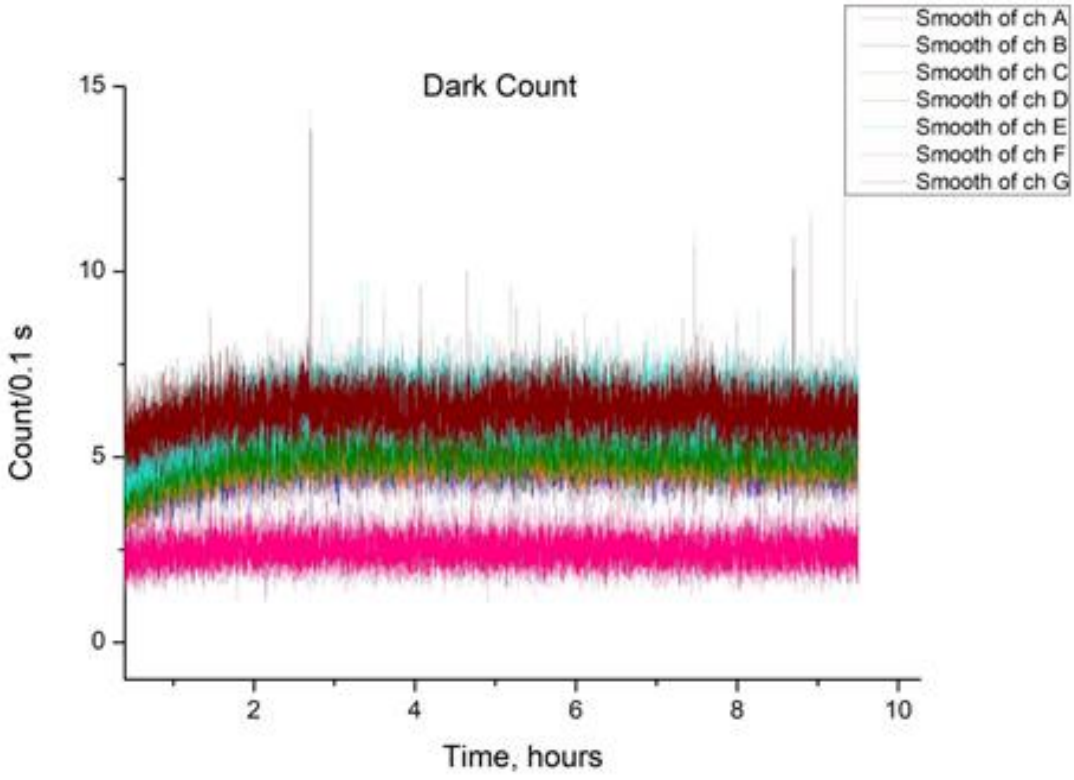


Figura 21: Medição de contagem escura utilizando a MCU. Fonte: O Autor

Através desta caracterização de contagem escura realizada ao longo de 9,5 horas, percebe-se que a taxa média de contagem é de aproximadamente 5 fótons/ 0.1 s. Uma das PMTs apresentou uma taxa de contagem de ~2,5 fótons/0.1 s, indicando ser de um modelo menos suscetível ao ruído térmico.

5 CONCLUSÃO

Foi apresentado um sistema digital de baixo custo para aquisição de pulsos multicanal, utilizando um kit de desenvolvimento Altera DE2-115 com FPGA Cyclone IV, com Hardware descrito em linguagem VHDL. Embora no setup sejam utilizados 7 detectores, esta versão do sistema suporta até 8 detectores para contagem, podendo ser expandido, e estando limitado apenas pela quantidade de pinos disponíveis no GPIO da placa.

Com o programa do Matlab, pode-se ler visualizar os dados em tempo real, sendo plotado a somatória dos pontos obtidos ao longo do tempo de aquisição, e salvar os dados em um diretório de preferência do usuário.

Também foram discutidas algumas implementações futuras de funcionalidades, que podem ser por exemplo, temas de iniciação científica de alunos de graduação que se interessam por esta área.

Espera-se que com este projeto sejam realizados diversos experimentos relacionados à fotônica aplicada e que isso gere frutos positivos para a pesquisa realizada no LaFA.

6 REFERÊNCIAS BIBLIOGRÁFICAS

[1] TOCCI, Ronald J.;WIDMER, Neal S; MOSS, Gregory L. **Sistemas digitais: princípios e aplicações.** Revisão técnica: Renato Giacomini. Tradução Jorge Ritter . 11ª. ed. – São Paulo. Pearson Prentice Hall, 2011. ISBN 978-85-4300-694-9

[2] FERDJALLAH, Mohammed. **Introduction to digital systems: modeling, synthesis, and simulation using VHDL.** Hoboken, NJ: John Wiley & Sons, 2011. ISBN 978-0-470-90055-0

[3] MOORE, Gordon E. **Cramming more componentes onto integrated circuits Electronics.** Electronics Magazine, Volume 38, Number 8. April 19, 1965. Retirado de:<https://web.archive.org/web/20090126170054/http://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf>Acesso em: 25/08/2019

[4] OLIVEIRA, Caio Augusto; AGUIAR, Jéssica Azevedo; FONTANINI, Mateus Galvão Said. **Apostila: Dispositivos Lógicos Programáveis.** Universidade Estadual Paulista, UNESP, Colégio Técnico Industrial de Guaratinguetá “Professor Carlos Augusto Patrício Amorim”.

[5] FERDJALLAH, Mohammed. **Introduction to Digital Systems: Modeling, Synthesis and Simulation Using VHDL.** ISBN 978-0-470-90055-0

[6] BROWN, Stephen D**. Fundamentals of digital logic with VHDL design**. 3rd ed. ISBN 978–0-07–352953–0

[7] FREIRE, Tiago Samir de Sousa**. Interfaceamento de entrada e saída em aplicações com uso de FPGA.** Monografia para obtenção do título de Engenheiro Eletricista, Universidade Federal do Ceará, Centro de Tecnologia Departamento de Engenharia Elétrica. Fortaleza, Dezembro de 2010.

[8] WEBER, André Felippe; CECHINEL, Helenluciany; THEISGES, Maria Luiza; MOECKE, Marcos**. Arquitetura FPGAs e CPLDs da ALTERA.** Disponivel em: <https://wiki.sj.ifsc.edu.br/wiki/images/2/2a/DLP29006-AE1-Tema1-2016-1.pdf> Acesso em 27/09/2019.

[9] ORDONEZ, Edward David Moreno. Pereira, Fábio Dacêncio. Penteado, Cesar Giacomini. Pericino, Rodrigo de Almeida. **Projeto, Desempenho e Aplicações de Sistemas Digitais em Circuitos Programáveis (FPGAs)** – Pompéia: Bless, 2003, 300p. ISBN: 85-87244-13-2

[10] PEDRONI, Volnei A. **Circuit Design and Simulation with VHDL** – 2nd ed. The MIT Press. Cambridge, Massachussetts. ISBN 978-0-262-01-433-5.

[11] SAWANT, Minal. **Past, Present, Future – Xilinx on Mars Rovers...!** Retirado de: <https://forums.xilinx.com/t5/Adaptable-Advantage-Blog/Past-Present-Future-Xilinx-on-Mars-Rovers/ba-p/944915> Acesso em 14/08/2019;

[12] National Instruments. **FPGA Fundamentals**. Retirado de: <http://www.ni.com/pt-br/innovations/white-papers/08/fpga-fundamentals.html> Acesso em 14/08/2019;

[13] **Altera DE2-115 Development and Education Board – Overview.** Retirado de: <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=139&No=502> Acesso em 19/09/2019.

[14] **Altera DE2-115 Development and Education Board – Layout.** Retirado de: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=139&No=502&PartNo=3#section> Acesso em 19/09/2019;

[15] **Altera DE2-115 Development and Education Board From Terasic Inc.** Retirado de: <https://www.intel.com/content/www/us/en/programmable/solutions/partners/partner-profile/terasic-inc-/board/altera-de2-115-development-and-education-board.html#board-quality-metrics> Acesso em 19/09/2019.

[16] **Intel FPGAs devices**. Retirado de: <https://www.intel.com/content/www/us/en/products/programmable/fpga.html> Acesso em 20/09/2019

[17] HAMAMATSU. **Head-on PMT. Photon Counting Head H7360 Series**. Disponível em: <https://www.hamamatsu.com/resources/pdf/etd/m-h7360e.pdf> Acesso em: 21/10/2019;

[18] HAMAMATSU. Photomultiplier Tubes. Basics and Applications. Third edition (Edition 3a). Disponível em: <https://www.hamamatsu.com/resources/pdf/etd/PMT_handbook_v3aE.pdf> Acesso em: 21/10/2019;

ANEXO A: Descrição de Hardware em VHDL

```
-- Pulse Counter
-- Multichannel Counting Unit
-- Pedro Henrique S. Oliveira – Laboratorio de Fotonica Aplicada/2017

--Top level entity
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY lafaunitcounter IS
PORT
(
-- o transmitter to RS-232 port where data are sent to Matlab
            UART_TXD    :OUT  STD_LOGIC;
-- The 50 MHz clock that is provided on the DE2 Board
            Clock_50    :IN         STD_LOGIC;

-- The 200 MHz clock generated by the PLL Component
--          Clock_200   :OUT STD_LOGIC;

-- The switchs 0 through 17 on the DE2 Board
            SW                :IN         STD_LOGIC_VECTOR(17 DOWNTO 0);

-- The input pins on 40 pin expansion header GPIO pins
-- (which can be used for input or output signals).
-- Note that the pins on the expansion header do not match the pin
assignments used by
--  Quartus II when programming the DE2-115 Board
      GPIO_3, GPIO_7, GPIO_11, GPIO_15,GPIO_21, GPIO_25, GPIO_29, GPIO_33
      :IN         STD_LOGIC ;

--The respective GND for each channel
      GPIO_4, GPIO_8, GPIO_12, GPIO_16,GPIO_22,GPIO_24,GPIO_30,GPIO_34 :
OUT  STD_LOGIC;

-- The output test signal pins on the 14 pin general purpose header
-- (which can be used for input or output signals).
-- Note that the pins on the expansion header do not match the pin
assignments used by
--  Quartus II when programming the DE2 Board
-- The red LED lights 0 through 17 on the DE2-115 Board
            LEDR        :OUT  STD_LOGIC_VECTOR(17 DOWNTO 0)
      );

END lafaunitcounter;

ARCHITECTURE Behavior OF lafaunitcounter IS
--This COMPONENT is the Megafunction "lpm_counter" using a 14 bit output
and an asynchronous clear

      COMPONENT data_trigger_counter
            PORT
            (
                  aclr  : IN  STD_LOGIC;
                  clock : IN  STD_LOGIC;
                  q     : OUT       STD_LOGIC_VECTOR (14 DOWNTO 0)
            );
      END COMPONENT;
```

```
-- This COMPONENT is the Megafunction "lpm_counter" using a 13 bit output
and an asynchronous clear
      COMPONENT baud_counter
            PORT
            (
                  aclr  : IN  STD_LOGIC;
                  clock : IN  STD_LOGIC;
                  q     : OUT       STD_LOGIC_VECTOR (14 DOWNTO 0)
            );
      END COMPONENT;

-- This COMPONENT is the Megafunction "lpm_counter" using a 32 bit output
and an asynchronous clear
      COMPONENT counter
            PORT
            (
                  aclr  : IN  STD_LOGIC;
                  clock : IN  STD_LOGIC;
                  q     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
            );
      END COMPONENT;

-- This COMPONENT takes in the single photon counts and sends it out
-- on the RS232 port, the data stream is started by data_trigger every
1/10th of a second
-- and the rate of the data_stream is controled by the 19200 bits/sec baud
clock
      COMPONENT DataOut
            PORT
            (
                  A     :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
                  B     :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
                  C     :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
                  D     :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
                  E     :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
                  F     :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
                  G     :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
                  H     :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);

                  clk               :IN         STD_LOGIC;
                  data_trigger      :IN         STD_LOGIC;
                  UART_TXD          :OUT  STD_LOGIC
            );
      END COMPONENT;


-- This component takes in the 50 MHz clock and turns it in a 200MHz clocl

      COMPONENT pll
            PORT (
                  inclk0 : IN STD_LOGIC  := '0';
                  c0     : OUT STD_LOGIC

            );
      END COMPONENT;

-- This SIGNAL counts the baud clock until it reaches 1920, which occurs
every 1/10th of a second
      SIGNAL data_trigger_count: STD_LOGIC_VECTOR(14 DOWNTO 0);
-- This SIGNAL is turned on every 1/10th of a second for one 50 MHz clock
pulse and resets
```

```
-- the photon detection counters
      SIGNAL data_trigger_reset: STD_LOGIC;
-- This SIGNAL is turned on every 1/10th of a second and begins the data
stream out
      SIGNAL data_trigger: STD_LOGIC;
-- This SIGNAL acts as a clock to output data at the baud rate of 19200
bits/second
      SIGNAL baud_rate_clk: STD_LOGIC;
-- This SIGNAL counts the 50 MHz clock pulses until it reaches 2604 in
order to time the baud clock
      SIGNAL baud_rate_count: STD_LOGIC_VECTOR(14 DOWNTO 0);
-- These SIGNALs represent the four input pulse from the photon detectors

      SIGNAL A, B, C, D, E, F, G, H: STD_LOGIC;


-- This SIGNAL represents the top level design entity instantiation of the
number of counts
-- in the detectors A, B, C, D, E, F, G, and H respectively
      SIGNAL A_top, B_top, C_top, D_top, E_top, F_top, G_top, H_top:
STD_LOGIC_VECTOR(31 DOWNTO 0);

-- This SIGNAL represents the number of counts in the detectors A, B, C,
and D respectively
      SIGNAL A_out, B_out, C_out, D_out, E_out, F_out, G_out,H_out:
STD_LOGIC_VECTOR(31 DOWNTO 0);
-- This SIGNAL is the only variable that is sent to the computer from the
program
      SIGNAL Output: STD_LOGIC_VECTOR(31 DOWNTO 0);

      SIGNAL Clock_200: STD_LOGIC;

      BEGIN
-- This initializes the input PMTs signals when the respective switches is
ON
-- Note that for this current circuit design, you might check if the data
read is correct. Use SW(15)-sw(8)
-- to send fixed values through


      WITH SW(7) SELECT
            A <= GPIO_3       WHEN '1',
                 '0'          WHEN '0';

      WITH SW(6) SELECT
            B <= GPIO_7  WHEN '1',
                 '0'          WHEN '0';

      WITH SW(5) SELECT
            C <= GPIO_11      WHEN '1',
                 '0'          WHEN '0';

      WITH SW(4) SELECT
            D <= GPIO_15      WHEN '1',
                 '0'          WHEN '0';

      WITH SW(3) SELECT
            E <= GPIO_21      WHEN '1',
                 '0'          WHEN '0';

      WITH SW(2) SELECT
```

```
            F <= GPIO_25      WHEN '1',
                 '0'          WHEN '0';

      WITH SW(1) SELECT
            G <= GPIO_29      WHEN '1',
                 '0'          WHEN '0';

      WITH SW(0) SELECT
            H <= GPIO_33      WHEN '1',
                 '0'          WHEN '0';


      --envia valores fixos conhecidos aos sinais de contagem

      -- envia GND para os pinos adjacentes aos sinais
GPIO_4   <= '0';
GPIO_8   <= '0';
GPIO_12  <= '0';
GPIO_16  <= '0';
GPIO_22  <= '0';
GPIO_24  <= '0';
GPIO_30  <= '0';
GPIO_34  <= '0';


-- Once the output of the 14 bit counter reaches 1920, this process turns
on the SIGNAL 'data_trigger'
-- The SIGNAL 'data_trigger' then acts as a clock pulse, reseting the
counts and changing the display
      PROCESS ( data_trigger_count )
            BEGIN
      -- 1920 => gate = 1/10 sec 19200 => gate= 1 sec
IF data_trigger_count = "000011110000000" THEN
                  data_trigger_reset <= '1';
                  data_trigger <= '1';
            ELSIF data_trigger_count = "000000000000000" THEN
                  data_trigger_reset <= '0';
                  data_trigger <= '1';
            ELSIF data_trigger_count = "000000000000001" THEN
                  data_trigger_reset <= '0';
                  data_trigger <= '1';
            ELSE
                  data_trigger_reset <= '0';
                  data_trigger <= '0';
            END IF;
      END PROCESS;

-- Once the output of the 13 bit counter reaches 2,604, this process turns
on the SIGNAL 'baud_rate_clk'
-- The SIGNAL 'baud_rate_clk' then acts as a clock pulse, send the data out
at the specified baud rate
      PROCESS ( baud_rate_count )
            BEGIN
--          IF baud_rate_count = "000101000101100" THEN --Clock_50
            IF baud_rate_count = "010100010110000" THEN --Clock_200
                  baud_rate_clk <= '1';
            ELSE
                  baud_rate_clk <= '0';
            END IF;
      END PROCESS;
```

```
      CLK: pll PORT MAP (Clock_50, Clock_200);

-- Uses the 14 bit counter and ~9,600 baud rate clock to count to 1/10th of
a second to trigger DataOut
      C0: data_trigger_counter PORT MAP ( data_trigger_reset,
baud_rate_clk, data_trigger_count );

-- Uses the 13 bit counter and 50 MHz clock to count the baud rate
C1: baud_counter PORT MAP ( baud_rate_clk, Clock_200, baud_rate_count );

      CA: counter PORT MAP ( data_trigger_reset, A, A_top );
      CB: counter PORT MAP ( data_trigger_reset, B, B_top );
      CC: counter PORT MAP ( data_trigger_reset, C, C_top );
      CD: counter PORT MAP ( data_trigger_reset, D, D_top );
      CE: counter PORT MAP ( data_trigger_reset, E, E_top );
      CF: counter PORT MAP ( data_trigger_reset, F, F_top );
      CG: counter PORT MAP ( data_trigger_reset, G, G_top );
      CH: counter PORT MAP ( data_trigger_reset, H, H_top );

--        counter(aclr : IN, clock : IN, q : OUT);

-- This process sets the single photon and coincidence photon count output
arrays every 1/10th of a second
      PROCESS( data_trigger_reset )
      BEGIN
            IF data_trigger_reset'EVENT AND data_trigger_reset = '1' THEN
                  A_out <= A_top;
                  B_out <= B_top;
                  C_out <= C_top;
                  D_out <= D_top;
                  E_out <= E_top;
                  F_out <= F_top;
                  G_out <= G_top;
                  H_out <= H_top;
            END IF;
      END PROCESS;

-- Sends the A, B, C, D, E, F, G e Hout on the RS-232 port
      D0: DataOut PORT MAP(A_out, B_out, C_out, D_out, E_out, F_out, G_out,
H_out, baud_rate_clk, data_trigger, UART_TXD);

-- Turns on the corresponding red LED whenever one of the DE2 board
switches is turned on
      LEDR <= SW;

END Behavior;
```

```
-- DATA out
-- This entity is the serial framer
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY DataOut IS
      PORT
      (
            A           :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
            B           :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
            C           :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
            D           :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
            E           :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
            F           :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
            G           :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
            H           :IN         STD_LOGIC_VECTOR(31 DOWNTO 0);
            clk         :IN         STD_LOGIC;
            data_trigger      :IN         STD_LOGIC;
            UART_TXD    :OUT  STD_LOGIC
      );

END DataOut;

ARCHITECTURE Behavior OF DataOut IS

SIGNAL index: STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL Output: STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL data_select: STD_LOGIC_VECTOR(2 DOWNTO 0);

BEGIN

      PROCESS( clk, index )
      BEGIN
            IF clk'EVENT AND clk = '1' THEN
                  IF index = "111111" AND data_trigger = '1' THEN
                        index <= "000000";
                        UART_TXD <= '1'; -- a throw away bit
                        OUTPUT <= A;
                        data_select <= "000";

                  ELSIF index = "000000" THEN
                        index <= "000001";
                        UART_TXD <= '0'; -- the first start bit

                  ELSIF index = "000001" THEN
                        index <= "000010";
                        UART_TXD <= Output(0);

                  ELSIF index = "000010" THEN
                        index <= "000011";
                        UART_TXD <= Output(1);

                  ELSIF index = "000011" THEN
                        index <= "000100";
                        UART_TXD <= Output(2);

                  ELSIF index = "000100" THEN
                        index <= "000101";
                        UART_TXD <= Output(3);

                  ELSIF index = "000101" THEN
```

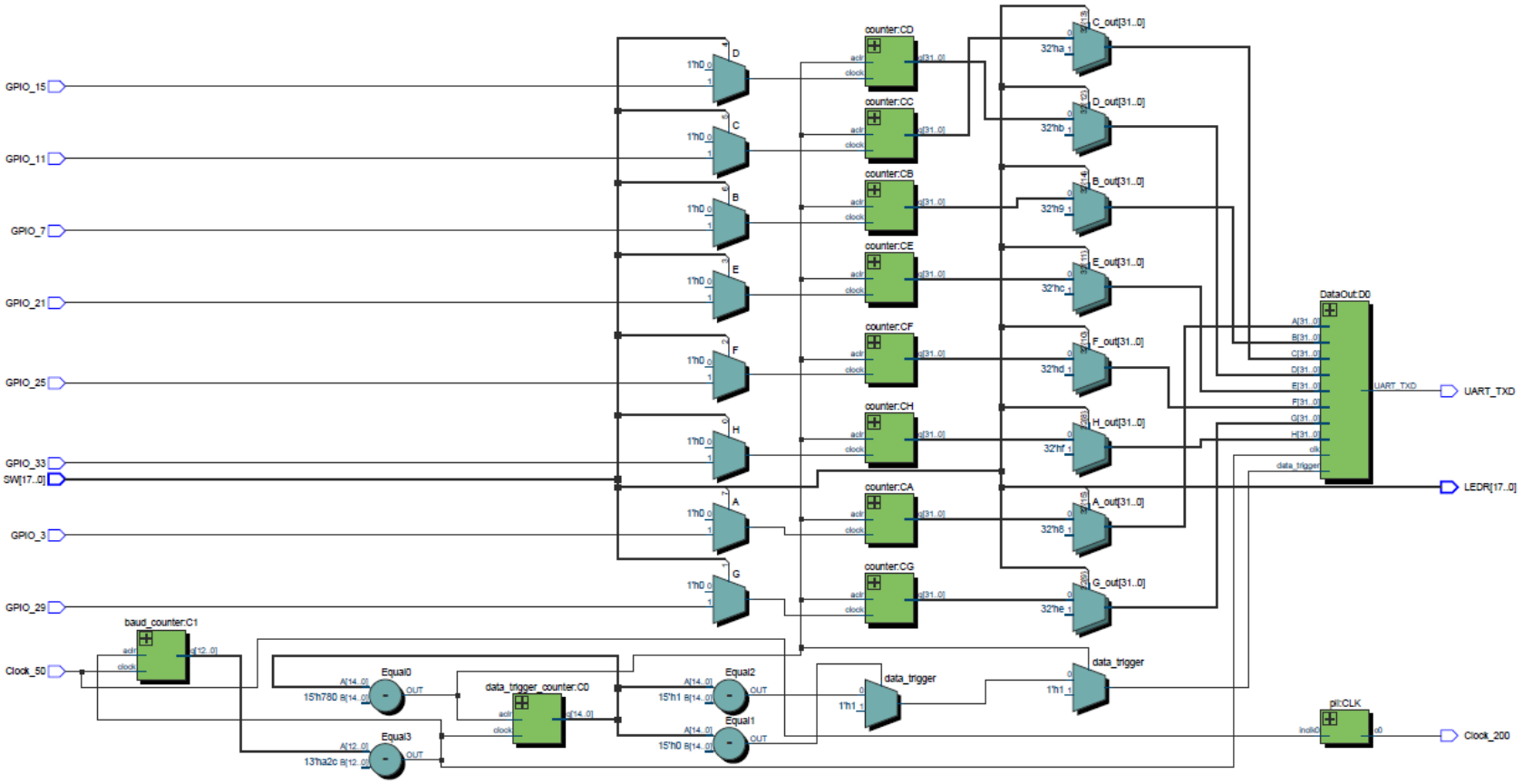
```
      index <= "000110";
      UART_TXD <= Output(4);

ELSIF index = "000110" THEN
      index <= "000111";
      UART_TXD <= Output(5);

ELSIF index = "000111" THEN
      index <= "001000";
      UART_TXD <= Output(6);

ELSIF index = "001000" THEN
      index <= "001001";
      UART_TXD <= '0'; -- the termination bit

ELSIF index = "001001" THEN
      index <= "001010";
      UART_TXD <= '1'; -- the first stop bit

ELSIF index = "001010" THEN
      index <= "001011";
      UART_TXD <= '0'; -- the second start bit

ELSIF index = "001011" THEN
      index <= "001100";
      UART_TXD <= Output(7);

ELSIF index = "001100" THEN
      index <= "001101";
      UART_TXD <= Output(8);

ELSIF index = "001101" THEN
      index <= "001110";
      UART_TXD <= Output(9);

ELSIF index = "001110" THEN
      index <= "001111";
      UART_TXD <= Output(10);

ELSIF index = "001111" THEN
      index <= "010000";
      UART_TXD <= Output(11);

ELSIF index = "010000" THEN
      index <= "010001";
      UART_TXD <= Output(12);

ELSIF index = "010001" THEN
      index <= "010010";
      UART_TXD <= Output(13);

ELSIF index = "010010" THEN
      index <= "010011";
      UART_TXD <= '0'; -- the termination bit

ELSIF index = "010011" THEN
      index <= "010100";
      UART_TXD <= '1'; -- the second stop bit

ELSIF index = "010100" THEN
      index <= "010101";
```

```
      UART_TXD <= '0'; -- the third start bit

ELSIF index = "010101" THEN
      index <= "010110";
      UART_TXD <= Output(14);

ELSIF index = "010110" THEN
      index <= "010111";
      UART_TXD <= Output(15);

ELSIF index = "010111" THEN
      index <= "011000";
      UART_TXD <= Output(16);

ELSIF index = "011000" THEN
      index <= "011001";
      UART_TXD <= Output(17);

ELSIF index = "011001" THEN
      index <= "011010";
      UART_TXD <= Output(18);

ELSIF index = "011010" THEN
      index <= "011011";
      UART_TXD <= Output(19);

ELSIF index = "011011" THEN
      index <= "011100";
      UART_TXD <= Output(20);

ELSIF index = "011100" THEN
      index <= "011101";
      UART_TXD <= '0'; -- the termination bit

ELSIF index = "011101" THEN
      index <= "011110";
      UART_TXD <= '1'; -- the third stop bit

ELSIF index = "011110" THEN
      index <= "011111";
      UART_TXD <= '0'; -- the fourth start bit

ELSIF index = "011111" THEN
      index <= "100000";
      UART_TXD <= Output(21);

ELSIF index = "100000" THEN
      index <= "100001";
      UART_TXD <= Output(22);

ELSIF index = "100001" THEN
      index <= "100010";
      UART_TXD <= Output(23);

ELSIF index = "100010" THEN
      index <= "100011";
      UART_TXD <= Output(24);

ELSIF index = "100011" THEN
      index <= "100100";
      UART_TXD <= Output(25);
```

```
                ELSIF index = "100100" THEN
                      index <= "100101";
                      UART_TXD <= Output(26);

                ELSIF index = "100101" THEN
                      index <= "100110";
                      UART_TXD <= Output(27);

                ELSIF index = "100110" THEN
                      index <= "100111";
                      UART_TXD <= '0'; -- the termination bit


                ELSIF index = "100111" THEN
                      index <= "101000";
                      UART_TXD <= '1'; -- the fourth stop bit

                ELSIF index = "101000" THEN
                      index <= "101001";
                      UART_TXD <= '0'; -- the fifth start bit

                ELSIF index = "101001" THEN
                      index <= "101010";
                      UART_TXD <= Output(28);

                ELSIF index = "101010" THEN
                      index <= "101011";
                      UART_TXD <= Output(29);

                ELSIF index = "101011" THEN
                      index <= "101100";
                      UART_TXD <= Output(30);

                ELSIF index = "101100" THEN
                      index <= "101101";
                      UART_TXD <= Output(31);

                ELSIF index = "101101" THEN
                      index <= "101110";
                      UART_TXD <= '0';

                ELSIF index = "101110" THEN
                      index <= "101111";
                      UART_TXD <= '0';

                ELSIF index = "101111" THEN
                      index <= "110000";
                      UART_TXD <= '0';

                ELSIF index = "110000" THEN
                      index <= "110001";
                      UART_TXD <= '0';


                ELSIF index = "110001" AND data_select = "000" THEN
                      index <= "000000";
                      data_select <= "001"; -- increments data_select to
begin output of B
                      Output <= B;
                      UART_TXD <= '1'; -- the fifth stop bit
```

```
ELSIF index = "110001" AND data_select = "001" THEN
      index <= "000000";
      data_select <= "010"; -- increments data_select to begin output of C
      Output <= C;
      UART_TXD <= '1'; -- the fifth stop bit

ELSIF index = "110001" AND data_select = "010" THEN
      index <= "000000";
      data_select <= "011"; -- increments data_select to begin output of D
      Output <= D;
      UART_TXD <= '1'; -- the fifth stop bit

ELSIF index = "110001" AND data_select = "011" THEN
      index <= "000000";
      data_select <= "100"; -- increments data_select to begin output E
      Output <= E;
      UART_TXD <= '1'; -- the fifth stop bit

ELSIF index = "110001" AND data_select = "100" THEN
      index <= "000000";
      data_select <= "101"; -- increments data_select to begin output F
      Output <= F;
      UART_TXD <= '1'; -- the fifth stop bit

ELSIF index = "110001" AND data_select = "101" THEN
      index <= "000000";
      data_select <= "110"; -- increments data_select to begin output of G
      Output <= G;
      UART_TXD <= '1'; -- the fifth stop bit

ELSIF index = "110001" AND data_select = "110" THEN
      index <= "000000";
      data_select <= "111"; -- increments data_select to begin output of H
      Output <= H;
      UART_TXD <= '1'; -- the fifth stop bit

ELSIF index = "110001" AND data_select = "111" THEN
      index <= "110010";
      UART_TXD <= '1'; -- the fifth stop bit

ELSIF index = "110010" THEN
      index <= "111111";
      UART_TXD <= '0'; -- the start bit of the termination byte

ELSE
      index <= "111111";
      UART_TXD <= '1'; -- sets all subsequent bits to negative voltage
                  END IF;
            END IF;
      END PROCESS;
END Behavior;
```

ANEXO B: Diagrama de Lógica RTL



ANEXO C: Script Matlab

```
%% Lafa Multichannel Counting Unit
% Este código têm a finalidade de ler continuamente os valores de contagem
% de 8 canais conectados ao FPGA. Através do command window, o usuário pode
escolher a porta
%COM utilizada, o tempo de aquisição e a quantidade de loops que serão
realizados.
%
% Plota cada canal em um gráfico individual em uma Única Janela
%%
clc;
clear all;
close all;

%% Parâmetros Configuráveis

PORT = 'COM6';
BAUD = 19200;

CHANGE_FILENAME = 1;

verbose = 0;

file='data_output';

%%
disp('** LaFa Multichannel Counting **')
gate = 0.1; % 100 milissegundos, fixo pelo FPGA

time = input('Entre com o tempo de contagem (por amostra): '); % length of
the measurement in seconds

while time < gate
    disp('Aviso: o valor mínimo deve ser de 0.1 segundo')
    time = input('Entre com a janela de contagem por amostra (seg): '); %
length of the measurement in seconds
end

samples = time/gate;       % number of samples
% comment='test'; % Usa para salvar os dados em .MAT

loops = input('Quantidade de amostras: ');
%%
MM  = 41;           % 41 -1 = 40-bytes
MAX = samples * MM; % 10*41 = 410

data_iter  = 1;
trigg_iter = 0;
set_iter   = 0;
error_data = 0;

data_parser = zeros(1,8);

data = zeros(1, MAX);
outpt_vector = zeros(samples, 8);

% Para nosso setup atual, estamos desconsiderando o canal A pois só
```

```
% lida-mos com 7 tubos independentes

B = nan(1,loops);
C = nan(1,loops);
D = nan(1,loops);
E = nan(1,loops);
F = nan(1,loops);
G = nan(1,loops);
H = nan(1,loops);

%% Obtain date and time and modify the file name
matlab_time = now;
datetime = datestr(matlab_time, 'yyyymmddHHMMSS');

if CHANGE_FILENAME==1
    file = strcat(file,'-',num2str(datetime));
end

txtfile=strcat('data_output/',file,'.txt');
datafile=strcat('data_output/',file,'.mat');

%%
FontSize = 10;
size_screen = get(0,'screensize');

time_start = now; % Hora do início
time_end = time_start + (loops*time)/(60*60*24); % Tempo quando finalmente
é medido;
time_me = time_end - time_start;

figure1 = figure(1);
figure1.Position = [0 0 size_screen(3) size_screen(4)];
figure1.Units = 'pixels';
axes1 = axes('Parent',figure1);

subplot(241);
p1 = plot(B,'-*g','MarkerSize',4,'LineWidth',2);

subplot(242);
p2 = plot(C,'-*b', 'MarkerSize',4,'LineWidth',2);

subplot(243);
p3 = plot(D,'-*r', 'MarkerSize',4,'LineWidth',2);

subplot(244);
p4 = plot(E,'-*y', 'MarkerSize',4,'LineWidth',2);

subplot(245);
p5 = plot(F,'-*k', 'MarkerSize',4,'LineWidth',2);

subplot(246);
p6 = plot(G,'-*m', 'MarkerSize',4,'LineWidth',2);

subplot(247);
p7 = plot(H,'-*c', 'MarkerSize',4,'LineWidth',2);

%% Informações para o usuário
disp(['ID do arquivo: ' file])
%disp(['Samples: ' num2str(samples)])
disp(['Duração da aquisição: ' num2str(loops*time) ' segundos'])
```

```
disp(['Tempo      estimado:      '      char(duration(seconds(loops*time),
'Format','hh:mm:ss'))]);

disp('Não desligue este computador.')
disp('Contagem em progresso...')

%%
if ~isempty(instrfind)
    fclose(instrfindall)
    delete(instrfindall)
end

s = serial(PORT, 'BaudRate', BAUD,'DataBits',8,'Parity','none');
s.BytesAvailableFcnCount = 40;
s.BytesAvailableFcnMode = 'byte';
s.BytesAvailableFcn = @instrcallback;
s.InputBufferSize = 512;
fopen(s);

%%
if verbose
    disp(s)
end
flushinput(s);

%%
tic

for iter_loops = 1:loops
    while(set_iter <= samples)

        rd = fread(s, 1); % N=1 element into a column vector

        if (size(rd,1) == 1)

            data_iter = data_iter +1;

            data( data_iter ) = bin2dec( strcat('0', dec2bin( rd ) ) ); %
cada byte lido, adiciona 1 em data_iter
            %data salva cada byte lido

        end


        % Para a contagem dos canais, nunca vai acontecer de representar um
        % número com 8-bits '1' seguidos. Isso, pois os números são
        % representados no formato {0 7-bits 0 1}.
        % Logo, quando houver '1111 1111', significa que o frame acabou.
        %
        if ( data( data_iter ) == 255 ) % '1111 1111'

            trigg_iter = 0; %Conta os bytes lidos em um frame

            if size(data_set, 2) == MM  % numero de colunas, se for 41

                set_iter = set_iter +1; %contador de frmes em cada loop
                if verbose
                    disp(['#Frame=  ' num2str(set_iter)])  %  Number  of  the
current frame
                end
                %%
```

```
                if (data_set(1) == 255)

                    for channel = 0:7
                        one_chan = data_set(2+5*channel : 5*channel+6);
                        data_parser(channel+1)                                  =
bin2dec(strcat(dec2bin(one_chan(5),7),dec2bin(one_chan(4),7),dec2bin(one_ch
an(3),7),dec2bin(one_chan(2),7),dec2bin(one_chan(1),7)));
                        % dec2bin(num_dec, 7) representação binária com 7
bits

                    end

                    outpt_vector(set_iter,:) = data_parser(:);

                end
            end

            if size(data_set,2) > MM
                error_data = error_data+1 ;
            end

            data_set = zeros(0);
        end

        trigg_iter = trigg_iter + 1;

        data_set(trigg_iter)=data(data_iter);  %data_set  é  um  vetor  que
guarda os bytes lidos em cada frame

        if verbose
            disp(['#Byte= ' num2str(trigg_iter)])
            disp(['#Sample= ' num2str(data_iter)])
        end
    end
   %salva os dados
   %saved_data = outpt_vector(:,2:8); % Salva todos os canais com excessão
do A
    saved_data = outpt_vector(2:end,2:8);
    save(txtfile,'saved_data','-ASCII', '-append');
   %save(datafile,'saved_data','-ASCII','-append');
    disp(iter_loops);
    flushinput(s);

    set_iter  = 0;


    %% Pós processamento
    % 1) Integral dos dados lidos

    %Plota a média dos pontos lidos ao longo de um loop
    %B(iter_loops) = mean(outpt_vector(:,2));

    %Plota a integral dos pontos contados ao longo de um loop

    %% TESTES !!!
    B(iter_loops) = sum(outpt_vector(2:end,2));
    C(iter_loops) = sum(outpt_vector(2:end,3));
    D(iter_loops) = sum(outpt_vector(2:end,4));
    E(iter_loops) = sum(outpt_vector(2:end,5));
    F(iter_loops) = sum(outpt_vector(2:end,6));
    G(iter_loops) = sum(outpt_vector(2:end,7));
```

```
    H(iter_loops) = sum(outpt_vector(2:end,8));
    %disp(['B = ' num2str(B(iter_loops))]);

    %% 2) Plote

    subplot(241); delete(p1); %delete(t1);
    p1 = plot(B,'-*g', 'MarkerSize',4,'LineWidth',2);
    subplot(242); delete(p2);
    p2 = plot(C,'-*b', 'MarkerSize',4,'LineWidth',2);
    subplot(243); delete(p3);
    p3 = plot(D,'-*r', 'MarkerSize',4,'LineWidth',2);
    subplot(244); delete(p4);
    p4 = plot(E,'-*y', 'MarkerSize',4,'LineWidth',2);
    subplot(245); delete(p5);
    p5 = plot(F,'-*k', 'MarkerSize',4,'LineWidth',2);
    subplot(246);delete(p6);
    p6 = plot(G,'-*m', 'MarkerSize',4,'LineWidth',2);
    subplot(247); delete(p7);
    p7 = plot(H,'-*c', 'MarkerSize',4,'LineWidth',2);

    drawnow; % Atualizar o gráfico exibido na tela


end
toc
disp('Aquisição Finalizada. Dados salvos com sucesso.')


%%
fclose(s);
delete(s);
```