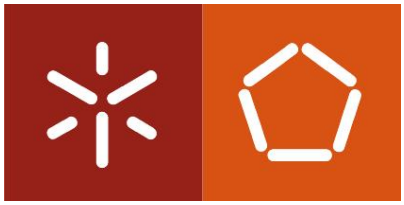
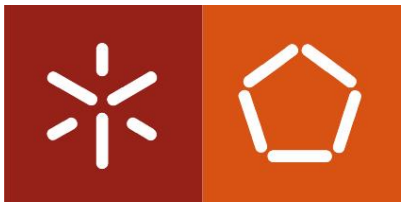


Universidade do Minho
Escola de Engenharia

Davide Filipe Ferreira Guimarães

Plataforma de testes programável baseada em FPGA para microsensores capacitivos

Novembro de 2014



Universidade do Minho
Escola de Engenharia

Davide Filipe Ferreira Guimarães

Plataforma de testes programável baseada em FPGA para microsensores capacitivos

Dissertação de Mestrado
Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Luís Rocha
E Coorientação do
Professor Doutor Jorge Cabral

Novembro de 2014

Universidade do Minho, ____/____/____

_____

Agradecimentos

As minhas primeiras palavras de agradecimento vão sem dúvida para os meus pais Adélio Guimarães e Rita Silva pelo apoio e suporte que sempre me deram tanto no meu percurso académico como em todas as etapas da minha vida, e também para o meu irmão Manuel.

Ao meu orientador, professor doutor Luís Alexandre Rocha, e ao meu co-orientador, professor doutor Jorge Cabral pela confiança e pelo apoio em mim depositado durante o desenvolvimento deste trabalho.

Ao ESRG pertencente ao Departamento de Eletrónica da Universidade do Minho pelo acolhimento e suporte continuo demonstrado tornando-se essencial para a elaboração deste trabalho, assim como a todos os meus colegas de laboratório que providenciaram excelentes momentos de boa disposição. Um obrigado especial ao Vasco Lima e ao Filipe Serra Alves pela disponibilidade e apoio incansável, estando sempre prontos a ajudar e a partilhar conhecimentos.

À minha namorada Sónia Rego, que sempre me apoiou quer nos bons quer nos maus momentos e que me proporciona sempre bem estar e felicidade.

Queria também agradecer a Eurico Moreira pelo companheirismo e amizade demonstrados neste percurso académico.

Por fim ao Pedro Ribeiro e ao Pedro Rocha, pessoas com as quais partilhei casa durante este percurso académico, e com os quais guardarei na memória momentos de boa disposição.

Resumo

Um dos ramos tecnológicos que mais evoluiu na última década está relacionado com o mundo sensorial, mais concretamente os microsensores. O grande leque de aplicações que lhes estão associados leva à alteração sistemática da implementação dos sistemas que os integram, o que se transforma num problema quando estes sistemas são implementados puramente em *hardware*.

O objetivo principal desta dissertação prende-se com o desenvolvimento de uma plataforma de testes, baseada em FPGA, para microsensores capacitivos. Esta plataforma incorpora um sistema de aquisição e um sistema de atuação, permitindo ainda uma configuração facilitada da topologia implementada com recurso à plataforma de desenvolvimento ZedBoard, um sistema programável de elevado desempenho. Em FPGA são implementados os módulos de interface com o *hardware* e no microprocessador, em ambiente Linux, são efetuadas as configurações necessárias bem como o processamento de dados.

A definição do mecanismo de aquisição, quer por um conversor analógico para digital quer por comparadores analógicos, a definição dos número de atuadores electroestáticos que possam estar presentes no microsensor, bem como a inclusão de um filtro digital ou de vários canais de comunicação série são alguns dos aspetos associados à plataforma.

De forma a validar e a testar as funcionalidades incorporadas na plataforma, foram implementadas algumas leis de controlo, elevando o patamar de versatilidade relativamente às plataformas existentes.

Palavras-chave: *Microsensores capacitivos, Plataforma de testes, FPGA, Linux*

Abstract

One of the branches of technology that most evolved in the last decade is related to the sensory world, specifically microsensors. The wide range of applications associated with them leads to a systematic change of the implementation of the systems that integrate them, which becomes a problem when these systems are implemented purely in hardware.

The main objective of this thesis concerns the development of a test platform based on FPGA for capacitive microsensors. This platform incorporates an acquisition system and actuation system, allowing an easy configuration of the implemented topology. Using the development platform ZedBoard, a programmable high performance system was developed, where key interface modules with the hardware are implemented on a FPGA and in a microprocessor in Linux environment the necessary configurations and data processing are performed.

The definition of the acquisition mechanism, through an analog to digital converter or analog comparators, the definition of the number of electrostatic actuators that is associated to the microsensor, as well as inclusion of a digital filter and serial comunication channels are some of the features enabled by the platform.

In order to certify and validate the platform's functionalities some control laws were implemented and tested, raising the level of versatility that is reduced in existing platforms.

Keywords: *Capacitive microsensors, testing platform, FPGA, Linux*

Conteúdo

Abreviaturas e siglas

ADC	- Analog-to-Digital Converter
ASIC	- Application Specific Integrated Circuits
AXI	- Advanced eXtensible Interface
BRAM	- Block Random Access Memory
CLK	- Clock
CMOS	- Complementary Metal-Oxide-Semiconductor
CPU	- Central Processing Unit
DAC	- Digital-to-Analog Converter
dB	- Decibel
DRG	- Disc Resonance Gyroscope
DSP	- Digital Signal Processor
FIR	- Finite Impulse Response
FPGA	- Field-Programmable Gate Array
FSBL	- First Stage Boot Loader
GND	- Ground
GPIO	- General-purpose input/output
HDL	- Hardware Description Language
HDMI	- High-Definition Multimedia Interface
Hz	- Hertz
IC	- Integrated Circuit
IP	- Intellectual Property
IRU	- Inertial Reference Unit
LED	- Light Emitting Diode
LPF	- Low-pass Filter
LSB	- Least Significant Bit
MCU	- Multipoint Control Unit
MEMS	- Microelectromechanical Systems
MOSI	- Master Output Slave Input

MSB	- Most significant bit
MSPS	- Mega Samples Per Second
NC	- Not Connected
PC	- Personal Computer
PI Controller	- Proportional-Integral Controller
PL	- Programmable Logic
PLL	- Phase-locked loop
PS	- Processing System
PWM	- Pulse-width modulation
RAM	- Random Access Memory
SCLK	- Serial Clock
SCU	- Snoop Control Unit
SDIN	- Serial Data In
SoC	- System-on-a-Chip
SPI	- Serial Peripheral Interface
SS	- Slave Select
SYNC	- Digital Interface Synchronization Input Pin
UART	- Universal Asynchronous Receiver/Transmitter
USB	- Universal Serial Bus
V	- Volt (Tensão)

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1. *Enquadramento*

Hoje em dia os equipamentos tecnológicos e o seu desenvolvimento influenciam diretamente a vida quotidiana das pessoas, uma dependência que tende a aumentar com o evoluir do tempo. Dentro de um vasto leque desses equipamentos surgem os microsensores capacitivos com atuação electroestática.

Esses microsensores têm sempre associado um circuito de aquisição que permite a obtenção dos valores correspondentes ao deslocamento capacitivo das estruturas. Assim, é realizada a leitura da capacidade para posteriormente ser usado. Este valor é usualmente convertido para um valor digital recorrendo a um conversor analógico digital. Outro sistema que poderá estar associado é o sistema de atuação que possibilita a atuação nas microestruturas de modo a que se ative um mecanismo pré-concebido. Em alguns casos e tendo como objetivo a coordenação dos sistemas de atuação e leitura surge o sistema de controlo para assegurar o correto funcionamento do sistema.

Na quase totalidade dos projetos desenvolvidos, em que o seu propósito seja a leitura do sinal fornecido por sensores capacitivos [1] ou mesmo piezorestivos [2], tanto o sistema de aquisição como o sistema de atuação, inerentes ao sensor, são implementados em *hardware*. Apesar de não haver problemas relacionados com a velocidade de aquisição, qualquer tipo de alteração nos circuitos é de difícil implementação, sendo que é também necessário um elevado conhecimento nos tipos de metodologias adotadas para as poder alterar com sucesso.

Um exemplo dessa situação seria uma implementação em *hardware* baseado em FPGA. A alteração de parâmetros ou topologias do sistema implicaria a alteração de blocos de código verilog HDL[1]. Para se fazer essa alteração é necessário perceber a função dos módulos existentes caso contrário não se teria a noção do que alterar. Para além desse conhecimento,

[1] Linguagem de descrição de *hardware* usada para modelar sistemas eletrónicos.

haveria desperdício de tempo a pensar na implementação e em que módulo se teriam de alterar e corria-se o risco de corromper todo o sistema sem saber a razão de tal.

Essa necessidade de mudar o tipo de atuação ou leitura pode surgir, por exemplo, nos processos de caracterização ou implementação de sistemas de controlo nos quais é necessário mudar o modo de atuação. Dependendo do tipo e atuação a que as estruturas estão sujeitas, estas reagem de forma diferente e com isso obtém-se resultados diversificados, daí a necessidade de regularmente alterar as topologias implementadas.

1.2. *Objetivos*

Nesta dissertação pretende-se desenvolver uma plataforma de testes digital direcionada para microsensores capacitivos. O objetivo é retirar complexidade ao sistema facilitando deste modo o trabalho de potenciais desenvolvedores ou utilizadores ativos de microsensores. A plataforma de desenvolvimento terá que ser facilmente programável, recorrendo com isso a uma linguagem de programação comum, e terá que conceder o total acesso ao sistema.

1.3. *Estrutura da Dissertação*

No primeiro capítulo é introduzido o tema desta dissertação fazendo-se um breve enquadramento em jeito de motivação seguido da definição dos objetivo e da descrição da estrutura deste documento.

No segundo capítulo é apresentado o estado de arte, sendo referidas diferentes plataformas existentes bem como as suas caracterizas de modo a serem tiradas ilações importantes para o desenvolvimento deste trabalho.

No terceiro capítulo é feita uma análise ao sistema a implementar, identificando os requisitos e as restrições para posteriormente ser definida a abordagem a seguir.

O quarto capítulo diz respeito ao *design* do sistema. É definido e descrito todo o *hardware* e *software* bem como as ferramentas de desenvolvimento usadas. São também expostos os conceitos teóricos relacionado com os casos de teste a implementar.

O quinto capítulo diz respeito à implementação do sistema. São descritos todos os processos de desenvolvimento, desde a conceção dos circuitos de *hardware* até ao desenvolvimento das aplicações de *software* desenvolvidas.

No sexto capítulo apresentam-se os passos de execução bem como os resultados relativos à aplicação dos casos de testes.

Por último, no sétimo capítulo são tiradas as respetivas conclusões e limitações do trabalho desenvolvido bem como ilações para trabalho futuro.

2. Estado de Arte

Neste estado de arte serão abordados diversos trabalhos em que foram desenvolvidos, em FPGA ou em microprocessadores, sistemas de controlo ou de monotorização, onde é necessária a aquisição e atuação de sinais relacionados com microsensores capacitivos. Pretende-se com isso demonstrar que este trabalho pode ser usado com os microsensores que se desenvolvem no mercado e que a plataforma desenvolvida tem funcionalidades extremamente uteis e das quais se podem retirar diversas vantagens.

2.1. *Projetos Desenvolvidos*

Segue-se de seguida alguns projetos desenvolvidos na atualidade e um breve enquadramento com o trabalho a desenvolver.

1. Em 2012, Dunzhu Xia, Cheng Yu e Yuliang Wang, publicaram um artigo onde é implementado um método de miniaturização digital para um micro-giroscópio em silício (SMG) [3]. O diagrama de blocos do sistema consiste em conversores analógico para digital e digital para analógico, num dispositivo baseado em FPGA e outros periféricos para transmissão de dados como porta-série. Pretendeu-se assim desenvolver um sistema hibrido, onde nos componentes de *hardware* são tratados os sinais quer de aquisição quer de atuação e na FPGA são tratados os sinais, quer a nível de filtragem e posterior implementação de um sistema de controlo quer para monitorização quer para as comunicações pela porta série.

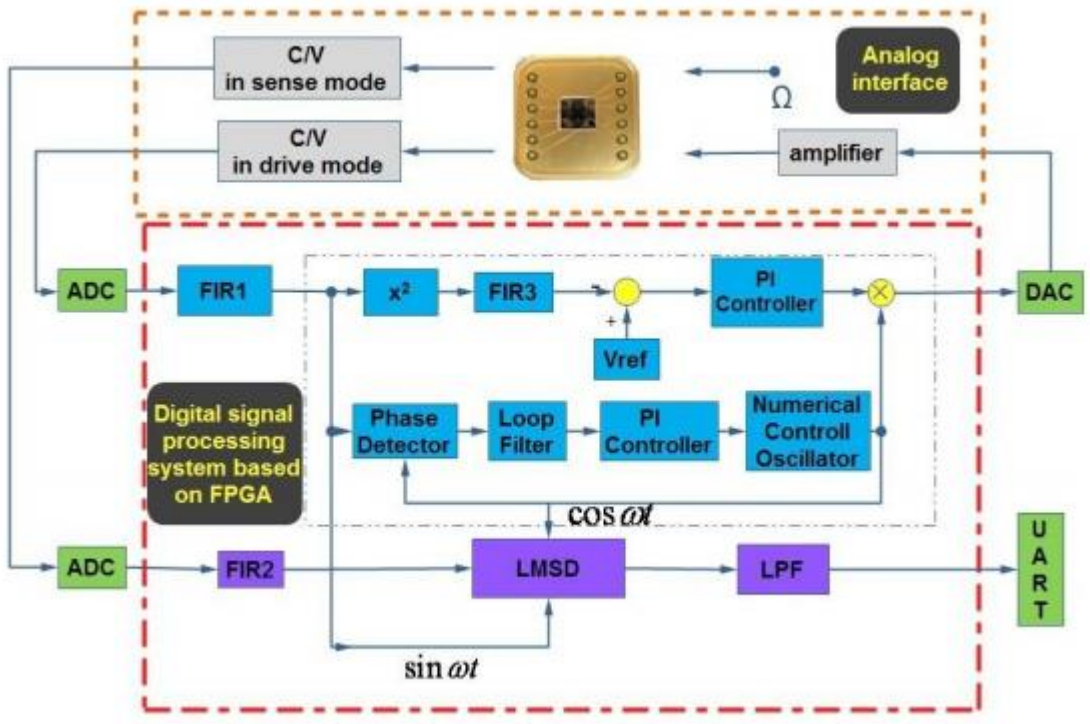


Figura 2.1 - Diagrama de blocos do sistema [3]

2. Em 2005, na 13ª conferência internacional Transducers, Henrik Rödjegård apresentou um artigo que descreve a implementação de um controlo digital para giroscópios MEMS [4]. O controlo é implementado em FPGA que está conectado ao giroscópio através de um circuito analógico de interface.

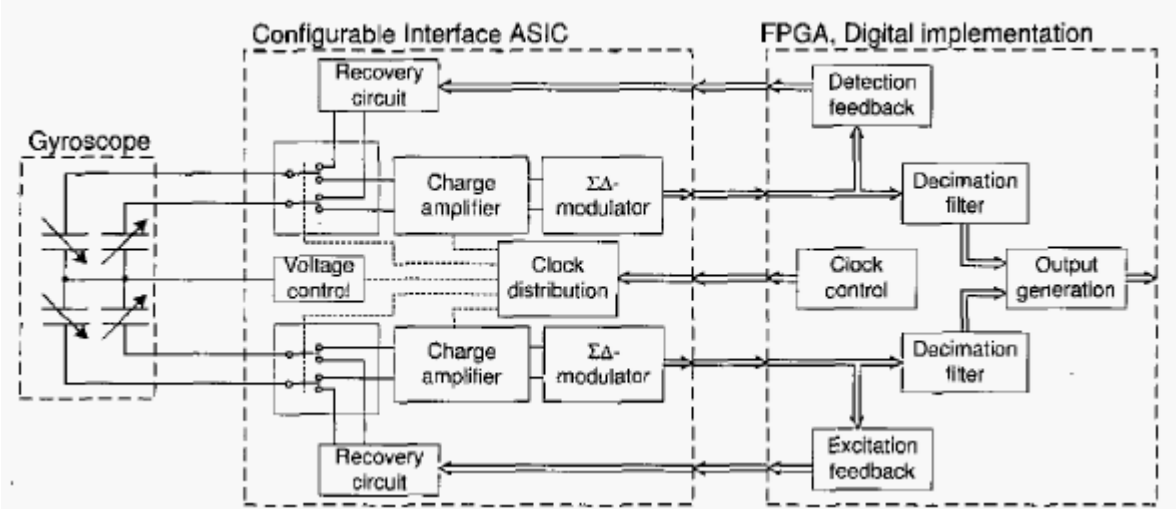


Figura 2.2 - Diagrama de blocos do sistema [4]

Nesta implementação a conversão dos sinais a saída dos *charge amplifiers* é levado a cabo por moduladores sigma-delta. Posteriormente em FPGA o sinal é decimado através de

um filtro e é gerado um *bitstream* de saída. Também na FPGA é implementado um algoritmo de controlo responsável por enviar um sinal de *feedback* para o ASIC.

3. Em 2008, Didier Keymeulen e um grupo de investigadores, fizeram uma publicação onde discutem a implementação de um controlo digital para um MEMS *Disc Resonance Gyroscope* (DRG) [5], usualmente integrados em unidades de referência inercial (IRU), recorrendo a uma plataforma FPGA.

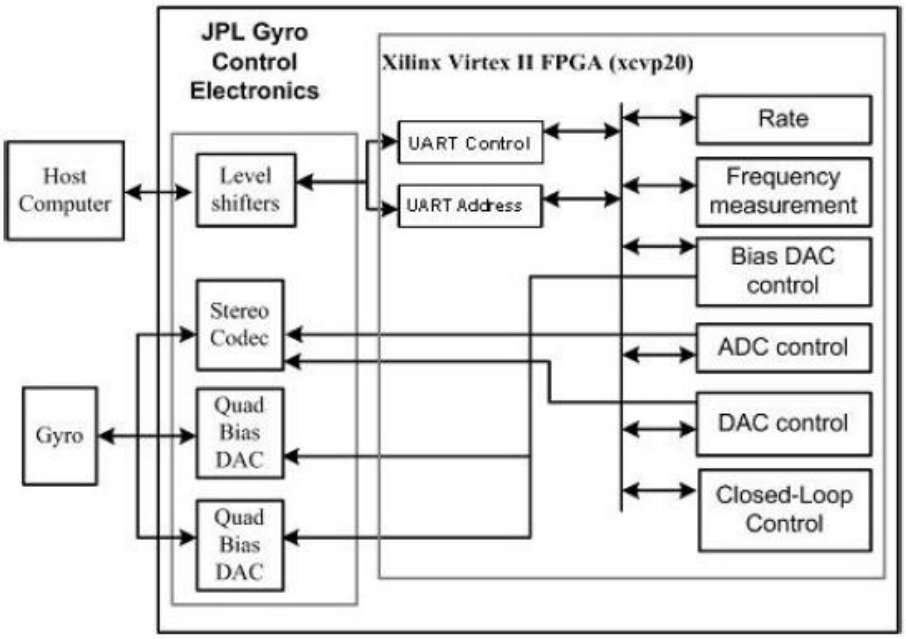


Figura 2.3 - Diagrama de blocos do sistema [5]

A Figura 2.3 ilustra a implementação dos sistemas analógicos e digitais usados no controlo do micro-giroscopio. Inclui diversos conversores analógico para digital e digital para analógico, comunicação série *Universal Asynchronous Receiver/Transmitter* e outras funcionalidades incorporadas na FPGA da Xilinx. Com recurso à UART, a FPGA encontra-se conectada a um PC local que através de sinais previamente estabelecidos configura os modos existentes na mesma. Neste caso existem duas linhas de dados UART, uma para estabelecer o controlo da plataforma e a outra para adquirir dados existentes na mesma.

4. Em 2007, numa conferência internacional onde se debateu sobre *"Information Acquisition"*, Guo Qinglei, Liang Huawei, Miao Shifu e Huang Jian apresentaram uma proposta para um inclinómetro baseado em acelerómetros MEMS [6], em que recorria a um microprocessador da família 8051 para a aquisição do sinal.

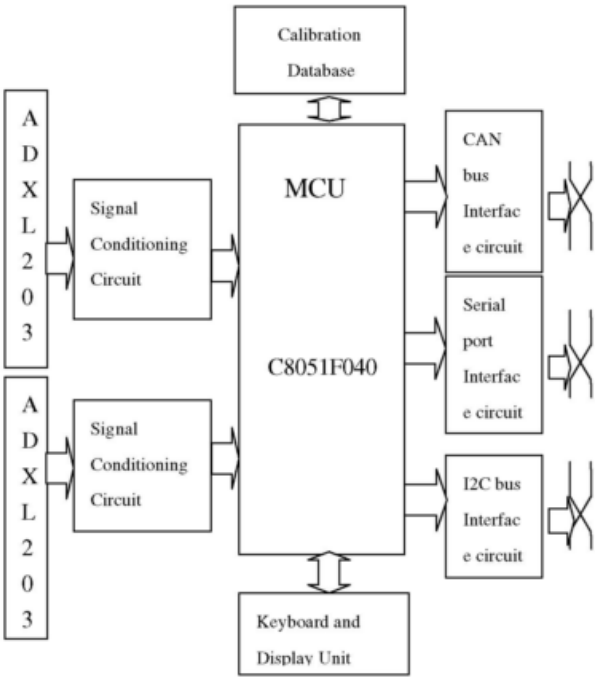


Figura 2.4 - Diagrama de blocos do sistema [6]

Neste sistema o microcontrolador é o kernel do sistema, e tem a função de processar os dados adquiridos e gerir todas as partes do sistema.

5. O projeto que se segue foi desenvolvido pela empresa ITMEMS. É uma plataforma de instrumentação que permite adquirir informação sobre sensores inerciais MEMS tal como, frequência de ressonância, fator de qualidade, tensões de pull-in, entre outros [7].

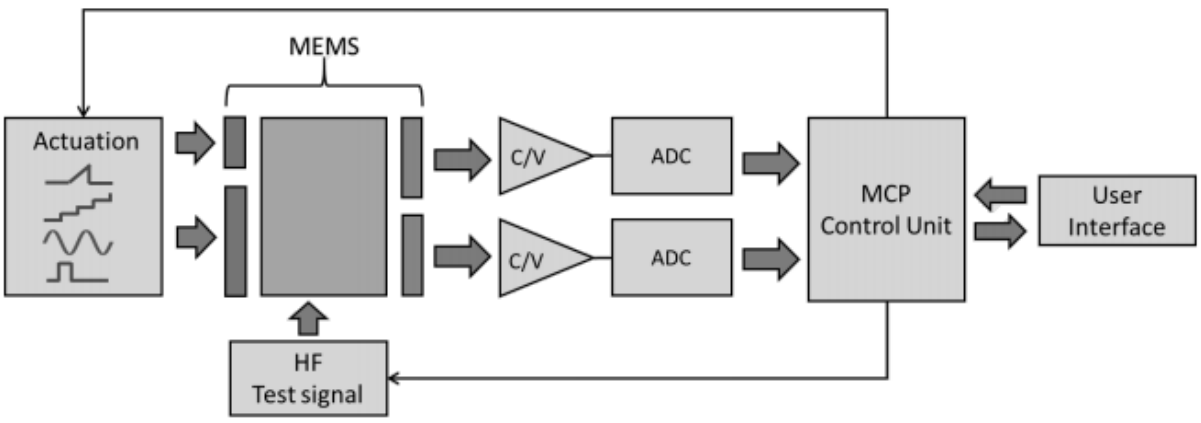


Figura 2.5 - Diagrama de blocos do sistema [7]

A plataforma contém sistemas de atuação, de aquisição e um sistema de controlo. A unidade de controlo implementada é responsável pela configuração do sistema que através das configurações do utilizador implementa o segmento respetivo.

6. Em 2013, no "*IEEE International Symposium on Industrial Electronics"* (ISIE) F. S. Alves, R. A. Dias, J. Cabral e L. A. Rocha, apresentaram uma proposta de um inclinómetro baseado em MEMS [8] e que recorreria a uma placa de desenvolvimento com base em FPGA para gerir todo o sistema.

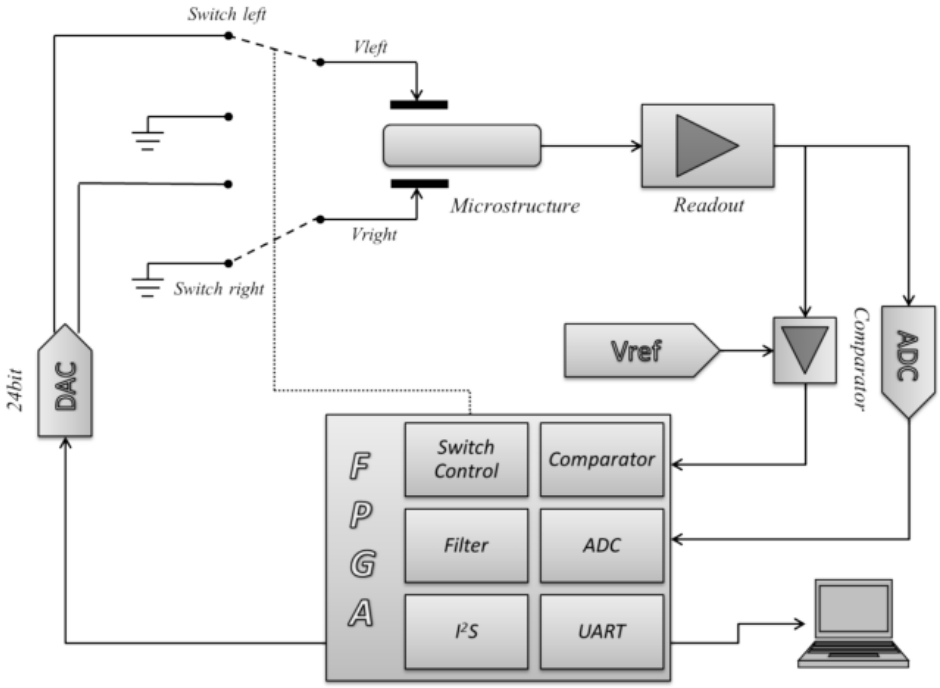


Ilustração 1 - Diagrama de blocos do sistema [8]

Neste artigo é apresentado um inclinómetro baseado em MEMS com atuação electroestática. Os sistemas de aquisição, de atuação e de controlo estão implementados em FPGA. Os dados adquiridos são enviados em tempo real para um terminal para posterior analise.

2.2. *Resumo do Estado de Arte*

Foram anteriormente apresentadas diferentes plataformas relacionadas com a aquisição, controlo e atuação em microsensores capacitivos. A maioria destas implementações são parcialmente incorporadas em FPGA [3]–[5], onde regra geral é feito o tratamento de

informação provenientes de ADCs ou mesmo de moduladores sigma delta. Nestes casos é garantida a melhor performance, fruto das vantagens que advêm de uma plataforma de *hardware* reconfigurável. Porém a versatilidade do sistema é deveras limitada no que diz respeito ao tempo de reformulação do código HDL e do tempo de geração do *bitstream*. Outra implementação possível e descrita anteriormente é o uso de um microprocessador [6], onde a versatilidade é mais alargada, todavia a performance é muito limitada. Quando se pretende executar algo mais que uma simples aquisição do sinal, os ciclos de relógio necessários para a execução da tarefa aumenta, não se obtendo o que previamente se esperava. Uma outra solução apresentada foi uma plataforma de testes com diferentes topologias implementadas e prontas a serem usadas [7]. Porém e apesar de oferecer vários mecanismos para realizar as mais diversas operações acaba por ser limitador ao nível de controlo e de atuação, bem como para teste de novas funcionalidades para além das disponibilizadas.

3. Análise do Sistema

Neste capítulo faz-se uma análise geral do sistema, de modo a definir a melhor metodologia que deve ser empregue neste projeto. Desde modo, nesta fase são definidas quais as restrições e os requisitos que devem ter-se em conta, assim como especificar a abordagem a implementar no desenvolvimento do trabalho apresentado nesta dissertação.

3.1. *Visão Geral*

Na seguinte figura apresenta-se de uma forma genérica uma visão geral do sistema, onde estará inserida a plataforma.

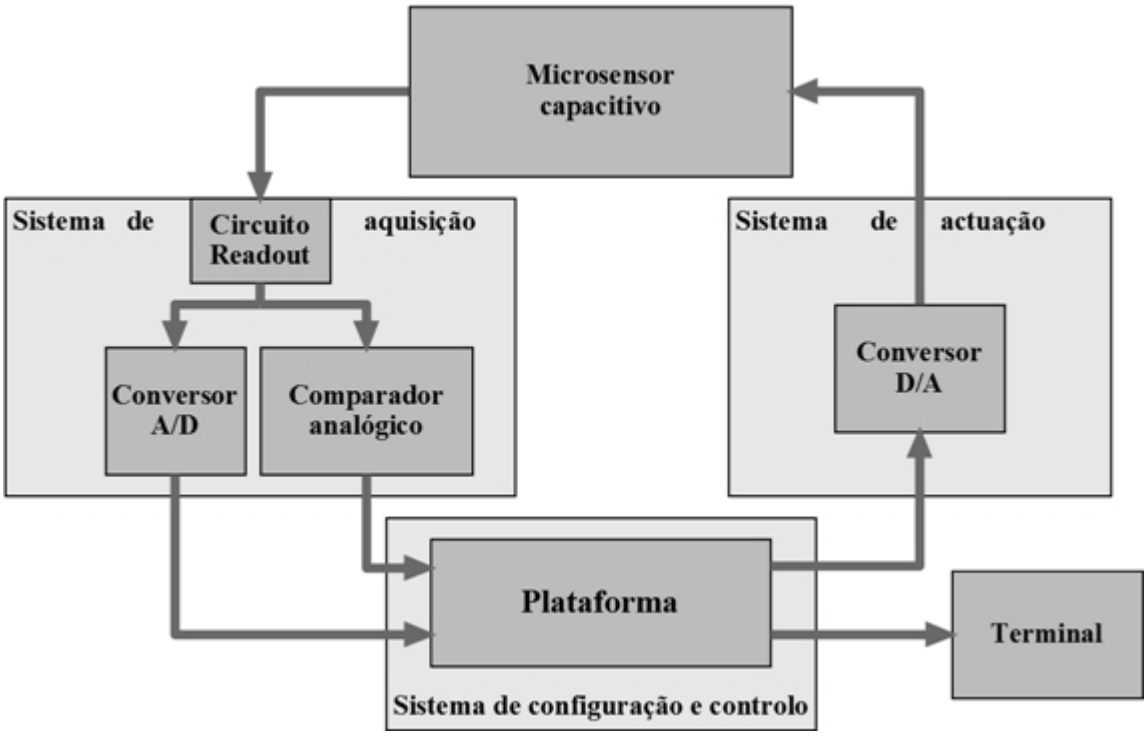


Figura 3.1 - Visão geral do funcionamento do sistema

Como se pode verificar na Figura 3.1 o sistema possibilitará dois modos de aquisição, através de um conversor A/D ou através de um comparador analógico. Os dados são recebidos pela plataforma que procede ao tratamento dos dados segundo um algoritmo implementado pelo utilizador. O sistema de atuação consistirá num conversor D/A e fará a interação com o microsensor. Integrado também com o sistema, há o módulo de comunicação UART para estabelecer uma comunicação externa.

3.2. *Requisitos do Sistema*

Neste subcapítulo apresentar-se-á os requisitos que estão inerentes ao sistema. Para melhor exposição destes requisitos dividir-se-ão em duas categorias independentes sendo estes os requisitos funcionais e os requisitos não funcionais. Nos requisitos funcionais serão abordadas as funcionalidades que o sistema terá que possuir e nos requisitos não funcionais abordam-se os aspetos, que apesar de não acrescentar nenhuma função adicional, melhoram a interação do sistema com o meio.

3.2.1. Requisitos Funcionais

No que diz respeito aos requisitos funcionais, o sistema terá que ser capaz de:

- Adquirir dados relativos aos sensores capacitivos, provenientes, por exemplo, de um ADC ou de um comparador analógico.
- Caso os dados sejam provenientes de um ADC, é necessário filtra-los de modo a eliminar o ruido e frequências indesejadas.
- Permitir ao utilizador definir a frequência de aquisição, dentro de uma gama de valores pré-definidos.
- Possibilitar a implementação de teorias de controlo e que se adequem ao sistema proposto, de modo a controlar a estrutura com o deslocamento desejado.
- Permitir selecionar o DAC em que é pretendido definir o seu valor de atuação.
- Definir o valor de atuação pretendido para um determinado DAC.
- Definir o nível lógico de atuação de um determinado DAC, isto é, definir o estado de atuação (ativo ou não ativo).

- Possibilitar comunicações com terminais externos, como por exemplo porta série, possibilitando assim uma interação em tempo real com um *software* externo.
- Possuir a opção de *reset* de modo a reiniciar todo o sistema.

3.2.2. Requisitos Não-Funcionais

Quanto aos requisitos não funcionais, a plataforma terá que obedecer a alguns critérios no que diz respeito à interação com o utilizador. Esta terá que ser intuitiva, de fácil programação e que a sua gestão não necessite de elevado conhecimento técnico.

3.3. *Abordagem*

Analisando o sistema no seu todo podem-se identificar três partes que compõem a plataforma de testes das quais fazem parte o *hardware* externo, módulos implementados em FPGA e o sistema operativo.

No *hardware* externo poderá estar incluído:

- ADC, responsável por converter o sinal analógico do circuito de leitura num sinal digital.
- Comparadores analógicos, cuja função é comparar a posição do sensor com uma referência pré definida e retribuir um valor lógico.
- DAC, que converte os sinais provenientes da plataforma em sinais analógicos que irão servir como atuação no micronsensor.

O objetivo deste projeto é criar uma plataforma que teste diversas vertentes do sensor e portanto a parte que compõe o *hardware* externo terá que se encontrar totalmente desenvolvida de modo a ser corretamente testada, não sendo o seu desenvolvimento um dos objetivos desta dissertação (será usado hardware desenvolvido previamente [9]).

Relativamente aos módulos que serão implementados em FPGA servirão como interface entre a zona de operação a que o utilizador tem acesso, ou seja o sistema operativo e o microsensor. Estes terão múltiplas funções que serão idealizadas de acordo com as restrições abordadas no subcapítulo anterior podendo posteriormente ser alteradas caso haja limitações ligadas à própria plataforma.

Por fim tem-se o sistema operativo onde serão implementadas todas as ações requeridas e onde o utilizador poderá por em práticas todos os métodos de teste que pretender, dentro das possibilidades que a plataforma dispuser. Assim o sistema operativo e módulos implementados em FPGA estabelecem a comunicação com os módulos de modo a trocarem informação.

3.4. *Diagrama de Casos de Uso*

Na figura seguinte está ilustrado o diagrama de caso de uso descrevendo as funcionalidades do sistema criado no que toca à interação com o utilizador.

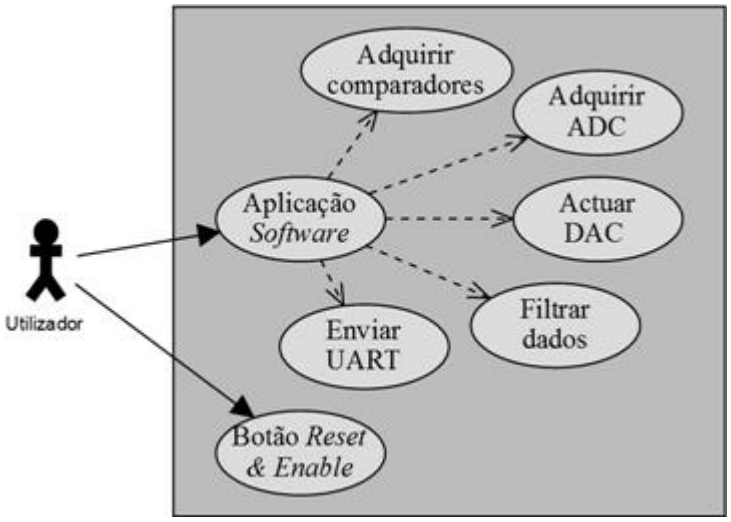


Figura 3.2 - Diagrama de casos de uso do sistema

4. Design do Sistema

Neste capítulo é analisado pormenorizadamente a melhor forma de responder aos requisitos enumerados no capítulo da análise. Serão apresentados os componentes de *hardware* que irão compor a plataforma assim como as respetivas interações entre os mesmos. Posteriormente serão ilustrados diversos casos de teste que servirão como base de teste do sistema como um todo a fim de testar as suas funcionalidades, servindo também como exemplo de teste para posteriores implementações por parte de um futuro utilizador.

4.1. *Especificação do Hardware*

Neste subcapítulo é analisado o *hardware* que melhor se enquadra para a obtenção dos objectivos a alcançar no desenvolvimento deste trabalho. É pretendido um sistema embebido que adquire valores digitais e que tenha a capacidade de os armazenar na sua totalidade. Também é pretendido que os consiga manipular através de um sistema de controlo e que seja gerado um valor de saída correspondente à atuação electroestática do sensor.

Uma primeira opção seria o recurso a um microcontrolador, porém surgem várias condicionantes com a implementação do mesmo. Uma das questões que se prende com a utilização dos microcontroladores é a sobrecarga do sistema, que normalmente conduz a um aumento de latência na transmissão de dados e também a um atraso na geração de interrupções, entre outros. Consequentemente haveria pouca velocidade de transmissão e pouca quantidade de dados transmitidos o que limitaria em muito as potencialidades da plataforma.

Assim é crucial a implementação de processamento paralelo para o desempenho crítico do sistema, sendo simultaneamente necessário um controlador central e um gestor de memória. Uma solução possível seria a construção de um sistema hibrido, um microcontrolador juntamente com uma FPGA, o que seria a junção do melhor de dois mundos, isto é, o processamento paralelo fornecido pela FPGA com a versatilidade de um

microcontrolador. Contudo a implementação separada destes mundos traz diversas limitações, tais como na limitação de conexões entre o microprocessador e a FPGA, no aumento tanto da potência consumida, como no aumento do tamanho e complexidade da PCB resultante, entre outros.

Neste segmente surge a fusão de processamento de sistema com lógica programável num só dispositivo. Um dos dispositivos que permite essa funcionalidade é a Zynq-7000 *All Programmable System on Chip*. Trata-se de um chip de 28nm *programmable logic* pertencente à família Serie 7 da Xilinx acoplado com um core processador dual ARM Cortex-A9 MP (667 MHz) com uma variedade de funções de interface embebidas como I/Os de elevado desempenho, elevado *throughput* AXI[2] com milhares de conexões entre PS (*Processing System*) e PL (*Programmable Logic*).

4.1.1. Avnet ZedBoard

Dentro das diversas placas de desenvolvimento que se baseiam na Xilinx Zynq-7000 *All Programmable System on Chip* selecionou-se a ZedBoard por uma questão meramente monetária, tendo em consideração que as capacidades da placa preenchem os parâmetros requeridos. Combinando um dual Cortex A9 Processing System com 85000 células de lógica programável 7 Series, esta placa contém interface e funções que permitem a implementação de uma variedade de funções. Entre outros a ZedBoard possui como características:

- Xilinx XC7Z020-1CGL484CES Zynq-7000 AP SoC
  - Operações até 667MHz
  - NEON Processing/FPU Engines
- Memória 512 MB DDR3
  - 256Mb Quad SPI Flash
  - 512 MB DDR3 (128 x 32)
  - 4GB SD Card
- Conexões:
  - 10/100/1000 Ethernet
  - USB-JTAG Programming

[2] Advance eXtensively Interface

    - SD Card
    - USB 2.0 FS USB-UART Brigde
    - 5 Headers Digilent Pmod
    - Conector FMC (Low Pin Count)
    - USB OTG 2.0 (Device/Host/OTG)
    - 2 Reset Button (1 PS, 1 PL)
    - 7 Push Button (2 PS, 5 PL)
    - 8 Switches (PL)
    - 9 User LEDs (1 PS, 8 PL)
    - ARM Debug Access Port (DAP)
    - Xilinx XADC Header
- Osciladores On-board
    - 33.333MHz (PS)
    - 100 MHz (PL)
- Display/Audio
    - Saída HDMI
    - VGA (12-bit Colour)
    - 128x32 OLED Display
    - Audio Line-in, Line-up, headphone, microphone
- Alimentação
    - 12V DC @ 3.0A (Max)
- Dimensões
    - Comprimento 16cm, largura 13.5cm

A Figura 4.1 ilustra o diagrama de blocos dos diferentes periféricos presentes no ZedBoard, bem como as respetivas conexões à Zynq-7000 SoC.

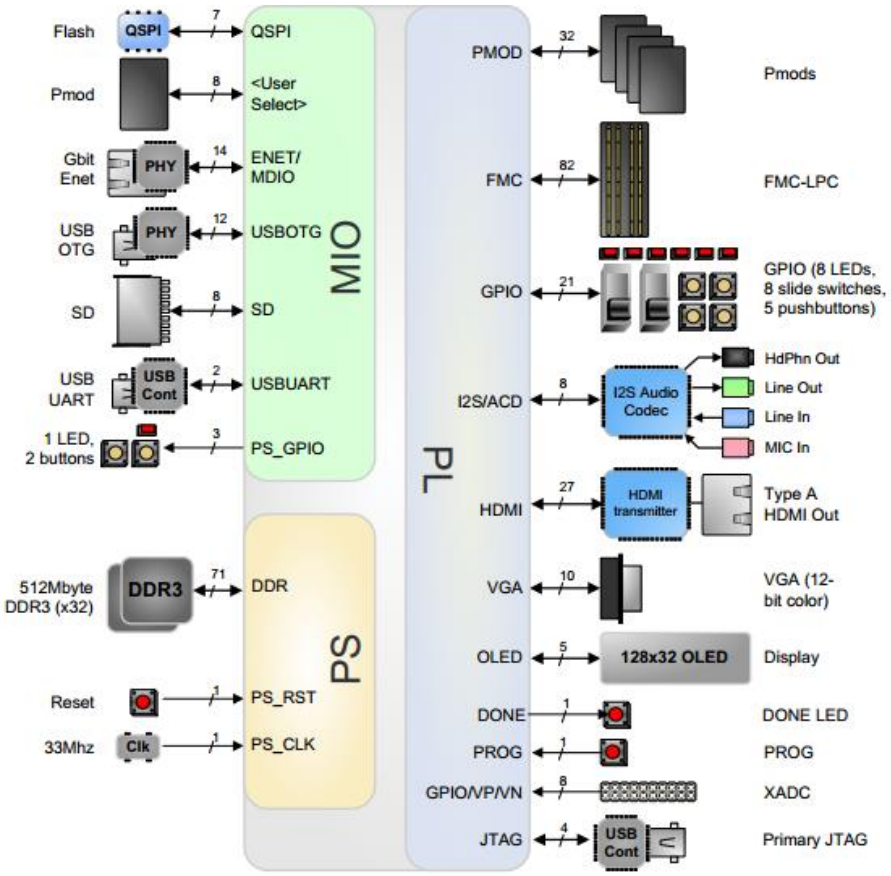


Figura 4.1 - Diagrama de blocos da ZedBoard [10]

4.1.2. Xilinx Zynq-7000 SoC

A família Zynq-7000 é baseada na arquitetura Xilinx programáveis SoC (AP SoC). Como já referido anteriormente trata-se de um conjunto de componente responsáveis pelo processamento do sistema aliado à lógica programável da Xilinx num só dispositivo. No que diz respeito ao processamento de sistema, para além do processador dual-core ARM Cortex-A9 MPCore, inclui também memória *on-chip*, interface de memória externa e um elevado conjunto de periféricos I/Os.

A Figura 4.2 ilustra os blocos funcionais da Zynq-7000 AP SoC. O PS e o PL têm domínios de alimentação separadas, possibilitando que o utilizador do dispositivo use a lógica programável para gestão. O processador no PS inicializa sempre primeiro. O PL pode ser configurado como parte do processo de inicialização ou configurado num momento posterior.

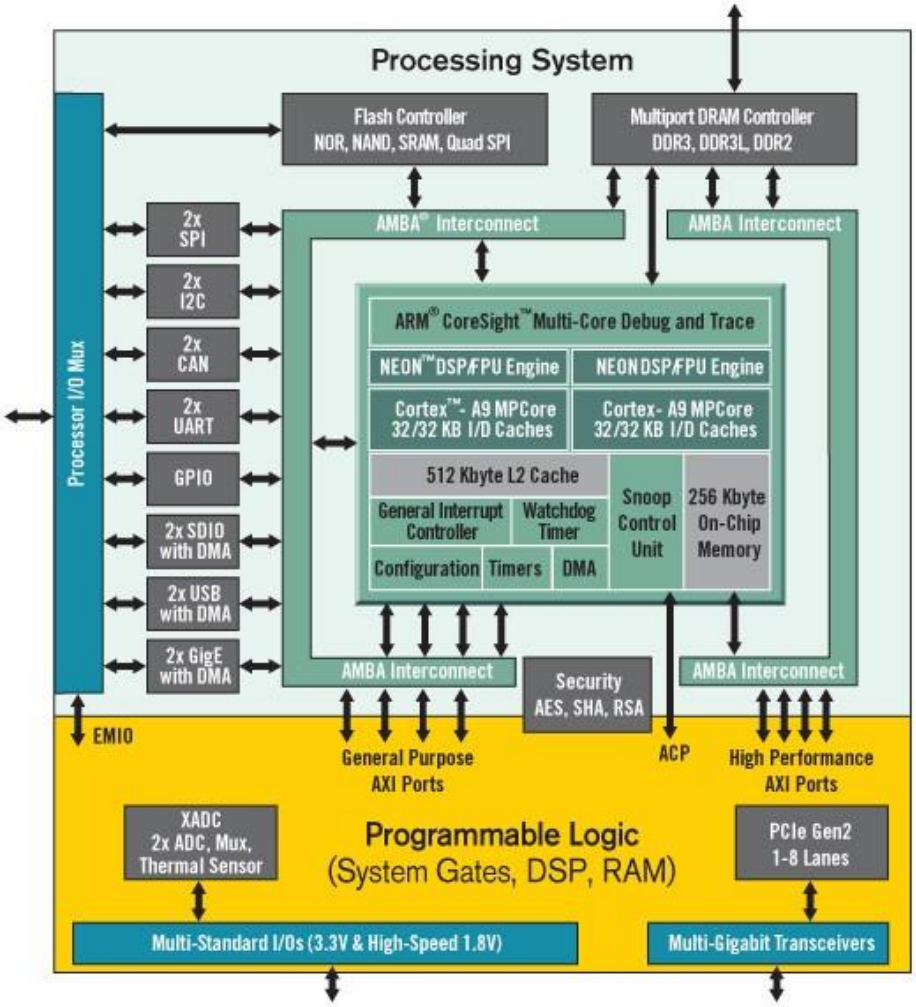


Figura 4.2 - Diagrama Zynq-7000 [11]

4.1.2.1. Zynq-7000 *Processor Overview*

O processamento do sistema é composto pelo processador, por duas caches, por uma unidade de controlo, por um acelerador de *software* e por um coprocessador. O processador ARM tem uma velocidade de processamento de 2.5 DMIPS/MHz por core que opera a uma frequência até 1GHz. A cache L1 tem 32KB de instruções e 32KB de dados, tendo cada *word* um comprimento de 32 bytes. A cache L2 advém da necessidade específica de uma cache de controlo com elevada performance e tem como características:

- 512KB de tamanho de cache.
- Suporta *lockdown*, sendo este um meio para bloquear as linhas da cache, útil em aplicações que necessitem de reduzir a latência.
- Suporta paridade.

- AXI interfaces
    - 1 AXI *master* interface para o controlador DDR.
    - 1 AXI *master* interface para todos os dispositivos presentes na PL e no PS.
    - 1 AXI *slave* interface para a unidade controlo.

Também incluído na unidade de processamento encontra-se a unidade de controlo. A *Snoop Control Unit* (SCU) conecta os dois processadores Cortex-A9 ao sistema de memória/periféricos via AXI interface. É responsável por:

- Gerir o recurso à cache L1 entre os processadores cortex-A9 e o Accelerator Coherence Port.
- Definir o acesso dos processadores cortex-A9 à cache L2.
- Fornecer acesso ao *on-chip* ROM e à RAM.
- Gerir os acessos ao Accelerator Coherence Port.

Quanto ao *Accelerator Coherence Port* (ACP) tem as seguintes características:

- É um *slave* AXI 64-bit *port* pertencente ao SCU fazendo a conexão com a PL.
- Pode ser usado por aceleradores que se encontrem na PL para aceder às caches L1 e L2 do ARM.
- O acesso às caches pode ser usada para a partilha de dados entre o CPU A9 e o acelerador da PL com baixa latência.

Por fim o coprocessador Cortex-A9 NEON e a unidade de ponto flutuante. A arquitetura ARM usa um coprocessador especifico que estende o ARM *instruction set*. As principais características são:

- NEON SIMD (*Single Instruction Multiple Data*) *engine accelerator* para aplicações multimédia.
- Instruções requerem menos ciclos de *clock* que o ARM:
    - NEON tipicamente reduz para metade o número de ciclos de relógio para uma determinada aplicação.
- Habilitado via *software*.

A *floating point unit* (FPU) *Engine* é outro coprocessador usado como uma extensão do NEON coprocessador. As principais características são:

- Elevada performance *single/double precision* nas operações de *floating point.*
- *Register set* partilhada com NEON.
- 2 MFLOPS/MHz performance.

4.1.2.2. *Programmable Logic Overview*

A PL é proveniente da tecnologia Xilinx 7 series FPGA (no caso da ZedBoard Artix-7). Inclui diversos tipos diferentes de recursos incluindo blocos de lógicas configurável (CLBs), blocos de RAM configuráveis (BRAM), DSP *slices* com 25x18 *multiplier*, 48-bit *accumulators* e *pre-adder*, um conversor analógico digital configurável (XADC), um *clock management tiler* (CMT), um bloco de configuração com 256b AES para descriptografia e SHA para autenticação, I/O configuráveis, entre outros. Todas esta informação está disponível em [12].

4.1.3. Sistema de Atuação

O *hardware* abrangido quer no sistema de aquisição quer no sistema de atuação não foram desenvolvidos no decorrer desta dissertação. Porém é crucial conhecer os componentes que estão integrados nesses sistemas a fim de se poder estabelecer comunicações com os mesmos.

O sistema de atuação tido em consideração é constituído por quatro DACs AD5791 cada um com o respetivo *switch* digital ADG1633. Espera-se que com o recurso a estes componentes se tire melhor partido do sistema para a obtenção dos melhores resultados, sendo esse o fator de escolha destes dispositivos. Assim, na escolha dos mesmos foi tido em conta diversos fatores entre os quais a resolução, a velocidade do protocolo de comunicação, o ruído e o preço.

4.1.3.1. Conversor Digital/Analógico - AD5791

O conversor digital analógico usado foi um conversor de 20 bits DAC do fabricante Analog Devices cuja função é fornecer um sinal com elevada resolução podendo operar a altas frequências. Assim, tendo em conta os parâmetros requeridos este conversor é uma opção viável para a aplicação desejada.

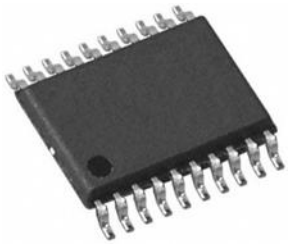


Figura 4.3 – Analog Devices AD5791 [13]

Para além da considerável resolução, permite *clocks* na ordem dos 35MHz, o que tendo em conta os 20-bit de resolução mais os 4 bits de configuração traduz-se numa frequência de atuação de 1,475MHz. Resumidamente as principais características são:

- Resolução de 20-Bits
- Escala de tensão de saída: ±14V
- ±16,5V de alimentação
- Consumo até 5,2mA
- Protocolo de comunicação SPI
- Frequência máxima de amostragem: 1,475MHz

Antes de se iniciar qualquer operação de escrita, é necessária fazer a configuração do DAC, sendo definido o modo de operação normal num endereço estipulado. Posteriormente no envio de dados é necessária a configuração ilustrada na Figura 4.4 onde o bit mais significativo representa o tipo de operação, os três bits que se sucedem correspondem ao endereço ao qual se pretende realizar a operação e os restantes bits correspondem ao dado a transmitir ou a receber.

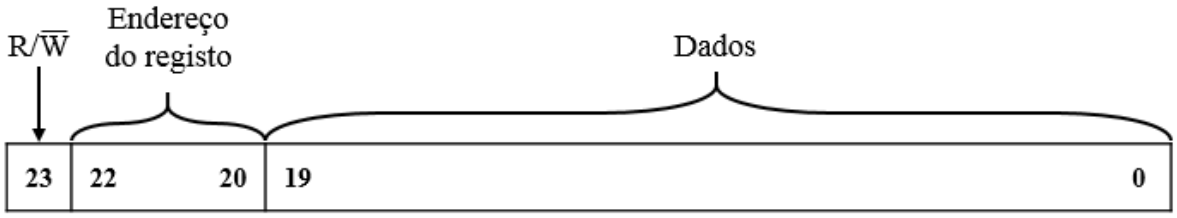


Figura 4.4 - Formato do registo de dados do AD5791 [13]

4.1.3.2. *Switch* Digital - ADG1434

Existe a necessidade de usar *switches* digitais que permitam a alternação entre o valor de tensão existente à saída do conversor D/A e a massa. Nesse segmento encontra-se o IC da Analog Devices ADG1434. Como se pode observar na Figura 4.5 o dispositivo possuí 4 *switches* digitais cada um com um bit para sinal de controlo e um bit para sinal de *enable*. Dependendo do bit de controlo da entrada D, isto é, estado *HIGH* ou estado *LOW* o *switch* alterna entre a entrada A ou B.

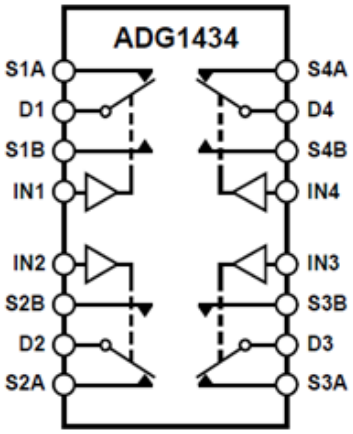


Figura 4.5 – Diagrama funcional do *switch* digital ADG1434 [14]

Este dispositivo tem uma baixa impedância e um tempo de resposta curto na comutação de canais. Em seguida apresentam-se as principais características do *switch*:

- Resistência interna 4.7Ω
- Corrente máxima permitida 115mA
- Dois modos de alimentação:
  - ±5V/±15V no modo dupla alimentação
  - 12V no modo alimentação única
- Tempo de transição 140ns.

4.1.4. Sistema de Leitura

A aquisição dos sinais provenientes do sensor é uma das partes críticas do sistema visto que pode limitar a performance de todo o sistema. Assim, um bom sistema de leitura

caracteriza-se por ter uma frequência de aquisição que não limite os potenciais resultados, robusto ao ponto de garantir o normal funcionamento do sistema e que introduza o mínimo ruído possível, de modo a não influenciar os resultados.

A obtenção dos valores pode ser realizada de diversos modos, recorrendo a comparadores analógicos que comparam o deslocamento da estrutura a um valor de referência previamente estipulado, ou recorrendo a um conversor analógico para digital que disponibilize um valor proporcional à tensão de saída do circuito de leitura.

4.1.4.1. Circuito *Readout*

A primeira fase do sistema de aquisição de microsensores capacitivos trata-se do processo de leitura da variação de capacidades que são proporcionais ao deslocamento da estrutura. Esse processo está incumbido a um amplificador de carga que faz a modulação em amplitude da variação da capacidade. Na figura seguinte pode-se observar um esboço dos típicos módulos que constituem um circuito de leitura de sensores capacitivos.

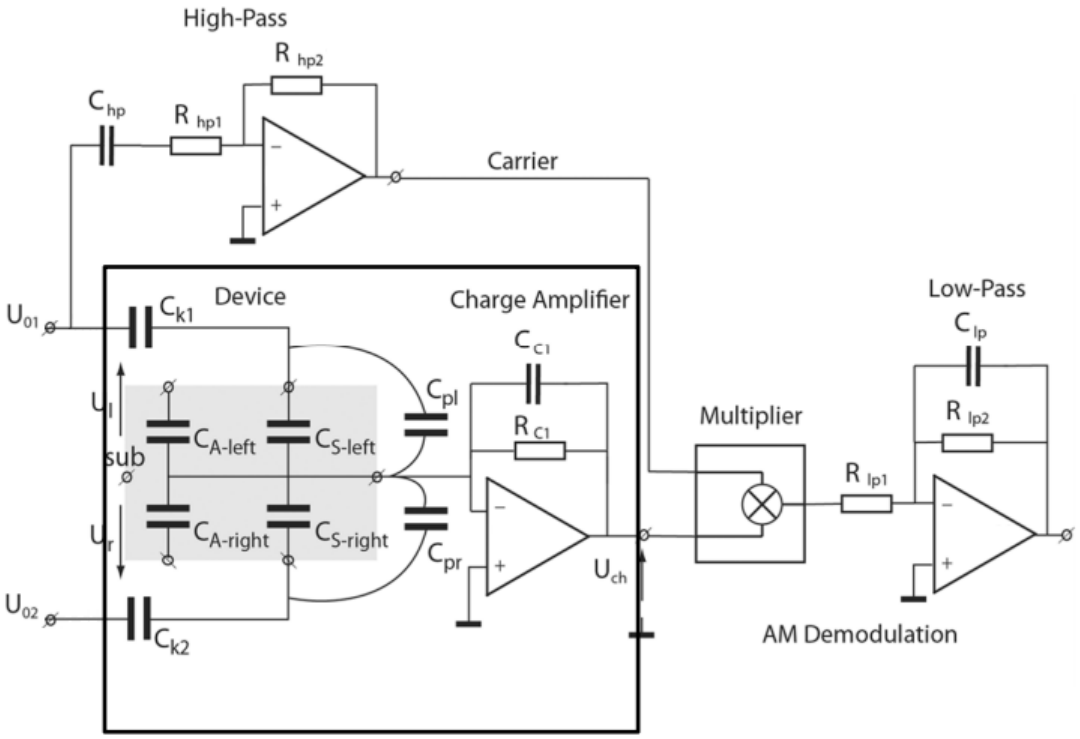


Figura 4.6 - Circuito equivalente do sensor e respetivo circuito de leitura [15]

Na Figura 4.6 está ilustrada no campo acinzentado o circuito equivalente a um sensor capacitivo que corresponde a condensadores que variam a capacidade consoante o deslocamento das estruturas. Aos terminais dessas estruturas estão ligadas as portadoras com frequência e amplitude fixas que provoca à saída do sensor uma variação de sinal com amplitude proporcional ao deslocamento. Em seguida é modulado em amplitude pelo *charge amplifier* sendo posteriormente desmodulado e filtrado.

Á saída do circuito *readout* aqui ilustrado encontra-se um sinal em tensão proporcional ao movimento do sensor (variação da capacidade). Irão ser implementados dois métodos diferentes de aquisição de sinal. A primeira topologia será a comparação da saída com uma referência analógica através de comparadores analógicos que pode ser útil por exemplo na deteção da tensão de *pull-in* que pode ser implementado para a medição de inclinações [1]. Outra implementação será a medição em valor absoluto do deslocamento das estruturas através de um conversor analógico digital [9].

4.1.4.2. Comparadores Analógicos

Para efeitos de interface do comparador com a plataforma a escolha do comparador não é relevante, uma vez que o tipo de output, isto é, dois estados de saída *HIGH* e *LOW*, é consensual entre os comparadores analógicos existentes no mercado. Assim não há a necessidade de especificação do mesmo.

4.1.4.3. Conversor Analógico Digital - ADS5560

O principal fator a ter em consideração na escolha de um conversor é a velocidade de aquisição. Quanto maior a frequência de aquisição mais eficiente será o sistema e melhores serão os resultados. Assim e tendo em conta esse fator o conversor usado foi o ADS5560 da Texas Instruments. É um conversor A/D de alta performance de 16-bit com uma elevada frequência de aquisição, na ordem de 40MSPS, o que se enquadra nos parâmetros pretendidos para a aplicação.

Figura 4.7 - Texas Intruments ADS5560 [16]

Este conversor da Texas Intruments dispõe de vários modos de funcionamento assim como vários protocolos de comunicação. Tendo em conta os objetivos desejados para a plataforma, o método que melhor se adequa para a aplicação é o modo CMOS paralelo com a informação a ser fornecida em complemento para 2 numa frequência de amostragem de 2MSPS. As principais características deste conversor A/D são:

- Tensão de alimentação: +3.3V
- Referência: Interna/Externa
- Modo de operação: *Double Data Rate* (DDR) LVDS/*Parallel* CMOS
- Resolução: 16-bit
- Taxa de amostragem máxima: 40MSPS
- Potência máxima: 674mW a 40MSPS

4.1.5. Sistema de Comunicação Externos

Neste tipo de sistema é usual realizar monitorizações ou estabelecer ligações com canais externos de modo a inicializar sequências de tarefas automáticas ou simplesmente para transferir dados. Assim é de elevada importância que através da plataforma se possa estabelecer esse tipo de comunicações. Com essa linha de pensamento decidiu-se implementar em *hardware* um canal de comunicação UART que liga a plataforma a um terminal externo.

4.1.5.1. Conversor USB para UART - FT232R

Para estabelecer a comunicação entre a plataforma e o terminal é necessário um conversor de USB para UART. Optou-se por uma implementação em FPGA em detrimento do uso da USB-OTG existente na ZedBoard, para assegurar a velocidade de comunicação e

também para não introduzir complexidade desnecessária ao sistema de *software*. O conversor escolhido foi o FT232R da Future Technology Devices International por uma questão de preço e acessibilidade.

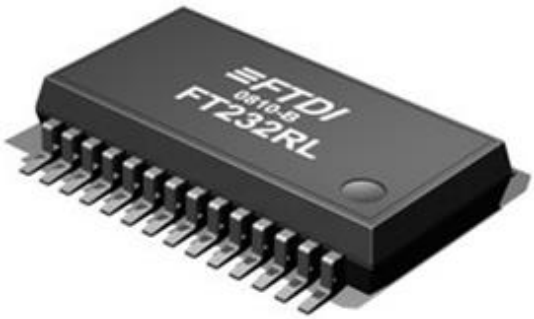


Figura 4.8 - FT232RL da FTDI no seu package SSOP [17]

De entre todas as configurações possíveis para esta aplicação conclui-se que neste caso não se justifica a implementação de todas as funcionalidades adjacentes ao dispositivo e assim o módulo será definido com um *baud rate* de 115000bps com 8-bit de dados, um *start bit*, um *stop bit*, sem bit de paridade. A alimentação será garantida pela tensão do barramento proveniente do terminal.

Seguem-se algumas das principais características do conversor da FTDI:

- Tensão de alimentação: +3.3V a +5.25V
- Memória EEPROM de 1024-bit para armazenar informações e configurações do dispositivo
- FIFO na receção e transmissão de dados para transmissões de altas frequências
- Conversor de nível entre +1.8V e +5V no UART e na CBUS
- Velocidade de transmissão de dados: 300baud[3] a 3Mbaud
- Compatível com USB 2.0 *Full Speed*.

4.1.6. Diagrama de Blocos

Na Figura 4.9 ilustra-se o diagrama de blocos de todo o sistema. Como componente central encontra-se a ZedBoard que fará a aquisição dos sinais provenientes do sistema de

[3] Medida de velocidade de sinalização que representa o número de mudanças na linha de transmissão ou eventos por segundo

leitura, tanto do ADS5560 como dos comparadores analógicos e a atuação nos DACs AD5791 e nos *switches* de controlo dos DACs.

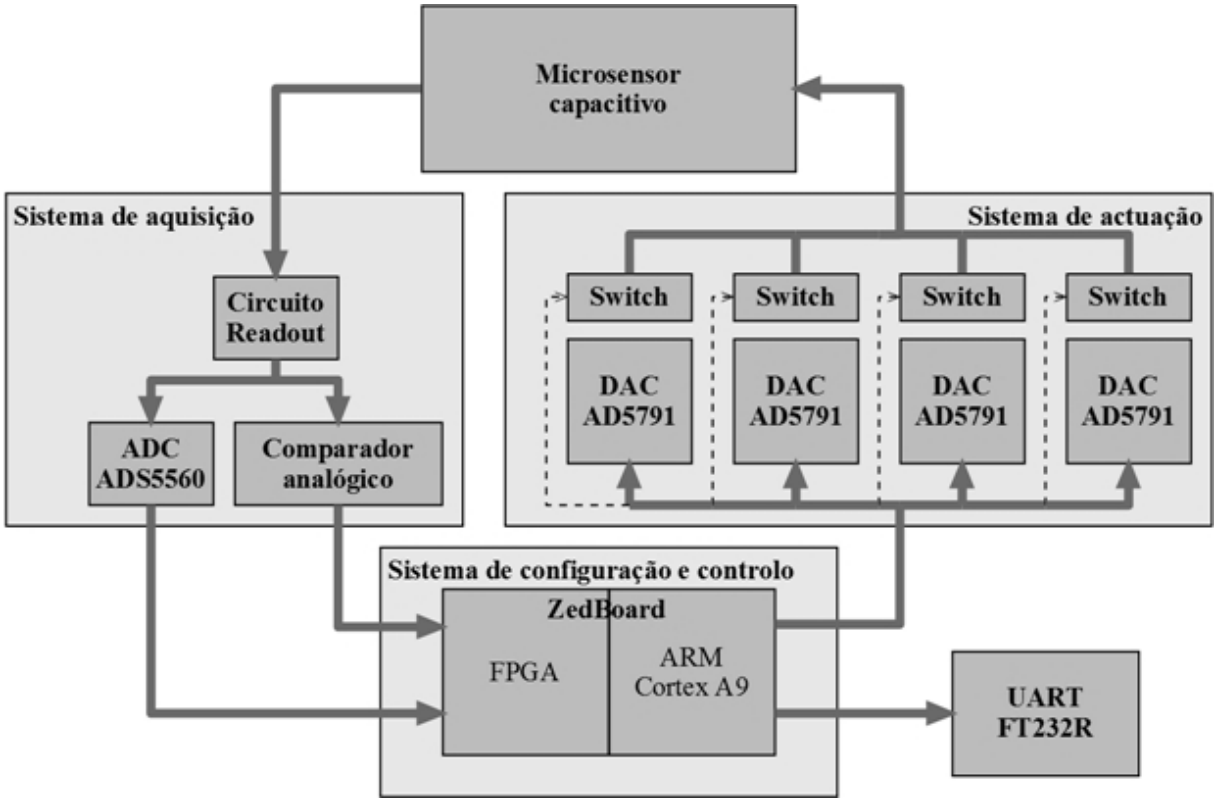


Figura 4.9 - Diagrama de blocos do sistema

4.2. *Especificação de Software*

Neste subcapítulo será abordada a distribuição Linux que irá ser incorporada na plataforma assim como os ficheiros associados. Serão também abordadas as ferramentas de *software* utilizadas ao longo do desenvolvimento deste projeto.

4.2.1. Ferramentas de *Software*

Serão descritas as funções das ferramentas de *software* utilizadas e o propósito de serem incluídas nesta dissertação. As ferramentas de *software* recorridas centram-se essencialmente nas ferramentas de *design* da Xilinx. Este *kit* de desenvolvimento é um

conjunto de ferramentas e de IP que são usadas para integrar componentes de *hardware* e *software* do sistema.

4.2.1.1. PlanAhead

Esta ferramenta de *design* e análise de *software* fornece um ambiente para o desenvolvimento da implementação no processo de design da FPGA e funciona como ferramenta base das ferramentas de design ISE na construção de todas as aplicações *System on Chip*. Tem diversas ferramentas incluídas como:

- Xilinx Embedded Development Kit (EDK)
  - Xilinx Platform Studio (XPS)
  - Software Development Kit (SDK)
- ChipScope *debugging tool*
- iMPACT *device programming tool*

Genericamente a ferramenta PlanAhead pode ser usada para:

- Gerir o fluxo de dados de *design* desde o desenvolvimento do RTL até à geração do *bitstream*
- Gerir as *constrainst* e implementa *Floorplanning*
- Implementa *Design Rule Checks* (DRC's)
- *Debug* com a ferramenta ChipScope *debugging*
- Analisa os resultados da implementação
- Implementa o planeamento dos pinos I/O
- Configura e lança múltiplas execuções de sintetizações e de implementações

4.2.1.2. XPS

Xilinx Plataform Studio é um ambiente de desenvolvimento usado para o desenho de porções de *hardware* do processador do sistema embebido. Especificações do processador,

periféricos e interconexões desses componentes, assim como as respetivas configurações, são abordados no XPS. As suas principais funções são:

- Permite a adição do núcleo do processador e respetivos periféricos, a edição dos parâmetros e implementa o barramento e sinais das conexões para geração do ficheiro *Microprocessor Hardware Specification* (MHS)
- Permite a visualização e geração do diagrama de blocos do sistema e/ou *design report*
- Suporte à gestão do projeto
- Permite o *export* dos ficheiros de especificação do *hardware* para o SDK.

4.2.1.3. SDK

A Xilinx *Software Developmento Kit* (SDK) fornece um ambiente para a criação de uma plataforma de *software* e respetivas aplicações tendo como alvos processadores embebidos das Xilinx. SDK inclui os *hardware designs* criado pelo Xilinx Platform Studio nos seus projetos e é nesse *workspace* em que se baseiam as suas aplicações.

4.2.2. Linux na ZedBoard

Inicialmente será descrito o modo como o Linux é configurado e como é iniciado na ZedBoard. Essa configuração pode ser dividida em quatro estados. Estado 0 é um estado em que não é configurável pelo utilizador. Neste estado é carregado o código executável do *First Stage Boot Loader* (estado 1), que posteriormente toma o controlo e prepara o sistema para que um *boot loader* maior possa ser carregado. O estado 2 ou U-Boot Loader aloca, faz o *load* e executa o Linux kernel. Também neste estado é transferido o *Device Tree* para o kernel. Posteriormente o sistema operativo inicializa o sistema de *hardware* e monta o *root file system*. No diagrama da Figura 4.10 ilustra-se o processo anteriormente descrito.

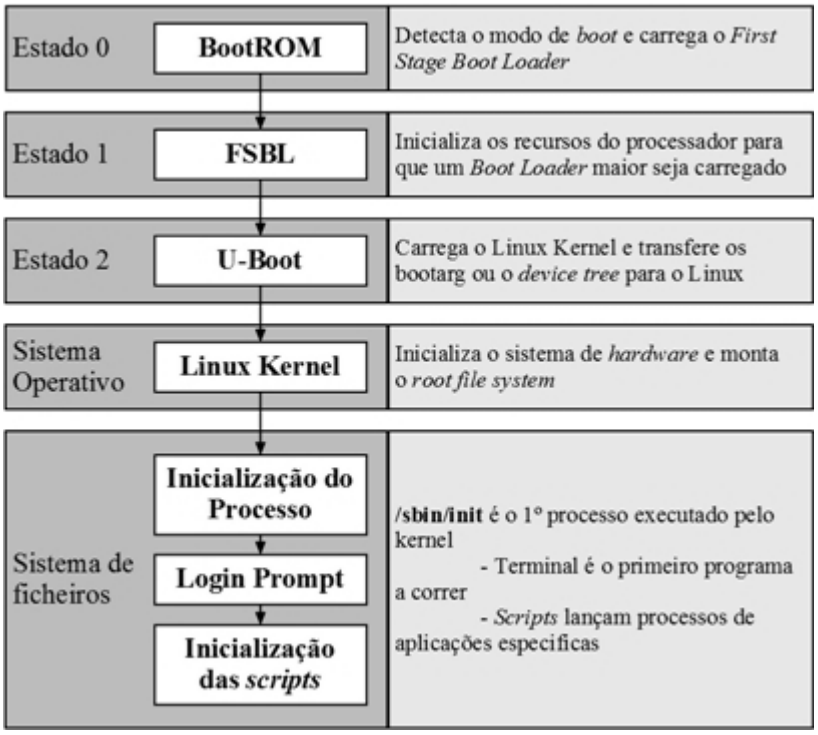


Figura 4.10 - Típica sequência de inicialização *overview*

4.2.2.1. Estado 0: BootROM

Uma memória ROM interna inerente à ZedBoard contém o código de inicialização do estado 0, onde é configurado um dos processadores ARM e os periféricos necessários para iniciar a execução do código relativo ao FSBL a partir do dispositivo de inicialização previamente selecionado. Existem 5 possibilidades diferentes de inicialização: NAND, NOR, SD-Card, Quad-SPI e JTAG. As primeiras 4 fontes são *non-secure master* e são usadas nos métodos de *master boots* em que o CPU carrega o *boot image* a partir de uma memória não volátil para o *processing system* (PS). Na ZedBoard a fonte de inicialização usada será a partir do SD-Card.

4.2.2.2. Estado 1: *First Stage Boot-Loader*

Este estado é responsável por implementar diversas funções entre as quais:

- Inicialização recorrendo aos dados de configuração PS fornecidos pelo XPS
- Programação da PL usando o *Bitstream*
- Colocação do FSBL ou do código da aplicação *bare metal* em memória

- Execução do *second stage boot-loader* ou aplicação *bare metal*.

Usando a configuração *user-interface* da Zynq-7000, o XPS gera o código para a inicialização dos registos MIO e SLCR. O *first stage Boot-Loader* é gerado pelo SDK.

4.2.2.3. Estado 2: Second Stage Boot-Loader

O U-Boot é um *open source Universal Boot Loader* que é frequentemente usado na comunidade Linux. Fornece suporte a uma larga gama de plataformas embebidas e assim com ampla variedade de arquiteturas de CPU incluindo ARM. É assim a solução disponibilizada pela Xilinx para o *second stage Boot-Loader*.

Este consegue obter a imagem do *kernel* a partir do SD-Card como é pretendido. Por defeito, o U-Boot inicializa um processo chamado de *autoboot*, que verifica a configuração dos pinos de *BootMode* para tomar conhecimento da fonte da imagem do *kernel*, do *device tree blob* e pelas imagens do ficheiros do sistema.

Para este sistema será utilizada uma versão da Digilent que criou a sua própria versão modificada de modo a suportar o seu *hardware* [18].

4.2.2.4. *Boot Image*

Usando o FSBL gerado pelo SDK, o ficheiro *bitstream* correspondente ao *hardware* e o ficheiro **U-Boot.elf**, que é fornecido pelo repositório da Digilent e compilado numa plataforma *cross bluild*, o *boot image* será criado, recorrendo ao SDK. O ficheiro resultante da junção dos ficheiros descrito será posteriormente incorporado no SD-Card.,

4.2.2.5. *Device Tree*

O Linux Kernel é um *software standalone*[4] que corre no *hardware*. Este fornece um interface *standard* para aplicações que necessitem de recursos de *hardware* sem o conhecimento detalhado dos mesmos. Porém, o kernel necessita de conhecer todos os detalhes

[4] Programa completamente autosuficiente

do *hardware* que utiliza. Para ultrapassar esta barreira, este recorre a uma estrutura de dados conhecida como *device Tree Blob* ou *Device Tree Binary* para descrever o *hardware*.

O DTB é uma base de dados onde estão descritas as características e localizações dos componentes de *hardware* da *board* em questão. É normalmente fornecido pelos fabricantes das placas/arquiteturas como parte do Linux Kernel *source tree*. Porém caso seja adicionado algum *hardware*, o DTB deve ser customizado também.

4.2.2.6. Linux Kernel

O Linux Kernel é um sistema operativo unix, que está usualmente na forma de distribuições Linux, sendo distribuído sob a licença da GNU General Public License e desenvolvido por colaboradores de todo o mundo. Implementa todas as funcionalidades espectáveis de um sistema operativo moderno unix, incluindo *multitasking*, memória virtual, livrarias partilhadas, execuções partilhadas *copy-on-write*, gestão de memória otimizada, etc.

Há uma distribuição oficial Linux Kernel, porém existem muitas outras distribuições que foram surgindo à medida que se desenvolvem projetos específicos como é o caso da plataforma da Digilent e que se encontra disponível nos seus arquivos.

4.3. *Especificação módulo de interface*

Neste subcapítulo são explicitados os módulos de interface instanciados ou desenvolvidos na ZedBoard por forma a garantir a diversas comunicações, quer com o *hardware* externo, quer nas comunicações internas entre a FPGA e o microprocessador.

4.3.1. Módulos de interface externos

Serão abordados todos os módulos que terão de ser implementados na FPGA desde os responsáveis por garantir o funcionamento de todas as comunicações com o *hardware* externo até à implementação do filtro digital.

4.3.1.1. Reset

Um dos módulos mais críticos no desenvolvimento de projetos em FPGA, sendo muitas vezes ignorado, é a necessidade de implementação de um *reset* do sistema. Nestes projetos é usualmente integrado um *reset* que é comum a todos os módulos e IP implementados. Acontece que cada *designer* de FPGA fornece a sua própria livraria, relativa aos elementos que desenvolve, e implementa, quer os *resets* síncronos quer os *resets* assíncronos, segundo a sua própria topologia. Esta diferença de implementação pode originar erros de várias ordens, sendo que consiste num erro que dificilmente se repete, tornando-o muito difícil de detetar [19].

Derivado da informação referida anteriormente, será então implementado um módulo que habilite todos os *resets* assincronamente e que posteriormente os desabilite em sincronismo com todos os *clocks* pertencentes ao sistema. Evitam-se assim problemas de sincronismos que hipoteticamente poderiam surgir.

4.3.1.2. PWM

Um parâmetro não abordado no subcapítulo anterior foi a geração das portadoras de sinal, entradas do circuito de *readout*. As portadoras nada mais são que duas ondas sinusoidais desfasadas de 180º. Um gerador de sinal desempenhava perfeitamente esse papel, mas implicaria sempre o acompanhamento do mesmo junto do sistema de *hardware* o que traz uma desvantagem. Desse modo definiu-se que a geração do sinal das portadoras seria implementada na FPGA. A plataforma fornece assim duas ondas PWM com *duty cycle* variável. Posteriormente é filtrada por um filtro ativo resultando nas duas ondas sinusoidais desejadas.

A frequência definida para as portadoras foi de 1MHz. Na geração das ondas PWM um critério de elevada relevância é a frequência de *clock*. Quanto maior a frequência do *clock* maior será a resolução das ondas produzidas. Analisando as I/O PLL disponíveis [20] na ZYNQ-7000 observa-se que conseguem gerar um *clock* máximo de 250 MHz o que corresponde a 250 ciclo máquina para a geração de uma onda de 1MHz. A topologia adotada foi a alteração do valor de *duty cycle* segundo um critério tal que se aproxime a uma onda PWM.

4.3.1.3. SPI

O DAC utilizado usa o protocolo SPI, e como tal foi preciso estabelecer a comunicação entre a ´FPGA e a conversor. O interface SPI é um barramento desenvolvido pela Motorola que opera no modo *Full Duplex* ou *Half Duplex*. É normalmente utilizado para comunicações curtas onde sejam necessárias grandes velocidades. O sincronismo é garantido pelo *clock* do sistema que é gerado pelo *master*. A interface do SPI permite a conexão de um ou vários dispositivos *slaves* a um único dispositivo *master* recorrendo ao mesmo barramento.

Possui 4 linhas de sinal:

- MOSI (*Master Output, Slave Input*)
  - Linha MOSI é usada para transferir dados do *master* para o *slave*.
- MISO (*Master Inpur, Slave Output*)
  - Linha MISO é usada para transferir dados do *slave* para o *master*.
- SCLK (*Serial Clock*)
  - SCLK *clock pulse output* do *master* e *clock pulse input* do *slave*. A linha SCLK é usada para sincronizar a transferência de dados entre o *master* e o *slave* através das linhas MOSI e MISO.
- SS (*Slave Select*)
  - O dispositivo SPI *master* usa esta linha para seleccionar o dispositivo SPI *slave*. SS é activo no estado LOW.

Na Figura 4.11 encontra-se ilustradas as linhas de comunicação entre um dispositivo *master* e um ou vários dispositivo *slave* que uma comunicação SPI contém.

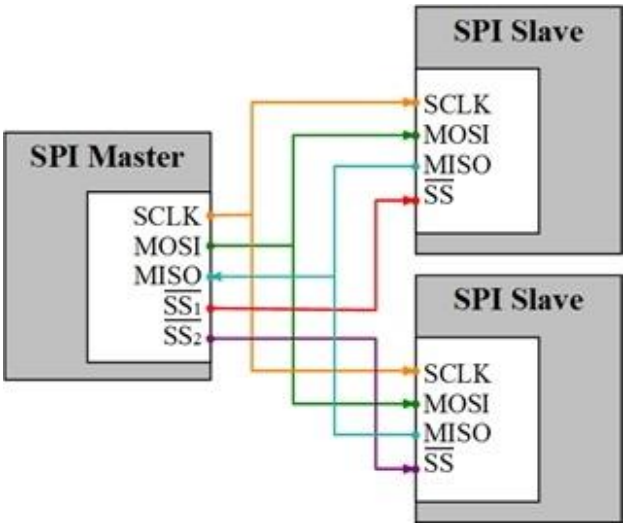


Figura 4.11 - Esquema barramento SPI

A transferência de dados é realizada *bit* a *bit*, começando no estado HIGH. A linha SS do *slave* deve estar num estado de alta impedância caso não seja selecionado.

4.3.1.4. UART

Outra comunicação que é necessário estabelecer é a comunicação série com o FT232R. Denomina-se comunicação série o processo de transmitir bytes de dados através do envio de bits individuais sequencialmente para um terminal que reagrupa os bits e reconstrui o byte enviado. Existem dois tipos de comunicação série:

- UART (Universal Asynchronous Receiver/Transmitter)
- USART (Universal Synchronous-Asynchronous Receiver/Transmitter)

A comunicação série a implementar será a UART uma vez que é a comunicação requerida pelo FT232R. Esta comunicação assíncrona não necessita de uma linha de *clock* para sincronizar o transmissor com o recetor. Para grandes quantidades de dados a transmitir esta comunicação não é a ideal uma vez que com o aumentar dos *bytes* transmitidos poderá ocorrer uma dessincronização de dados. Porém a quantidade de dados que se pretende transmitir sequencialmente será usualmente de 1 *byte*, não havendo risco de dessincronismo.

Na Figura 4.12 encontra-se ilustrado uma típica sequência de *bits* de uma comunicação UART.

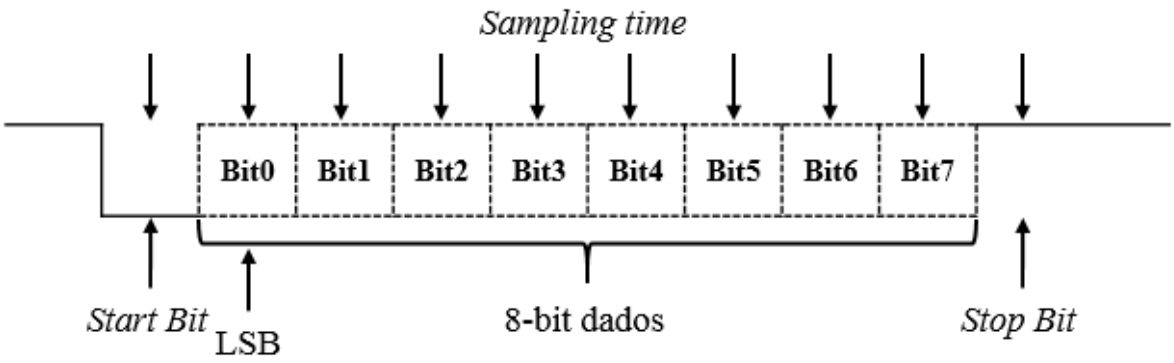


Figura 4.12 - Sequência de bits UART

4.3.1.5. Filtro Digital FIR

Caso a aquisição seja desempenhada pelo ADC, o valor adquirido poderá ter associado ruído, sendo que parte do ruido advém da portadora devido à desmodulação do sinal no circuito *readout*. Surge assim, a necessidade de desenvolver um filtro digital na FPGA que atenue o efeito do ruido.

Para o efeito pretendido definiu-se a implementação de um filtro digital FIR. Dado que utiliza para o cálculo de saída apenas combinações de entradas passadas e da entrada atual é um filtro não recursivo. Estes filtros são sempre estáveis uma vez que apresentam uma resposta finita o que significa que a resposta anula-se ao fim de um determinado ciclo de relógio. Com a utilização de um filtro IIR, isto é, recursivo, estar-se-ia a introduzir um risco de instabilidade ao sistema e portanto não seria boa opção. Dentro dos diversos FIR existentes definiu-se que a construção do filtro passa-baixo será implementada com recurso ao método da janela de Kaiser. Genericamente o método da janela é a especificação de uma resposta ideal para uma gama de frequências específica. Optou-se pela implementação de um filtro de Janela de Kaiser [21] derivado da baixo complexidade que a construção deste filtro exige.

Na Figura 4.13 está ilustrado o diagrama de blocos de um filtro digital FIR.

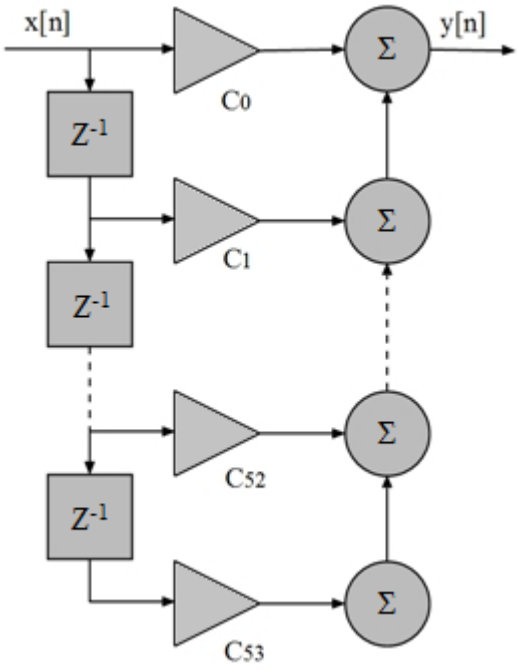


Figura 4.13 - Diagrama de blocos do filtro digital FIR

Um filtro Kaiser *Window* é definido pela seguinte equação:

$$\boldsymbol{w[n]} = \begin{cases} \dfrac{\boldsymbol{I_0}\left[\boldsymbol{\beta}\left(\mathbf{1}-\left[\dfrac{(\boldsymbol{n}-\boldsymbol{\alpha})}{\boldsymbol{\alpha}}\right]^{\mathbf{2}}\right)^{\frac{\mathbf{1}}{\mathbf{2}}}\right]}{\boldsymbol{I_0(\beta)}}, & \mathbf{0} \le \boldsymbol{n} \le \boldsymbol{M}, \\ \mathbf{0}, & \boldsymbol{sen\tilde{a}o}. \end{cases}$$

Onde $\alpha = \frac{M}{2}$ e $I_0(.)$ representa a função *zeroth-order modified* Bessel do primeiro tipo. Para o cálculo do parâmetro β são usadas as seguintes equações:

$$\Delta w = w_s - w_p$$

$$A = -20\log_{10}\delta$$

$$\beta = \begin{cases} 0.1102(A-8.7), & A > 50, \\ 0.5842(A-21)^{0.4} + 0.07886(A-21), & 21 \le A \le 50, \\ 0.0, & A < 21. \end{cases}$$

Sendo A a amplitude, w_s a frequência de corte inferior relativa à banda não passante, w_p a frequência de corte superior relativa à banda passante e Δw correspondente à região de transição.

Por fim resta calcular o número de coeficientes sendo este dado pela seguinte fórmula.

$$M = \frac{A-8}{2.285\Delta w}$$

Com a implementação das equações acima abordadas é possível calcular um número de coeficientes *M* que são transmitidos para a FPGA, preenchendo assim os coeficientes presentes na FPGA

4.3.2. Comunicação PS/PL

Analisando as funções que se pretendem implementar na ZedBoard, repara-se na necessidade de utilizar IP *cores* da Xilinx que, através dos barramentos existentes na placa, estabeleçam conexão entre a FPGA e o sistema operativo. Assim, para cada módulo de comunicação que se pretende implementar na FPGA haverá um módulo IP que estabeleça essa conexão.

4.3.2.1. Protocolo AXI Interconnect

Na gama de dispositivos Zynq-7000 da Xilinx foi adotado o protocolo Advanced eXtensible Interface (AXI) utilizado pela maioria dos *Intellectual Property* (IP) *cores* disponíveis na plataforma. Este protocolo pertence ao ARM AMBA que é um *standard* de comunicação On-Chip concebido para microcontroladores *de alta performance*.

Na Zynq-7000 está incluído o ARM AMBA 3.0 *interconnect*. Este localiza-se dentro do PS e tem como função ligar os recursos do sistema, usando os canais AXI ponto a ponto. É composto por linhas de endereços, dados e resposta de transições nas comunicações entre clientes *master* e *slave*.

A implementação do AXI traz diversos benefícios no que toca a produtividade, flexibilidade e também disponibilidade, uma vez que muitos IPs suportam o protocolo AXI. Permite a otimização de sistemas no que toca a frequência máxima, máximo *throughput*, baixa latência e combinações destes atributos. Esta flexibilização leva à otimização do produto sendo este parâmetro de extrema importância aquando da entrada no mercado. Aquando da sua utilização num sistema, a integração de IPs no referido sistema é facilitada, uma vez que no que toca ao recurso de barramentos apenas é necessário o domínio de uma família de interfaces (AXI).

As especificações do AXI baseiam-se no interface entre um único AXI *master* e um único AXI *slave* que representam núcleos IP que trocam informações entre si. Caso o protocolo AXI em causa seja *memory mapped,* as interfaces AXI *masters* e *slaves* podem ser conectadas em conjunto recorrendo a uma estrutura denominada de *Interconnect Block*. Os dados podem movimentar-se entre *master* e *slave* simultaneamente em ambas as direções (*full-duplex)*, uma vez que são fornecidos canais separados de dados e de endereçamento, tanto para a leitura como para a escrita.

Concretamente no que diz respeito ao interface PS/PL o *interconnect* possui três tipos de interface:

- AXI_ACP, uma porta *master cache coherent* para o PL.
- AXI_HP, quatro portas *master high performance* para o PL.
- AXI_GP, quatro portas de propósito geral (duas portas *master* e duas portas *slave).*

4.3.2.2. AXI GPIO

Este IP *core* da Xilinx define-se por canais de 32-bit que fazem o interface entre *inputs* e/ou *outputs* de propósito geral recorrendo ao protocolo de comunicação AXI4-Lite. São geralmente usados para importação e exportação de pequenas quantidades de dados entre o PS e o PL sendo o mecanismo mais indicado para este fim, segundo o fabricante [22]. Pode ser usado no modo *single* ou *dual channel* em que cada canal pode ser configurado de forma independente. As portas são configuradas dinamicamente como *input* ou *output* através de um buffer de 3 estados. Os canais podem também ser configurados para gerar interrupções quando ocorrer uma transição de estado na entrada. Na Figura 4.14 está presente o diagrama de blocos do IP descrito.

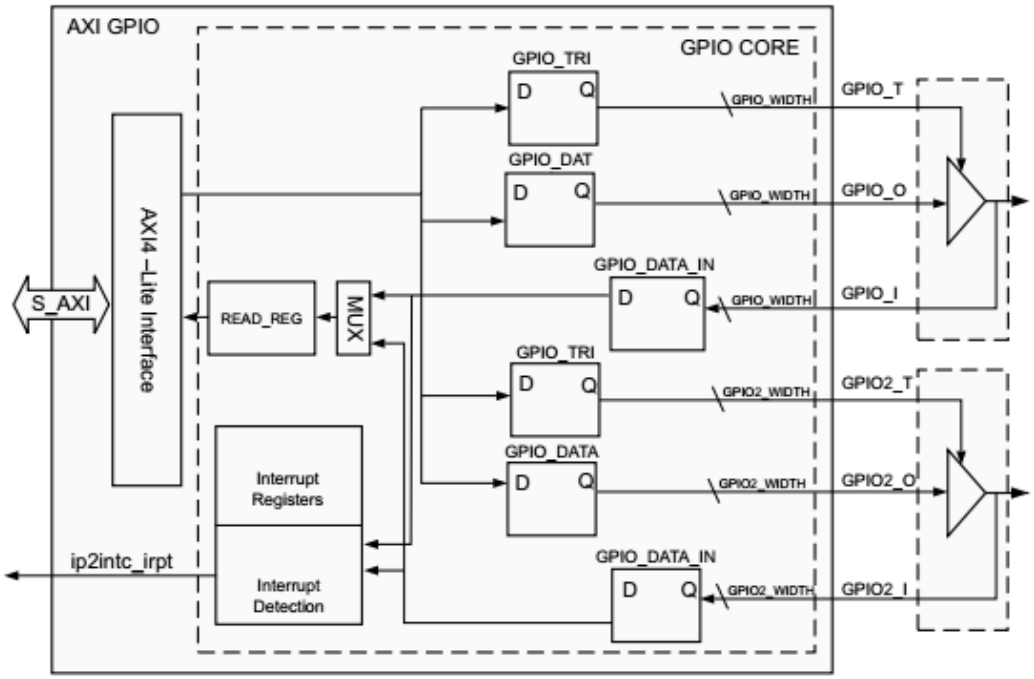


Figura 4.14 - Diagrama de blocos do AXI GPIO [22]

Assim, e analisando os módulos que se pretendem implementar na FPGA, observa-se que o AXI GPIO é adequado para estabelecer a interface do PS com os seguintes módulos:

- Módulo DAC responsável pela comunicação com o CS4334
- Ligação direta aos comparadores analógicos
- Modulo UART que estabelece ligação ao FT232R.

4.3.2.3. DMA Engine

O núcleo AXI *Direct Memory Access* (DMA) é um IP disponibilizado pela Xilinx que permite uma elevada largura de banda no acesso direto à memória entre o AXI *memory*

mapped e os periféricos IP, possibilitando a movimentação de um grande volume de dados a grande velocidade.

Este módulo fornece suporte a dois modos de operação sendo eles o modo *Independent Scatter/Gather* DMA e o modo *Simple* DMA. No contexto desta dissertação o modo de operação definido é *Simple* DMA que apesar de oferecer pior performance que o outro modo, requer menos recursos por parte da FPGA. A sua principal função será transferir os dados adquiridos na FPGA para o sistema operativo onde através de uma aplicação se possa processar os respetivos dados.

Juntamente com este IP, será integrada uma porta *high performance* (HP) 32 bit *slave* do *processing system*. Neste sistema, o AXI DMA atua como um dispositivo *master* que copia um *array* de valores provenientes de uma fonte de dados em *hardware* e copia-os para um *buffer* localizado numa região específica do sistema de memória DDR. O AXI DMA usa deste modo o *processing system* HP *slave port* para obter acesso à DDR ao nível da leitura e escrita.

A Zynq-7000 fornece internamente quatro interfaces AXI HP *slave*. Estas encontram-se conectadas ao barramento *master* PL. A principal vantagem na utilização deste módulo reside na sua capacidade de disponibilizar um elevado *throughput* entre o AXI *masters programmable logic* e o sistema de memória do *processing system* (DDR). A Figura 4.15 ilustra o diagrama de blocos do sistema aqui referido.

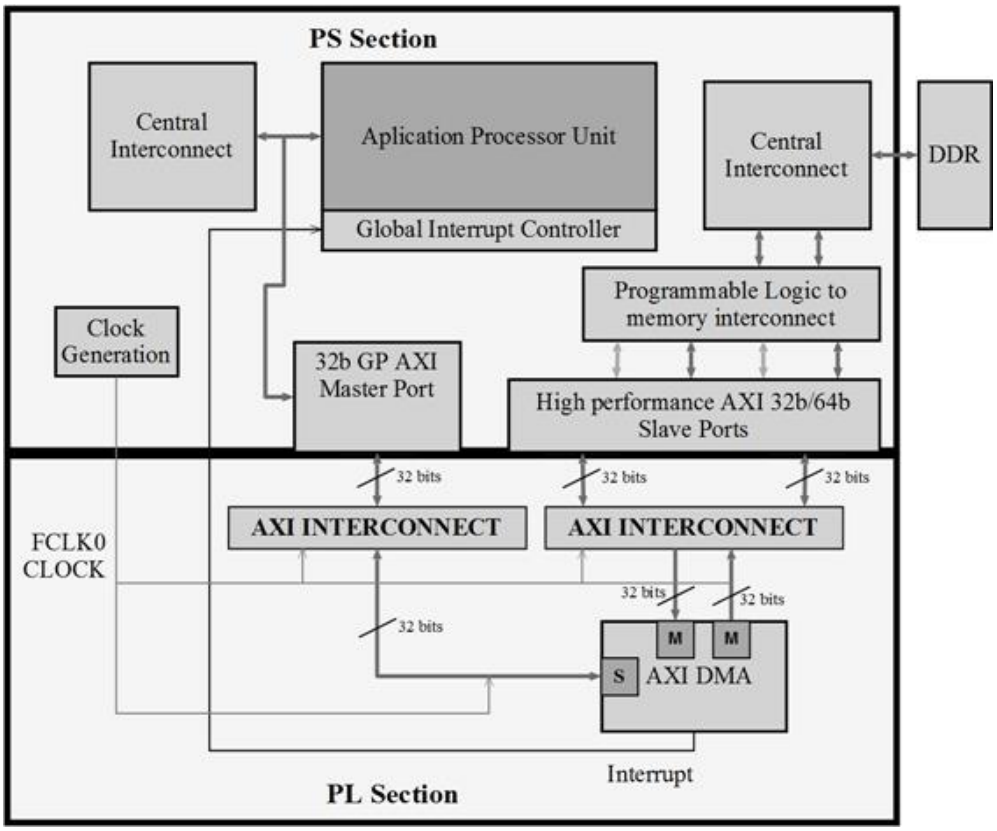


Figura 4.15 - Diagrama de interfaces entre blocos AXI DMA, AXI HP e memória DDR [23]

Este sistema estabelece as seguintes ligações:

- AXI DMA *Slave Port* é conectado ao PS *General Propose Master Port*. É usado pelo CPU para configurar os registos de configuração do AXI DMA para a transferência de dados e verificação dos estados.
- AXI DMA *Master Port* é conectado ao PS *High Performance Slave Port*. Esta conexão serve para o AXI DMA ler ou escrever os dados da/na DDR consoante a porta do DMA à qual está estabelecida a comunicação.
- AXI DMA *interrupt* é conectado a partir do IP à secção *interrupt controller*.

4.3.2.4. *Block* RAM Controller

O IP AXI *Block* RAM (BRAM) Controller é um IP *core* da Xilinx concebido como AXI *Endpoint slave* IP que integrado com o AXI *interconnect* permite a comunicação com um bloco de RAM local.

As principais características são:

- AXI4 (*memory mapped*) *slave* interface.
- Baixa latência do controlo de memória.
- Canais de leitura e de escrita separados recorrendo ao uso da tecnologia *dual port* FPGA BRAM.
- Comprimento de dados configurável (32-, 64-, 128-, 256-, 512-, e 1024-bit).
- Compatível com Xilinx AXI *Interconnect*.

A Figura 4.16 ilustra as conexões *top-level* e os módulos principais do AXI BRAM *controller* IP *core*. Este módulo pode ser configurado como *single port* ou *both ports* na conexão ao bloco BRAM, através da alteração de parâmetros.

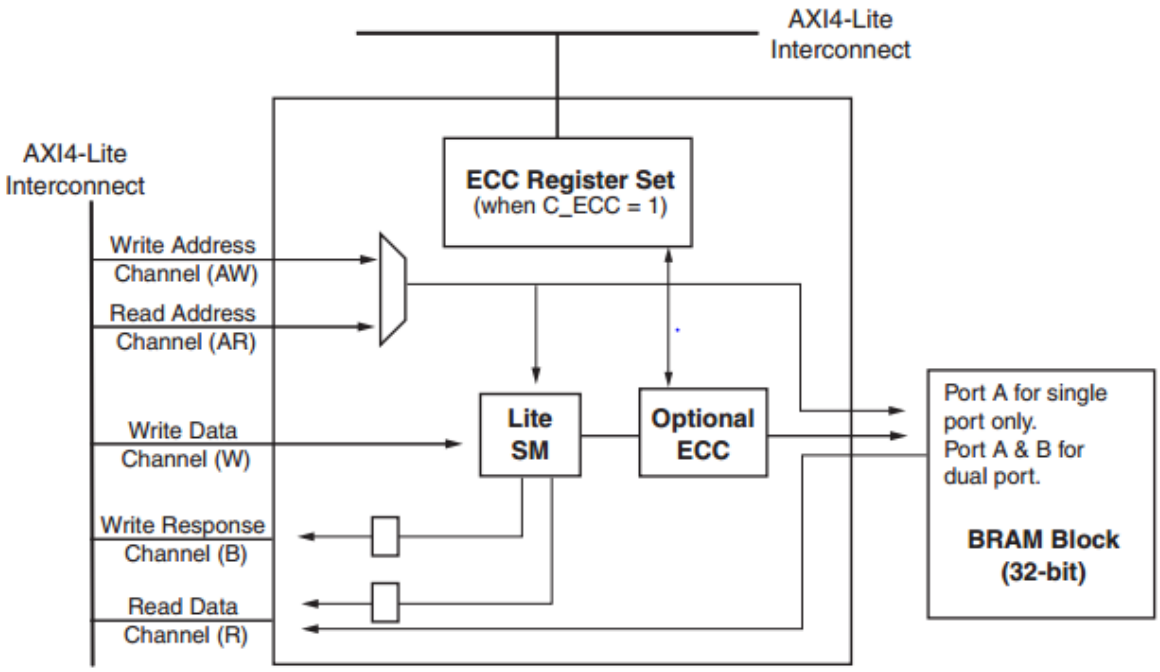


Figura 4.16 – Diagrama de blocos AXI4-Lite BRAM Controller [24]

Todas as comunicações com dispositivos AXI *master* são estabelecidas através de cinco canais AXI interface. Todas as operações de escrita são iniciadas no canal *Write Address* (AW) que especifica o tipo de escrita e a informação do endereço respetivo. Relativamente à escrita o canal *Read Address* (AR) comunica com todos os endereços e fornece informações de controlo quando o AXI *master* requere a realização de uma leitura.

O interface da BRAM é otimizado para fornecer elevado desempenho no interface de comunicação com o módulo BRAM. Se configurado como *single port*, a porta A garante todas as operações de leitura e escrita, reduzindo assim o espaço necessário à sua implementação. Se configurado como *both port*, a porta A é concebida como porta de escrita e a porta B do módulo BRAM é concebida como porta de leitura.

Sistemas gerados por integradores IP possuem um banco de blocos de RAM baseado na incorporação da RAMB18E1 ou da RAMB36E1 existentes na FPGA serie 7, incluído na Zynq-7000. O bloco de RAM é configurado recorrendo ao *Block Memory Generator IP core*.

4.4. *Casos de Teste*

Neste subcapítulo apresentam-se alguns sistemas de controlo que podem ser implementados através do uso da plataforma a desenvolver e que serão usados como casos de uso. Esses controlos a implementar serão todos de malha fechada, sendo então importante

perceber o conceito que advém da aplicação destes controladores em microsensores capacitivos.

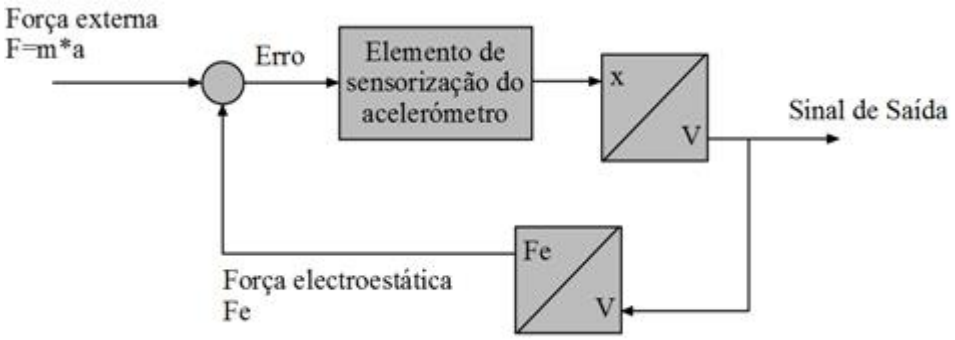


Figura 4.17 - Diagrama de blocos feedback força electroestática

A Figura 4.17 ilustra um diagrama de blocos correspondente a um modelo típico de microsensores capacitivos, onde existe uma realimentação negativa de modo a controlar o elemento sensorial. Este elemento pode ser aplicado para o desenvolvimento de acelerómetros, giroscópios, sensores de pressão, etc.

4.4.1. Controlo ON-OFF

Num atuador capacitivo, é possível usar controlo do deslocamento entre os elétrodos respetivos, para aplicações em microbombas e micromisturadoras de fluidos [25], para microsensores e microdispositivos mecânicos utilizados na indústria automóvel [26], para micropropulsores usados em microssatélites [27], entre muitas outras aplicações. Um dos controlos que se pode implementar é o controlo ON-FF.

É um controlo que consoante o valor da saída realimentado, alterna entre dois estados de comutação. O processo permite assim estabelecer uma variável controlada, que nesta implementação é a tensão à saída do circuito *readout*, em torno de uma referência previamente definida. A saída do controlador comuta entre ativada e desativada à medida que o deslocamento passa por um ponto de referência estabelecido. O elemento de controlo é um *switch* que possui somente dois estados, ligado ou desligado. A Figura 4.18 ilustra o diagrama de blocos da implementação deste controlo.

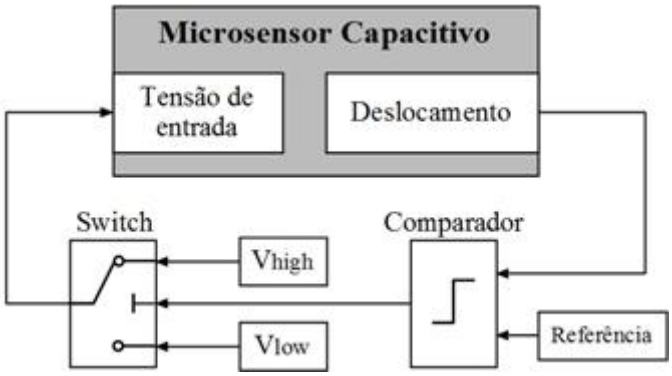


Figura 4.18 - Diagrama de blocos do controlo ON OFF

Este servirá para comparar os resultados obtidos com uma implementação de todo o sistema de controlo em FPGA e deste modo caracterizar a performance da plataforma a desenvolver nesta dissertação. Na Figura 4.19 está ilustrado o resultado esperado da implementação deste controlo.

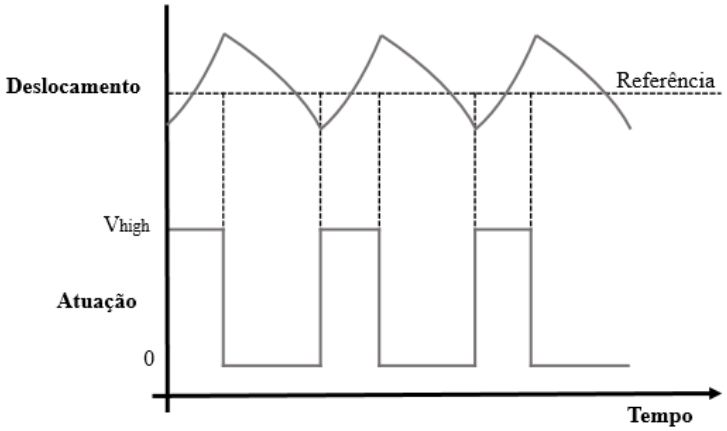


Figura 4.19 - Esquema de um controlo ON-OFF

Como alvo de teste, será usado uma microestrutura MEMS disponível no grupo de Microssistemas onde este trabalho foi desenvolvido. A microestrutura possuí dois elétrodos fixos e um móvel que se desloca entre eles. A tensão de atuação é aplicada aos terminais dos elétrodos fixos, enquanto a parte móvel se encontra ligada à massa, o que resulta numa força electroestática que faz mover a parte móvel. Apenas uma das duas tensões de entrada pode estar ativa, onde com base no ponto de referência, isto é, deslocamento positivo ou negativo em relação à origem, é acionada a tensão respetiva. Na Figura 4.20 encontra-se ilustrado o esquema da microestrutura MEMS previamente descrita.

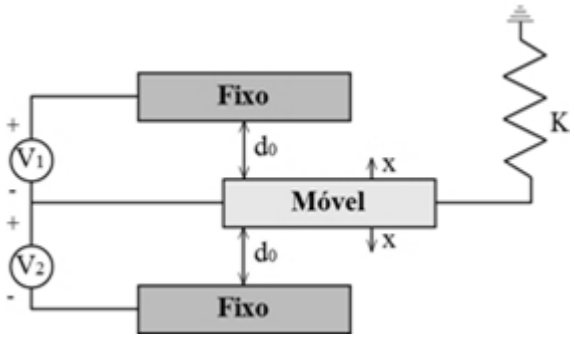


Figura 4.20 - Esquema microestrutura MEMS

4.4.2. FeedBack Linearization

Outro controlo para atestar as potencialidades da plataforma a desenvolver é o *feedback linearization*. É um controlo de malha fechada sendo a variável de controlo deste sistema a distância dos elétrodos da estrutura capacitiva. Através da realimentação negativa é feita a linearização dos valores medidos, com recurso a parâmetros inerentes à estrutura MEMS (que são previamente definidos), que juntamente com o erro em relação à referência produz um valor de atuação. Deste modo é possível controlar o dispositivo nas zonas não-lineares e instáveis. No caso dos atuadores capacitivos o deslocamento é estável apenas até a 1/3 do da distância entre elétrodos devido ao efeito de pull-in. Para deslocamentos superiores a 1/3 da distância entre elétrodos entra-se numa zona de instabilidade onde não se consegue prever o comportamento da estrutura (quando operada em malha aberta). A aplicação deste tipo de controlo é relevante na medida em que é possível controlar os deslocamentos do atuador para lá do limite estabelecido pelo efeito de pull-in.

A Figura 4.21 ilustra o diagrama de blocos do *feedback linearization control* que irá ser implementado. Mais detalhes deste controlo podem ser lidos aqui [15].

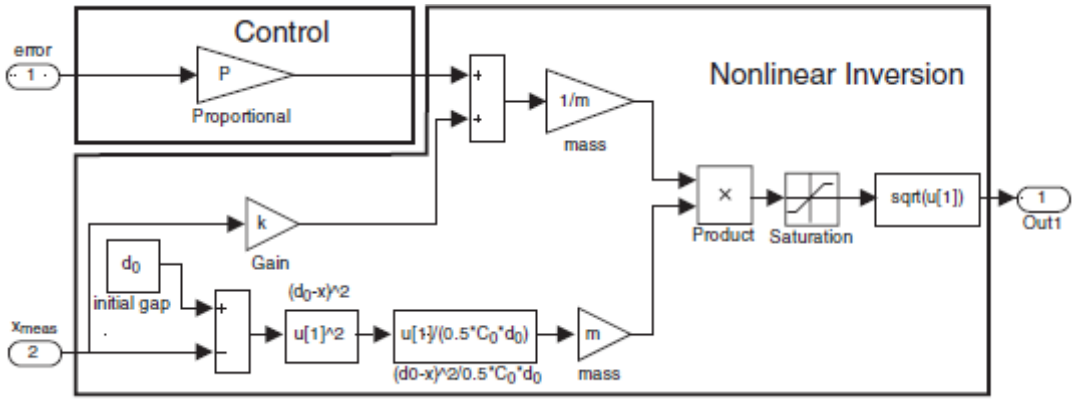


Figura 4.21 - Diagrama de blocos do controlo *Feedback Linearization* [15]

Os parâmetros indefinidos na imagem dizem respeito aos parâmetros que são intrínsecos à própria microestrutura, sendo deste modo características únicas da mesma.

Com esta implementação é testado o sistema de aquisição da plataforma, que engloba a leitura do ADC, a filtragem dos dados e envio dos mesmos para o *software*, e o sistema de atuação que incorpora a comunicação da aplicação com a FPGA e a escrita no DAC.

4.4.3. Acelerómetro baseado em modulação Sigma Delta

A modulação sigma-delta é um método de processamento digital de sinal usualmente utilizado para conversões de sinais analógicos em sinais digitais (ou vice-versa) podendo ser encontrado em ADCs ou DACs.

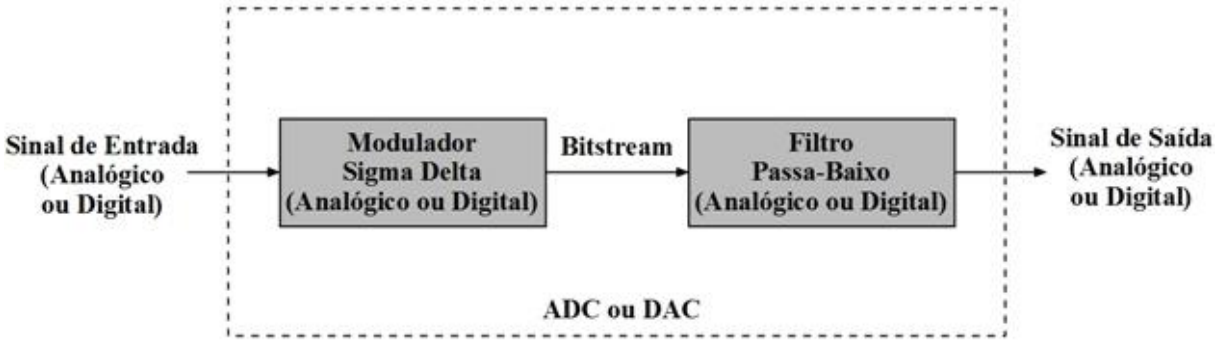


Figura 4.22- Diagrama de blocos conversão Sigma-Delta

Tal como se pode observar na Figura 4.22 um sigma delta ADC ou DAC consiste num modulador sigma delta que produz um *bitstream* seguido de um filtro passa baixo.

O modulador sigma delta é o núcleo dos conversores sigma delta cuja função é produzir um *bitstream* em que o seu nível médio é proporcional ao sinal disponível na entrada. Na Figura 4.23 encontra-se um diagrama de blocos relativo a um modulador sigma delta analógico de primeira ordem.

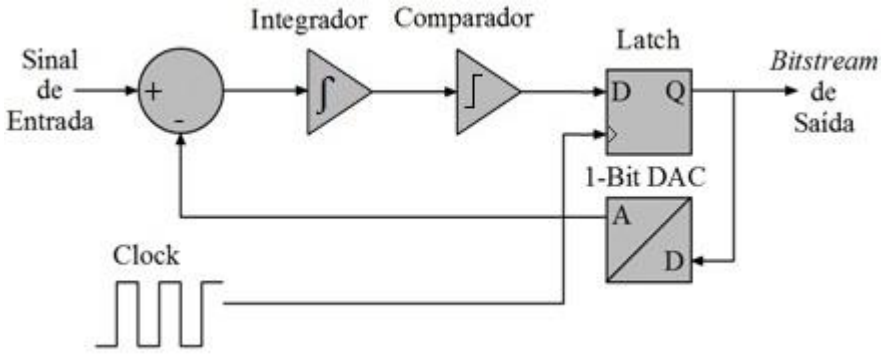


Figura 4.23 - Diagrama de blocos de um modelador sigma delta analógico de primeira ordem

Importante referir que as duas tensões de saída correspondentes respetivamente aos dois estados binários possíveis à entrada no DAC, tem que corresponder à mesma grandeza que o sinal de entrada.

Neste caso de uso o que se pretende fazer é aplicar a modulação sigma delta num controlo digital de modo a medir a aceleração. Neste caso a microestrutura integra a força resultante aplicada à estrutura, pelo que a integração é efetuada no domínio mecânico. Através do circuito de leitura é adquirido o sinal que é posteriormente comparado com uma referência, gerando um valor binário que servirá para a construção do *bitstream* e para definir em que lado da microestrutura se atua. Na Figura 4.24 pode ser observado o diagrama de blocos do sistema acima descrito.

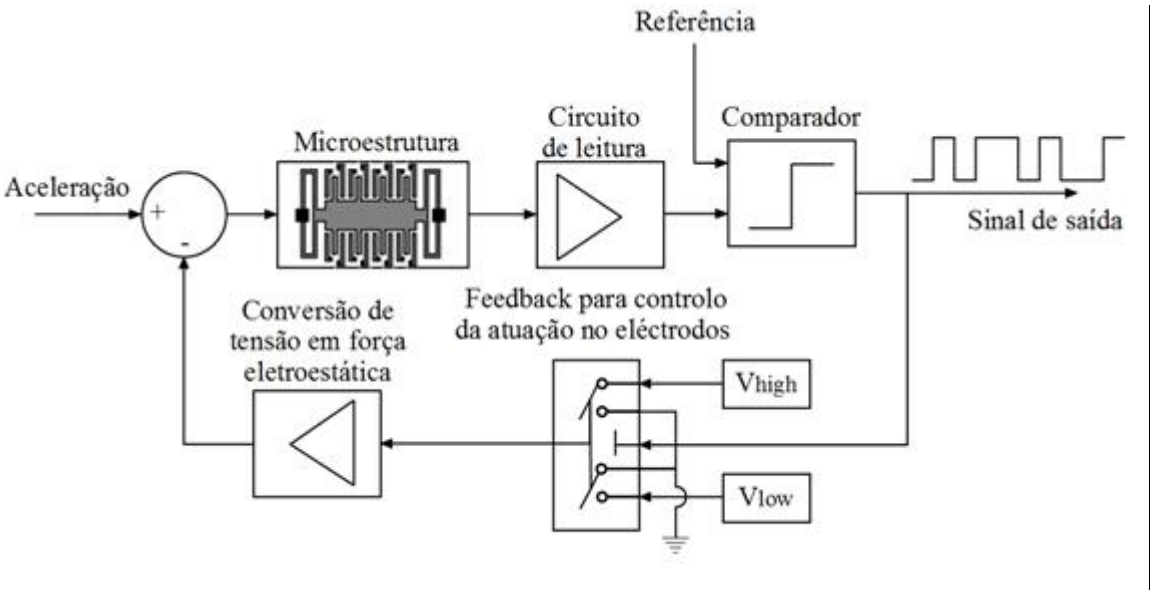


Figura 4.24 - Diagrama de blocos do acelerómetro baseado em modulação sigma delta

5. Implementação do Sistema

Neste capítulo é abordada a geração dos ficheiros essenciais para o processo de *boot* da ZedBoard. Será também apresentada toda a implementação desenvolvida nesta dissertação que vai desde o desenvolvimento de *hardware* e de módulos de *gateware* à programação de software.

5.1. *Geração de Ficheiros para a ZedBoard*

Antes de realizar qualquer tipo de implementação, primeiro foi necessário gerar os ficheiros correspondentes à distribuição Linux para a ZedBoard. Só posteriormente é possivel a interação entre uma aplicação no *user space* com o *kernel space* e com o *hardware* existente na Zynq-7000 SoC. O resultado final da geração dos ficheiros são 4 ficheiros: o **boot.bin**, **zImage**, **devicetree.dtb** e **ramdisk8M.image.gz**. Estes estão localizados dentro de um SD-Card de modo a que a ZedBoard faça o *boot* do sistema.

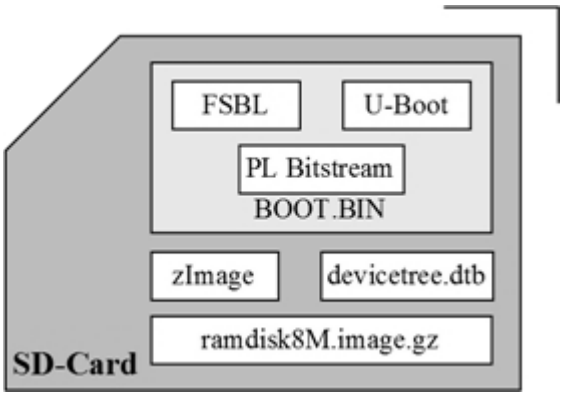


Figura 5.1 – Ficheiros de boot ZedBoard

A Figura 5.1 ilustra o resultado final após a geração de todos os ficheiros associados à inicialização do Linux na Zedboard [28]. Em seguida são demonstrados os processos para a geração dos ficheiros acima ilustrados.

5.1.1. Design Flow

Para o processo de compilação do U-Boot foi necessário recorrer a um *cross compiler* desenvolvido pela Sourcery Codebench e fornecida pela Xilinx. Depois do *download* dos ficheiros relativos ao u-boot, foram implementadas as respetivas configurações e posterior compilação. O processo encontra-se ilustrado na Figura 5.2.

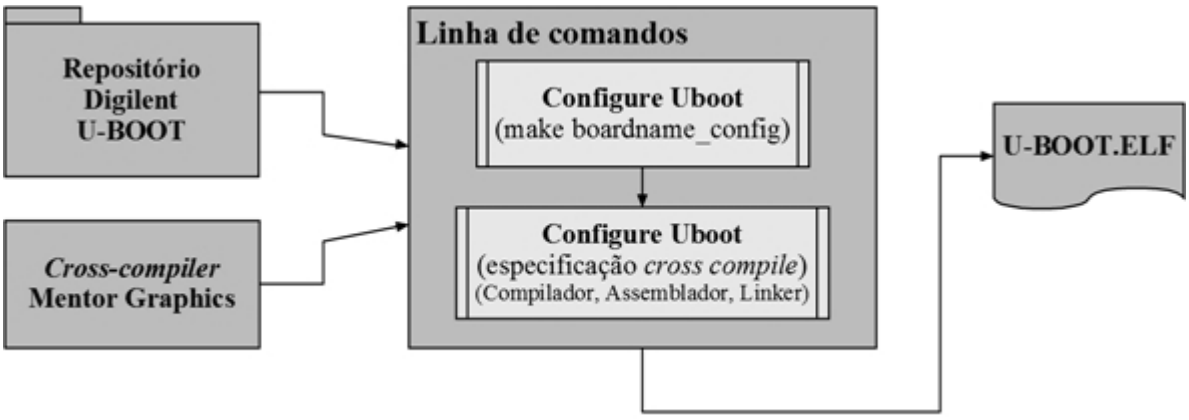


Figura 5.2 - Processo de geração U-Boot

Após a geração do *Second Stage Boot Loader* segue-se a geração do *Boot Image*. A Figura 5.3 mostra de forma genérica os principais aspetos a ter em consideração na construção da imagem de *boot*. Os passos que se apresentam a seguir são uma descrição mais detalhada.

1. Para o processo de *design* e de implementação utiliza-se a ferramenta PlanAhead, sendo esta a ferramenta central desde o início até à geração do *bitstream*. Depois disso o *hardware* é exportado para o SDK onde o desenvolvimento de aplicações de desenrolam.
2. É incluído o ARM Cortex A9 *Processing System* no projeto.
3. XPS é executado, a partir do PlanAhead, para configuração do PS e para adição de periféricos do *Programmable Logic*.

4. O *Processing System* 7 *wrapper* (PS7) instancia a secção do *processign system* da Zynq-7000 para a *Programmble Logic* e para a lógica externa na plataforma.
5. Configurações do PS7 para configuração de modo a selecionar a plataforma como sendo a ZedBoard, adicionar periféricos I/O, configurar memórias, frequências de relógio, etc.
6. Criar ou adicionar IP *Cores*. Quando finalizado o XPS é fechado e retorna-se ao PlanAhead. XPS cria nessa altura os ficheiros ps7_init.tcl, ps7_init.c, ps7_init.h, que contêm as configurações mínimas dos periféricos do PS, como os *clocks* e os controlos de memória. Cria também um ficheiro XML que contem a instanciação do processador e respetivos periféricos, e os endereços para a geração de FSBL e BSP, e um ficheiro MHS, que é uma *netlist* relativa ao subsistema *de hardware* que define o sistema embebido.
7. No PlanAhead, um *top-level* HDL tem de ser gerado.
8. Se o *design* necessitar de *constrainsts*, tipicamente usado para assegurar que os sinais são corretamente mapeados, um ficheiro *.ucf tem de ser criado ou importado para o PlanAhead. No caso do sistema implementado este ficheiro foi criado de modo a mapear os sinais externos.
9. Segue-se a geração do *Bitstream* para configurar a logica do PL de acordo com os periféricos implementados. Neste estado, o *hardware* é definido no ficheiro system.xml, e um **system.bit** *bitstream* é gerado.
10. Neste estado a parte relativa ao *hardware* já foi configurada e gerada. Agora é exportado para o SDK para criar a parte de *software.*
11. Para o desenvolvimento de um sistema operativo, uma aplicação *First Stage Boot Loader* (FSBL) tem de ser criada.
12. O FSBL gera um ficheiro executável, **FSBL.elf**, compilado de acordo com a arquitetura ARM Zynq-7000.
13. A combinação do FSBL, do U-Boot anteriormente abordado e do *bitstream* resulta no *Boot Image* (**boot.bin**).

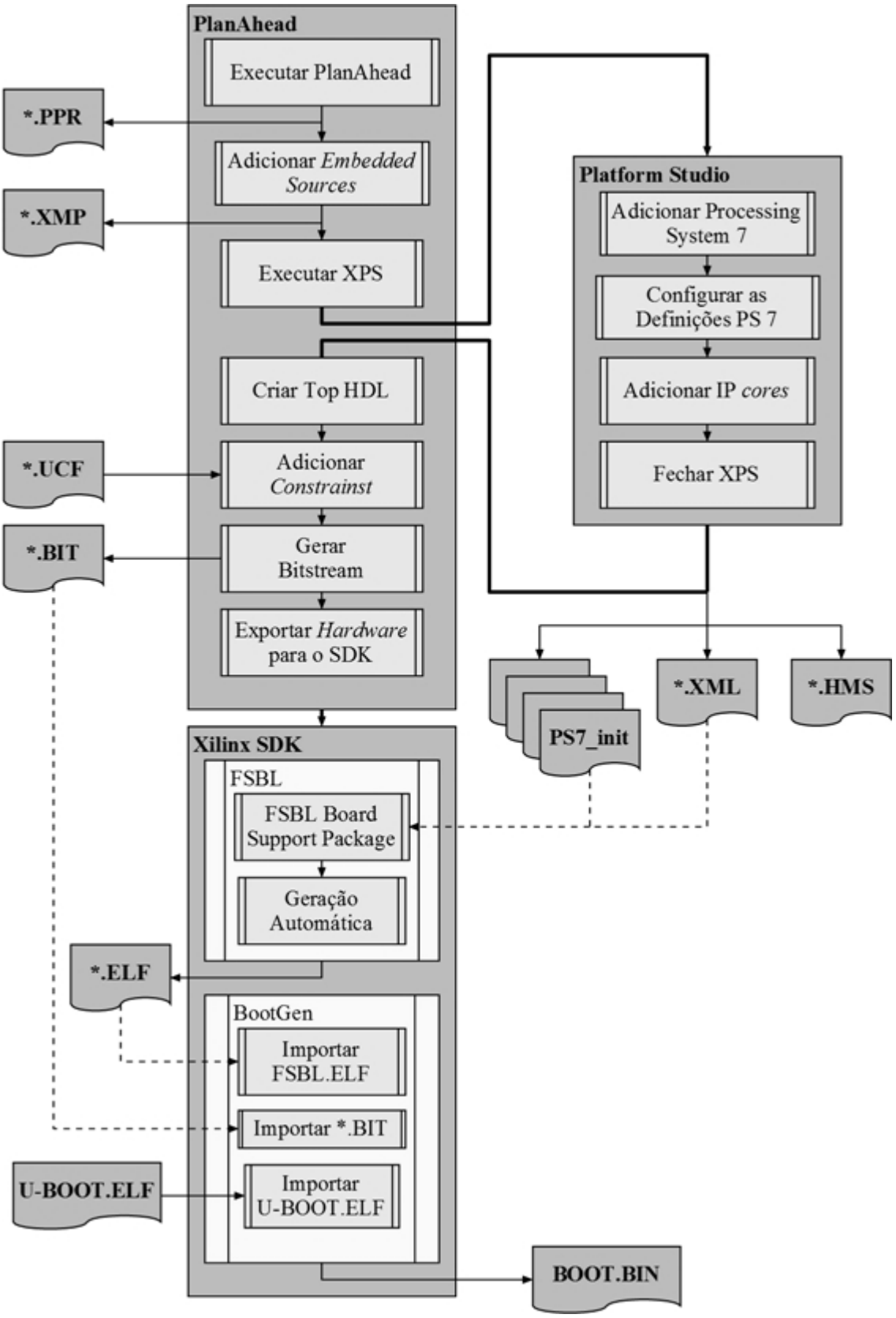


Figura 5.3 - Processo de geração BOOT.BIN

A distribuição Linux implementada no decorrer desta dissertação foi a disponibilizada pela Digilent [29]. Após *download* dos respetivos ficheiros, é especificada a plataforma a que

diz respeito. Posteriormente, se necessário, é configurado o Kernel, especificando-se os periféricos de *software*, e por último é dado espaço à geração e compressão de ficheiros dando origem ao ficheiro **zImage**.

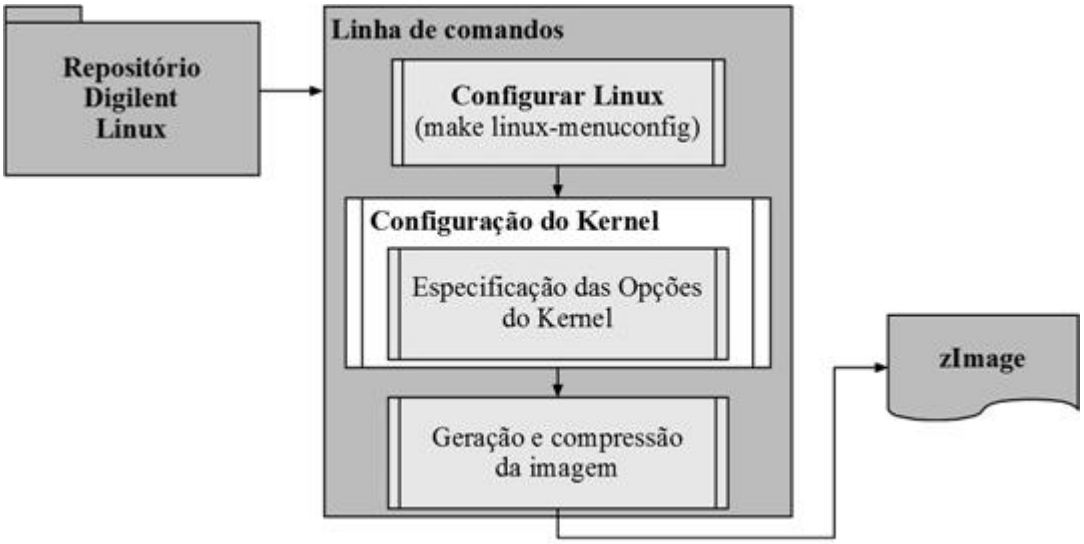


Figura 5.4 - Processo geração zImage

O próximo ficheiro a gerar é o **devicetree.dtb**. Aquando da configuração do sistema de *software* no SDK, é executada a ferramenta *device tree generator*, cujo propósito é a produção de uma fonte onde está contida a informação do design de *hardware*. É assim obtido um ficheiro *.dts que posteriormente é compilado de modo a gerar o ficheiro *.dtb pretendido para a inclusão no cartão SD. A Figura 5.5 ilustra de uma forma abstrata o processo descrito.

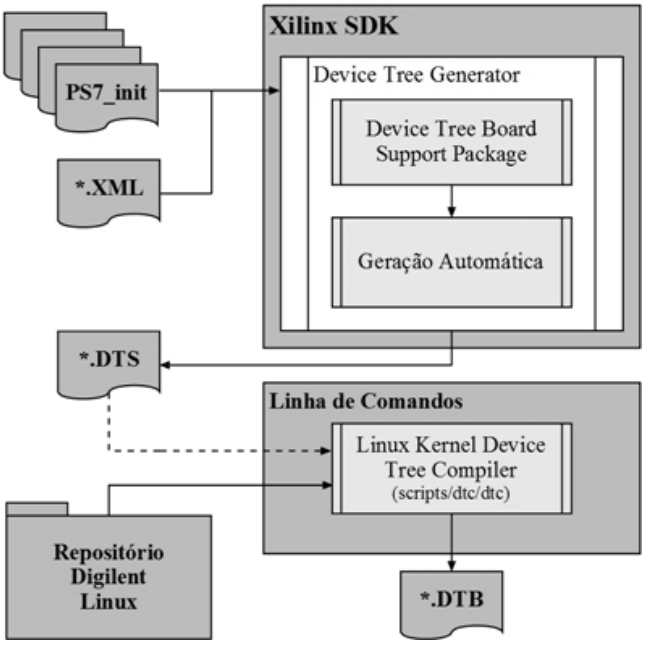


Figura 5.5 - Processo geração devicetree.dtb

Por fim resta o buildroot *file system*. Na página oficial de venda da Zedboard, [30], estão disponíveis diversos documentos de suporte à plataforma entre os quais está disponível o *buildroot file system* denominado de **ramdisk8M.image**. Como não se pretende alterar os ficheiros pré-definidos do sistema então não é necessária a sua criação e compilação, sendo assim utilizado o ficheiro disponibilizado pela Digilent.

5.2. *Hardware Desenvolvido*

Na implementação deste trabalho todo o *hardware* relativo aos sistemas de leitura e de atuação estava já implementado [9] e pronto a ser utilizado pela plataforma a desenvolver. Assim a parte de *hardware* que foi necessário desenvolver refere-se apenas à ligação dos conectores da ZedBoard às placas de aquisição e de leitura assim como estabelecer duas comunicações porta série para posterior comunicação com terminais externos.

. A Figura 5.6 e a Figura 5.7 ilustram as respetivas placas utilizadas.

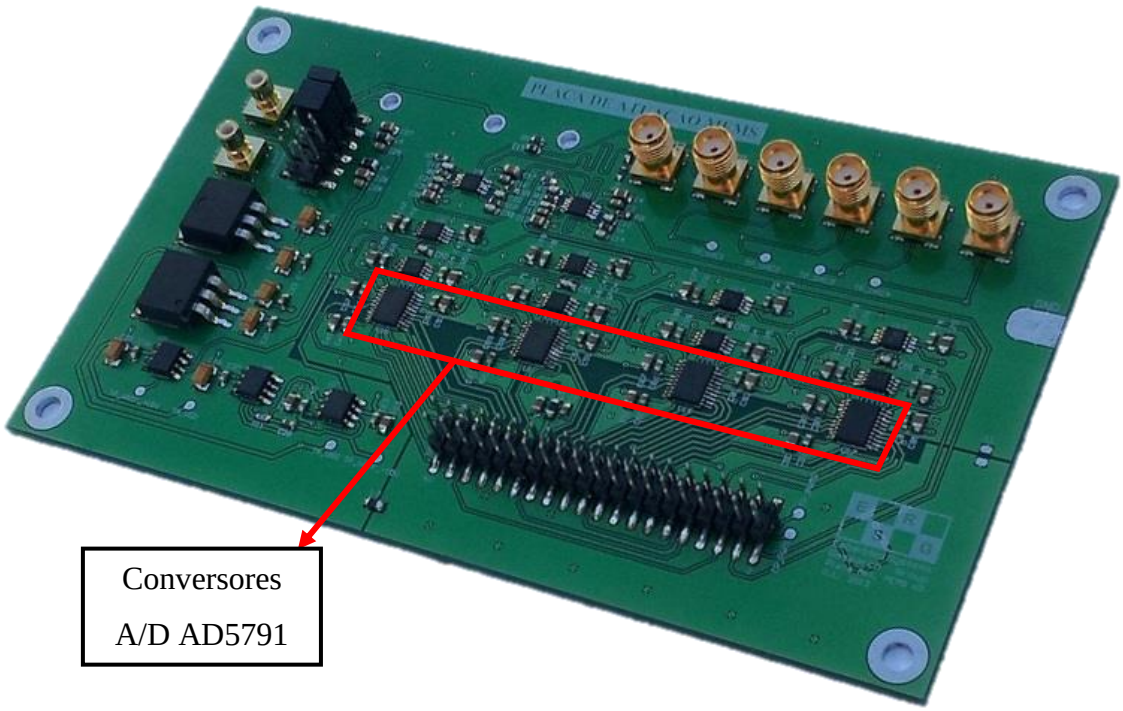


Figura 5.6 - Placa de circuito impresso do sistema de atuação

Nesta placa de atuação estão inseridos 4 DAC's AD5791 e 2 Switch's digitais ADG1433, componentes apresentados na categoria sistema de atuação no capítulo 4. Para

além disso esta placa possui duas entradas correspondentes a duas ondas PWM que depois de filtradas pelo filtro ativo são usadas como portadoras do circuito de *readout*.

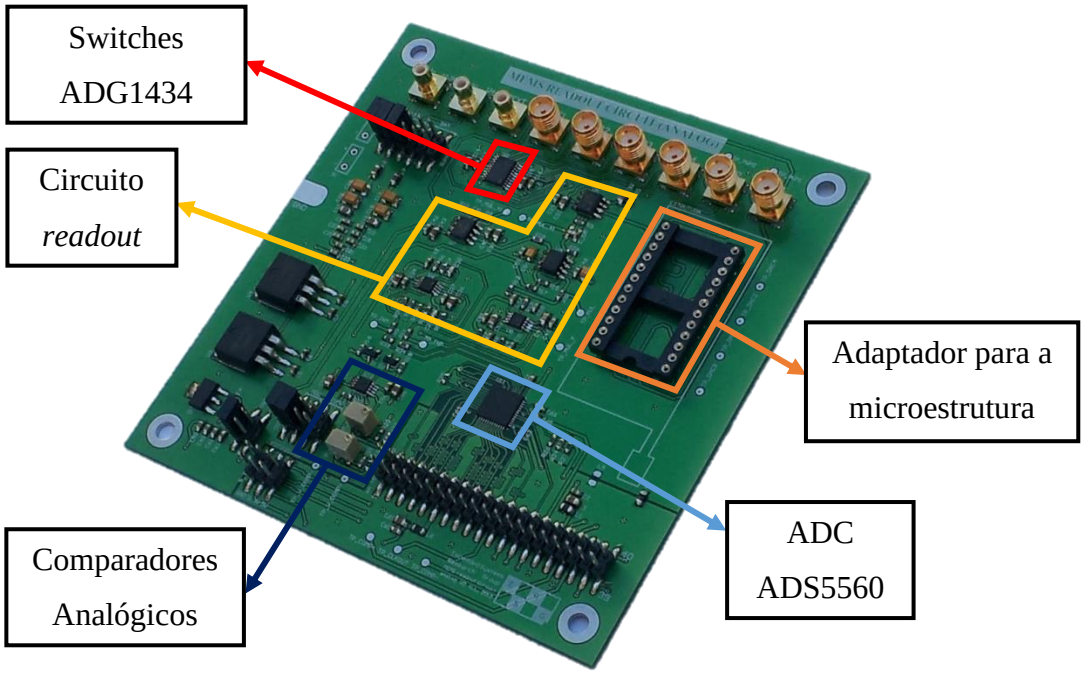


Figura 5.7 - Placa de circuito impresso do sistema de leitura

Relativamente à placa de aquisição, os sinais a ter em conta são os provenientes do ADS5560 e dos comparadores analógicos presentes neste circuito. A seleção do dispositivo de aquisição, conversor ou comparadores, é feito fisicamente através de 2 *jumpers* existentes na placa.

Relativamente à placa desenvolvida para interação com a ZedBoard, na distribuição dos pinos de ligação à placa de aquisição e à de atuação foi tido em conta a distribuição dos diferentes sinais nos respetivos conectores. A ligação do conector externo, onde se encontram os pinos provenientes das placas externas, aos conectores FMC existentes na ZedBoard foi determinada pela facilidade de construção física dessas ligações. Apenas o sinal de *clock* proveniente do ADC (*clock* de aquisição) teve que ter um critério especial. Sendo um *clock* externo deve ser conectado à FPGA aos pares diferenciais de pinos de *clock* que são denominados de *clock-capable* (CC) *inputs*. Estes pinos fornecem acesso dedicado de alta velocidade aos recursos de *clock* existente no dispositivo. Assim, e como as entradas *Clock-capable* recorrem a um *routing* dedicado, é necessário que sejam utilizados quando se trata de uma entrada de *clock* externo de modo a garantir os tempos do mesmo. Existem 4 pares de entradas CC (8 pinos no total) na ZedBoard. Sendo apenas necessário um *clock* externo de entrada, a entrada foi configurada como *single-ended clock*, o que implica que deve ser

conectada ao lado P (*master*) do par de pinos CC, podendo o lado N (*slave*) ser utilizado apenas como *user* I/O [20].

Posto isto restou fazer a distribuição de todos os sinais e alimentações à saída do conector FMC pertencente à ZedBoard, e que se encontram descritos na Tabela 5-1, na Tabela 5-2 e na Tabela 5-3.

Tabela 5-1 - Descrição pinos de entrada e saída do FMC - JA1

FMC – J1A					
Header Pin	**ZedBoard *Pin***	**Sinal**	***Header Pin***	**ZedBoard *Pin***	**Sinal**
H1-5	LA06_P	iDATA_ADC[2]	H1-20	LA09_N	iDATA_ADC[14]
H1-7	LA06_N	iDATA_ADC[4]	H1-22	LA13_P	iDATA_ADC[15]
H1-8	LA01_P_CC	iCLK_ADC	H1-26	LA13_N	iDATA_ADC[5]
H1-9	LA10_P	iDATA_ADC[6]	H1-28	LA17_P_CC	oCLK_ADC
H1-10	LA01_N_CC	iDATA_ADC[7]	H1-29	3V3	D3.3V
H1-12	GND	DGND	H1-30	GND	DGND
H1-13	LA10_N	iDATA_ADC[8]	H1-32	LA17_N_CC	iCOM[0]
H1-14	LA05_P	iDATA_ADC[9]	H1-34	LA23_P	iCOM[1]
H1-15	LA14_P	iDATA_ADC[10]	H1-37	LA27_N	DS1
H1-16	LA05_N	iDATA_ADC[11]	H1-38	LA23_N	DS2
H1-17	LA14_N	iDATA_ADC[12]	H1-39	LA26_N	DS4
H1-18	LA09_P	iDATA_ADC[13]	H1-40	LA26_P	DS3

Tabela 5-2 - Descrição pinos de entrada e saída do FMC - JA2

FMC – J1B					
Header Pin	**ZedBoard *Pin***	**Sinal**	***Header Pin***	**ZedBoard *Pin***	**Sinal**
H1-2	CLK1-M2C_P	iDATA_ADC[0]	H2-23	LA20_N	oLDAC [2]
H1-4	CLK1-M2C_N	iDATA_ADC[1]	H2-24	LA15_P	oLDAC[2]
H1-6	LA00_P_CC	iDATA_ADC[3]	H2-25	LA22_P	oCLEAR [2]
H2-2	CLK1-M2C_P	PWM_0	H2-26	LA15_N	oRESET[2]
H2-4	CLK1-M2C_N	PWM_180	H2-27	LA22_N	oSS [1]

H2-6	LA02_P	oSCLK[0]	H2-28	LA19_P	oMOSI[1]
H2-7	LA08_P	oSCLK[3]	H2-29	3V3	D3.3V
H2-9	LA08_N	oSCLK[2]	H2_30	GND	DGND
H2-10	LA04_P	oSCLK[1]	H2-31	LA25_P	oLDAC [1]
H2-12	GND	DGND	H2-32	LA19_N	oLDAC[1]
H2-13	LA12_P	oSS[3]	H2-33	LA25_N	oCLEAR [1]
H2-14	LA04_N	oMOSI[3]	H2-34	LA21_P	oRESET[1]
H2-15	LA12_N	oLDAC [3]	H2-35	LA29_P	oSS [0]
H2-16	LA07_P	oLDAC[3]	H2-36	LA21_N	oMOSI[0]
H2-17	LA16_P	oCLEAR[3]	H2-37	LA29_N	oLDAC[0]
H2-18	LA07_N	oRESET[3]	H2-38	LA24_P	oLDAC[0]
H2-20	LA11_P	oSS[2]	H2-39	LA31_P	oCLEAR [0]
H2-22	LA11_N	oMOSI[2]	H2-40	LA24_N	oRESET[0]

Tabela 5-3 - Descrição pinos de entrada e saída das comunicações UART

UART		
ZedBoard *Pin*	**Sinal**	**Dispositivo FT232R**
LA30_P	iRX[0]	1
LA30_N	iTX[0]	1
LA32_P	iRX[1]	2
LA32_N	iTX[1]	2

Como se pode observar na Tabela 5-3, definiu-se a implementação de dois canais de UART distintos. Como resultado, e tendo em conta os vários aspetos abordados anteriormente, a Figura 5.8 apresenta o circuito impresso da placa desenvolvida. O esquemático (Figura A.1) e o *layout* respetivo (Figura B.1 e Figura B.2) estão disponíveis nos anexos A e B.

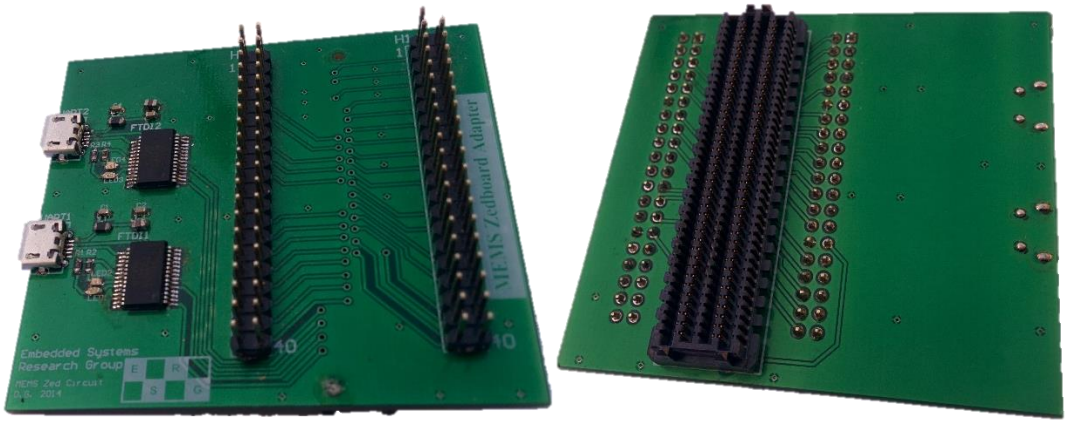


Figura 5.8 - Placa de circuito impresso do *hardware* desenvolvido

5.3. *Gateware Desenvolvido*

Neste subcapítulo são apresentados os módulos implementados no PL, necessários à ligação entre a ZedBoard e as placas de desenvolvimento apresentadas neste capítulo. Os módulos foram implementados utilizando a linguagem Verilog HDL e com recurso à ferramenta PlanAhead.

5.3.1. Módulo *Clock Generation*

Antes de implementar os módulos de *gateware* na *Programming Logic* é necessário garantir que todos os *clocks* relativos a esses módulos sejam gerados. Na tabela que se segue explicita-se os módulos e os respetivos *clocks*.

Tabela 5-4 - *Clocks* módulos PL

Módulo *Gateware*	***Clocks***
PWM	*250MHz*
DAC_GPIO	*100MHz, 25MHz*
SPI	*25MHz*
ADC_DMA	*100MHz, 50MHz, 25MHz, 5MHz, 1MHz, 500KHz, 5KHz*
UART_GPIO	*50MHz*
RESET	*250MHz)*

A necessidade destes *clocks* prende-se com a frequência de *clock* associada aos dispositivos com a qual os próprios módulos interagem sendo que no caso do módulo de *ADC_DMA* se prende também com a frequência de aquisição desejada. Foi definido que o *clock* geral do sistema seria de 100MHz, derivado das limitações de velocidade dos diversos IPs utilizados nesta dissertação, sendo que esta frequência é suficiente para satisfazer as necessidades propostas.

A Zynq-7000 SoC possui quatro PLLs independentes, configuráveis no PS, e que podem ser exportadas para o PL. Foram configurados os seguintes *clocks*: 250MHz, 100MHz, 50MHz e 5MHz. Foi criado um módulo, que através do recurso a algumas PLLs, gere os restantes *clocks* em défice. Neste módulo é também definido, através de uma linha informação proveniente do *user space*, a frequência de aquisição por parte do ADC e a frequência com que são transmitidos os dados para o *user space*.

```
module clk_gen(
    input iCLK_50M,
    input iCLK_5M,
    input iENABLE,      // Enable geral do sistema
    input iRST,         // Reset geral do sistema
    input [2:0] iCONFIG_ADC, // 0->IDLE, 1->50MHz, 2->25MHz, 3->5MHz,
4->1MHz, 5->500KHz
    input [1:0] iCONFIG_DMA, // 1->Envio a 5MHz, 2-> Envio a 5KHz
    output reg orCLK_25M,
    output oCLK_ADC,
    output oCLK_DMA
);
```

Figura 5.9 - Registos de entrada e saída do módulo gerador de *clocks*

Internamente neste módulo existem diversos contadores, que contam os ciclos de relógio necessários para gerar as transições dos *clocks* em falta (25MHz, 1MHz, 500KHz, 5KHz).

5.3.2. Módulo *Reset*

Neste módulo é implementado um método que habilita todos os *resets* de forma assíncrona mas que os desabilita de forma síncrona, prevenindo assim o aparecimento de *bugs* uma vez que existem diversos *clocks* no sistema com diferentes frequências. Na Figura 5.10 encontra-se ilustrado um exemplo da implementação usada. Neste exemplo é imposto ao sinal um ciclo de relógio de atraso para acontecer o sincronismo.

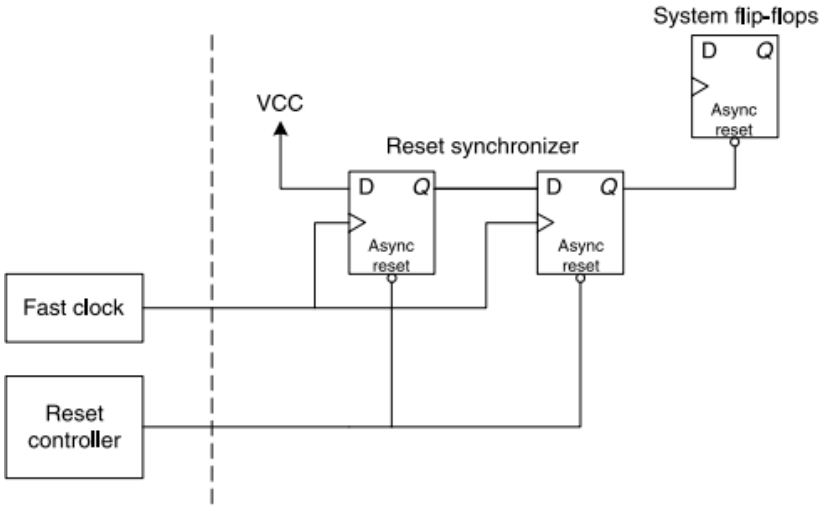


Figura 5.10 - Esquema do circuito de reset [19]

O *clock* usado para a implementação deste módulo será o de maior frequência, isto é 250MHz. Sendo o *clock* de menor frequência correspondente a 5MHz, foi necessário criar uma cascata de 50 flip-flops de modo a atrasar o sinal. Este atraso faz com que o sinal de *reset* seja habilitado no momento exato em que todos os *clocks* se encontram sincronizados. Os registos de *reset* são habilitados assincronamente via sinal externo e desabilitados após 50 ciclos do relógio de maior frequência. Na Figura 5.11 apresenta-se o código relativo a este módulo.

```
module reset_sync(
    input iCLK_250M,    // Reset de maior frequência
    input iRST,         // Reset sinal externo
    output reg orRST_SYST   // Reset geral do sistema
);

reg [50:0] rRESET_AUX;

always @(posedge iCLK_250M)
begin
    if(!iRST)   //Caso de reset
    begin
        orRST_SYST<=0;      //RESET dos Flip-flops
        rRESET_AUX<=51'd0
    end
    else
    begin
        rRESET_AUX[0]<=1;   // Passagem de estado entre os Flip-flops
        rRESET_AUX[1]<=rRESET_AUX[0];
                    (...)
        rRESET_AUX[50]<=rRESET_AUX[49];
        orRST_SYST<=rRESET_AUX[50];
    end
end
endmodule
```

Figura 5.11 - Registos e implementação do módulo *reset* [19]

5.3.3. Módulo UART

Neste módulo optou-se unicamente pelo sentido de comunicação da FPGA para o terminal. Deste modo a função do módulo UART faz o envio de um valor proveniente do *software*. A informação relativa à UART é proveniente de um módulo IP GPIO. Num canal é disponibilizado o dado a enviar, juntamente com um bit de controlo (iDATA_UART [8]) para a transmissão, em que é dada a autorização de envio sempre que ocorrer uma comutação de estado. A máquina de estado apresentada na Figura 5.12 demonstra o funcionamento deste módulo.

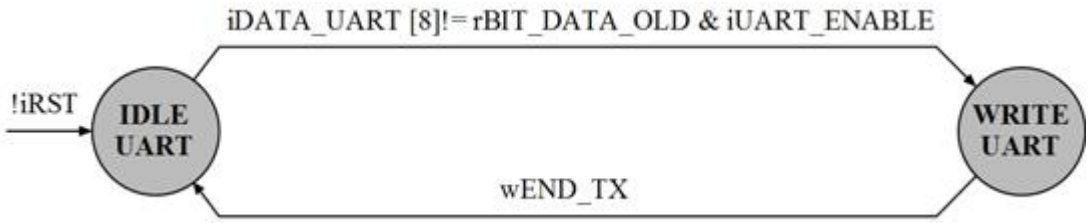


Figura 5.12 - Máquina de estados UART

Relativamente ao módulo responsável pela configuração dos bits de transmissão e de receção, "Tx" e "Rx" respetivamente, já se encontrava desenvolvido, implementado e validado em [31] e foi usado neste trabalho.

Por fim na Figura 5.13 são ilustrados os registos de entrada e saída do módulo anteriormente descrito.

```
module uart_write(
    input iCLK_50M,      // CLOCK UART 50MHz
    input iRST,          // RESET geral do sistema
    input iENABLE,       // ENABLE geral do sistema
    input iRX,           // Bit recepção
    input iUART_ENABLE   // Bit selecção
    input [8:0] iDATA_UART, // Bit de envio + Byte a enviar
    output oTX              // Bit transmissão
    );
```

Figura 5.13 - Registos e implementação do módulo uart_write

5.3.4. Módulo GPIO DAC

A este módulo está incumbida a tarefa de configurar o DAC de acordo com o nível de atuação pretendido. Assim, sempre que um novo valor de atuação é definido, automaticamente o DAC terá de ser reconfigurado com o novo valor pretendido. O *clock* do sistema foi configurado para 100MHz. Quanto ao *clock* usado para a transmissão de dados para o DAC foi de 25MHz, uma vez que é o *clock* recomendado [13], resultando numa frequência máxima de 1,04MHz para alteração do nível de atuação.

Tal como definido no 0, a informação relativa à atuação será transmitida através do IP AXI GPIO. Uma das características inerentes a esse IP é a permanência do nível de estado dos bits associados o que viabiliza a metodologia da leitura constante dos bits e posterior comparação com o estado anterior. Caso sejam diferentes, um novo valor de atuação terá de ser atualizado no DAC através do recurso ao módulo SPI. O módulo SPI desenvolvido e testado em [31] foi usado neste trabalho, e portanto não foi necessário desenvolvê-lo novamente. Quanto à implementação do módulo do DAC, este é constituído por duas máquinas de estados, uma relativa à leitura do GPIO e outra relativa à comunicação SPI. Na Figura 5.14 e na Figura 5.15 estão ilustradas as respetivas máquinas de estado.

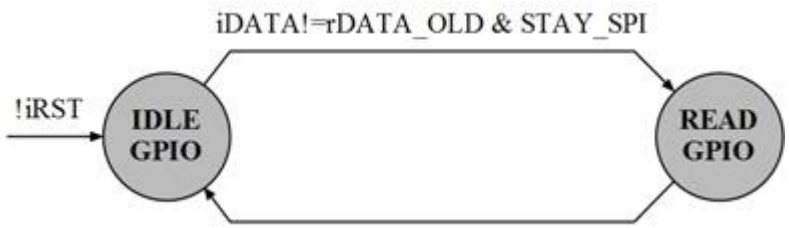


Figura 5.14 - Máquina de estados DAC_GPIO

Como já abordado anteriormente e como se pode verificar na Figura 5.14, o módulo espera a chegada de um novo valor de atuação, onde também verifica se o módulo SPI se encontra ocupado. Quando é verificada essa situação, o estado do DAC passa a "READ" voltando ao estado inicial no ciclo de *clock* seguinte.

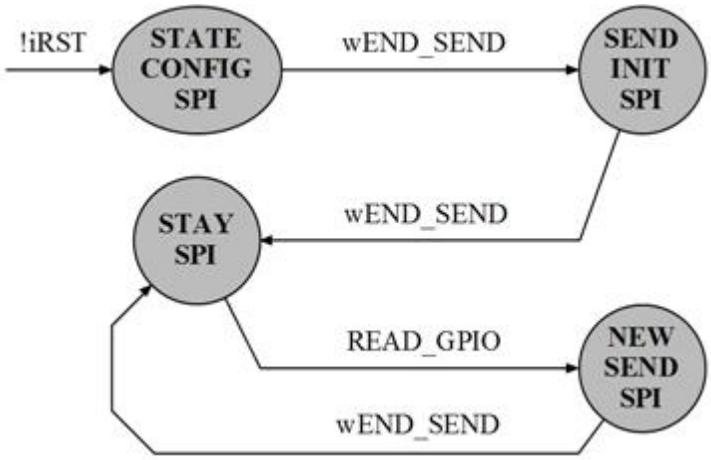


Figura 5.15 - Máquina de estado módulo SPI

Quanto à máquina de estados relativa ao módulo SPI esta possui vários estados. Inicialmente há uma configuração necessária, uma vez que o conversor possui vários modos de operação. Após concluída essa configuração, é estabelecido um valor de atuação correspondente à tensão de 0V, de modo a evitar atuações indesejadas. Uma vez no estado "STAY" é esperado que haja uma variação do valor a enviar para o DAC. Sempre que haja um novo valor este é atualizado voltando posteriormente para o estado "STAY" quando finalizada a transição. Depois de definido o comportamento do módulo, resta definir os registos de entrada e saída.

```
module gpio_dac(
    input iCLK_100M,    // Clock geral sistema
    input iCLK_SPI,     // Clock SPI 25MHz
    input iRST,         // Reset geral do sistema
    input iENABLE,      // Enable geral do sistema
    input [19:0] iDATA, // Valor de actuação
    input iDAC_ENABLE,  // Enable DAC
    // Sinais SPI
    output oSCLK,
    output oSS,
    output oMOSI,
    output oLDAC,
    output oRESET,
    output oCLEAR
);
```

Figura 5.16 - Registos de entrada e saída do módulo GPIO DAC

Os registos de *output* relacionados com o SPI vêm diretamente do módulo SPI implementado [31]. Foram instanciados quatros módulos GPIO DAC, cada um para cada dispositivo DAC disponível na placa de atuação ilustrada na Figura 5.6. O registo

iDAC_ENABLE, que está conectado a um dos quatros registos GPIO de seleção dos DACs, define que módulo se encontra selecionado e portanto é deste modo definido o conversor digital para analógico onde irá ser escrito um novo valor de atuação.

5.3.5. Módulos do sistema de leitura

Existem múltiplas opções de aquisição que podem ser habilitadas pelo utilizador. Assim, antes de iniciar qualquer tipo de aquisição de sinal terão que ser definidos todos os parâmetros respetivos. Essa informação será definida por *software* e será partilhada com o PL através do recurso a um IP AXI GPIO. A Figura 5.17 ilustra a distribuição dos registos do axi GPIO relativamente às diferentes opções disponíveis. A Tabela 5-5 descreve esses registos.

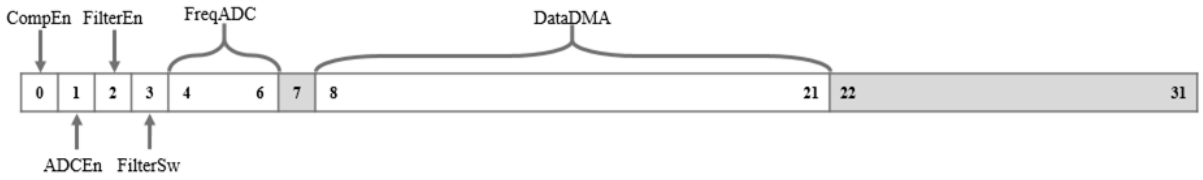


Figura 5.17 - Registos AXI GPIO Configuração

Tabela 5-5 - Descrição dos registos AXI GPIO Configuração

Bits	Nome do campo	Valor pré-definidos	Tipo de acesso	Descrição
0	*CompEn*	*0x0*	*R/W*	*Habilita/Desabilita a aquisição por comparadores*
1	*ADCEn*	*0x0*	*R/W*	*Habilita/Desabilita a aquisição pelo ADC*
2	*FilterEn*	*0x0*	*R/W*	*Habilita/Desabilita a utilização do filtro digital*
3	*FilterSw*	*0x0*	*R/W*	*Sinal para alteração dos valores dos coeficientes. Deteção em transição positiva*
4 a 6	*FreqADC*	*0x0*	*R/W*	*Especificação da frequência de aquisição*
8 a 21	*DataDMA*	*0x0*	*R/W*	*Especificação do número de words de cada pacote do DMA*

5.3.5.1. Módulo Filtro Digital FIR

Tal como descrito na fase de *design,* o filtro em questão é um filtro digital FIR passa-baixo que recorre ao método da janela de Kaiser. A primeira necessidade adjacente à implementação do filtro pretende-se com a disponibilidade dos valores relativos aos coeficientes. Como já definido nesta dissertação estes valores serão transferidos do PS para o PL através do IP AXI BRAM que por sua vez serão gerados em ambiente Linux através de uma aplicação de *software* que será abordada mais a frente.

Ao contrário do filtro implementado em [31], em que a Altera possui um módulo IP que realiza um determinado número de multiplicações e que posteriormente as soma, a Xilinx não disponibiliza nenhum IP com essas características tendo assim que se realizar todas as operações individualmente.

Uma vez que o utilizador pode determinar a gama de frequências com que pretende filtrar o sinal adquirido, definiu-se que o limite máximo de portas lógicas disponíveis na Zynq-7000, para a implementação dos IPs de multiplicação e de adição, determinaria a ordem do filtro. Após várias implementações, determinou-se um número de coeficientes máximo de 54.

Para a multiplicação instanciaram-se 54 módulos IP *multiplier* [32], no modo combinacional, em que um parâmetro de entrada é o sinal de entrada desfasado no tempo correspondente e o outro parâmetro é o respetivo coeficiente. Quanto à adição, uma vez que o módulo IP disponível [33] realiza a adição de apenas dois parâmetros, foi necessário recorrer ao uso de uma árvore lógica de modo a somar os 54 parâmetros. Surgiu assim a ideia de implementar sub-módulos que funcionam de modo hierárquico. Inicialmente é criado um módulo de adição múltipla de 4 valores. Em seguida é criado um módulo que através da instanciação do módulo 4 ADDER adiciona 10 elementos em simultâneo. Na Figura 5.18 pode-se observar a implementação aqui descrita.

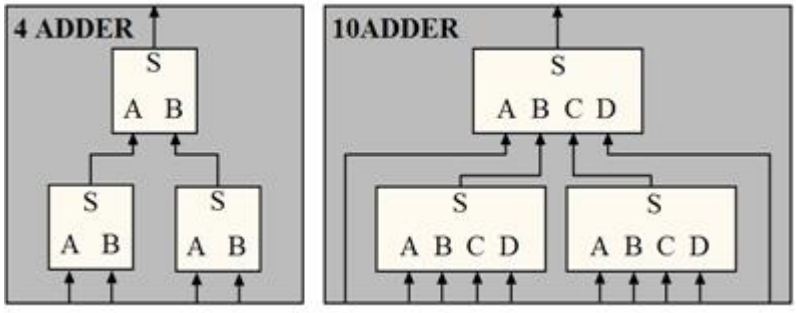


Figura 5.18 - Esquema dos módulos de adição de 4 valores e de 10 valores

Posteriormente, com a finalidade de somar todos os elementos, é criado um último módulo de adição. Neste são instanciados 6 módulos de adição de 10 elementos, dos quais 5 somam 50 elementos e o restante módulo soma os resultados das somas anterior com os 4 elementos restantes.

Por fim, é criado o módulo filtro FIR, onde são instanciados os vários módulos de multiplicação e o respetivo módulo de adição. A Figura 5.19 ilustra a máquina de estado relativa ao módulo FIR.

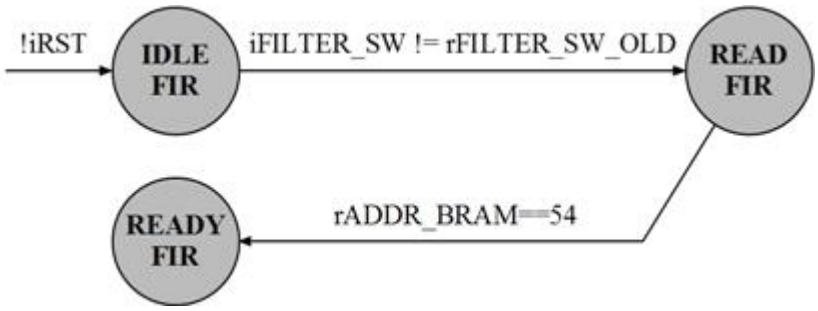


Figura 5.19 - Máquina de estado módulo FIR

A máquina de estado tem uma única função que é assegurar que os coeficientes são preenchidos antes de o filtro iniciar qualquer tipo de operação. Deste modo, o filtro encontra-se no estado IDLE e está à espera de uma transmissão positiva do sinal FilterSw (ver Figura 5.17). Acontecendo essa transição é inicializado o processo de leitura da BRAM. Neste estado, a cada ciclo de *clock* é incrementado o endereço da BRAM e realizada a respetiva leitura do valor. Como já especificado existe um ciclo de relógio de atraso neste tipo de leitura, porém sendo o endereçamento antes do processo de leitura inicializado a zero, a primeira leitura corresponderá ao primeiro valor da BRAM, e como é incrementado a todas as transições de relógio, os valores obtidos serão sempre os desejados, não havendo trocas de valores. Posteriormente e quando já se percorreram todos os valores da BRAM, isto é, a leitura dos 54 coeficientes, o estado passa para READY ficando então o filtro pronto a inicializar o processo de filtragem.

5.3.5.2. Módulo DMA ADC

Num nível acima ao módulo de filtro anteriormente apresentado é criado o módulo ADS5560. Neste módulo verifica-se se é ou não pretendida a filtragem dos dados provenientes do ADC. Essa informação é fornecida por um dos registos disponíveis no AXI GPIO de configuração da Figura 5.17.

Por último surge o módulo DMA ADC responsável pelo envio dos dados para o *software*. Neste módulo é atribuído, a um registo, o número de *words* a enviar pelo DMA, e posteriormente dá-se uma transferência periódica dos valores adquiridos, segundo uma frequência definida no módulo *clock generation*. A Figura 5.20 apresenta a máquina de estados relativa a este módulo

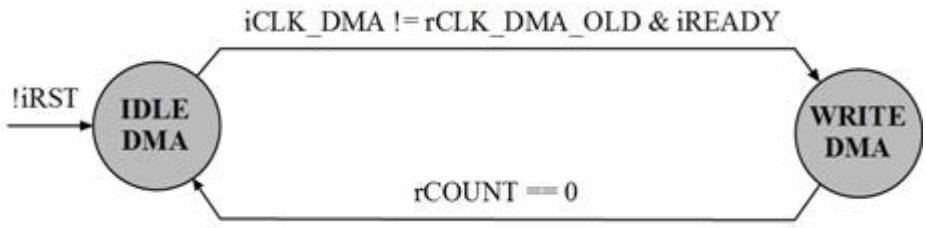


Figura 5.20 - Máquina de estados DMA ADC

Como se pode observar na Figura 5.20, um novo envio de dados ocorre quando o DMA se encontra disponível para inicializar transferência de dados e quando ocorre uma transição positiva do *clock* relativo à frequência de envio pretendida. Assim que o registo contador indique que já foram transmitidas todas as *words* então a transferência é dada como concluída e o registo contador é carregado novamente com o número de *words* definido para dar início a uma nova transmissão.

Em termos de armazenamento dos dados adquiridos podemos ter duas situações. A primeira em que o número de *words* a transferir é de apenas um e a segunda situação em que o número de *words* é superior a um. Relativamente à primeira situação não é necessário o armazenamento de valores visto que quando se iniciar a transferência, o valor de envio será o disponível no registo de aquisição respetivo. Quanto à segunda situação, será necessário a implementação de uma FIFO para armazenar os valores adquiridos. Foi implementado deste

modo um módulo IP FIFO da Xilinx [34] para o armazenamento. Este é configurado com dois *clocks* independentes, para a escrita e para a leitura, em que o *clock* de escrita possui a mesma frequência que a definida para a aquisição e o *clock* de leitura possui a mesma frequência que o *clock* do sistema, sendo a mesma incorporada no IP DMA. Definiu-se que a frequência de envio dos dados armazenados na FIFO é de 5KHz. Esta frequência garante a transferência de todos os dados para o *software*. O cenário em que é usada a frequência máxima de aquisição (50MHz) é aquele em que mais valores necessitam de ser armazenados. Tendo em conta a frequência de envio de 5KHz resulta num número de 10000 *words*. Assim no que diz respeito à profundidade da FIFO foi definida com 16383 *words*, sendo assim garantido que são armazenados todos os valores adquiridos.

Por fim segue-se a transferência dos dados para o DMA. O IP possui dois canais, um de leitura outro de escrita. Assim, e como a transferência de dados se dá no sentido PL para PS, são exportados para a FPGA os sinais relativos à escrita no IP sendo estes pertencentes ao canal M_AXI_MM2S. Na Tabela 5-6 encontram-se especificados os sinais exportados.

Tabela 5-6 - Descrição dos sinais AXI DMA canal de escrita

Nome do sinal	Interface	Tipo de Sinal	Descrição
s_axis_s2mm_tvalid	*S_AXIS_S2MM*	*I*	*Indica se o sinal de dados é valido*
s_axis_s2mm_tready	*S_AXIS_S2MM*	*O*	*Indica se o canal se encontra pronto para a receção de dados*
s_axis_s2mm_tlast	*S_AXIS_S2MM*	*I*	*Indica que é última word de dados*
s_axis_s2mm_tdata (31:0)	*S_AXIS_S2MM*	*I*	*Sinal de dados*

Após os sinais acima descritos estarem disponíveis no *hardware*, entram no módulo DMA ADC e aí serão explorados. A Figura 5.21 ilustra os tempos e a sequência de processos implementados na escrita de dados.

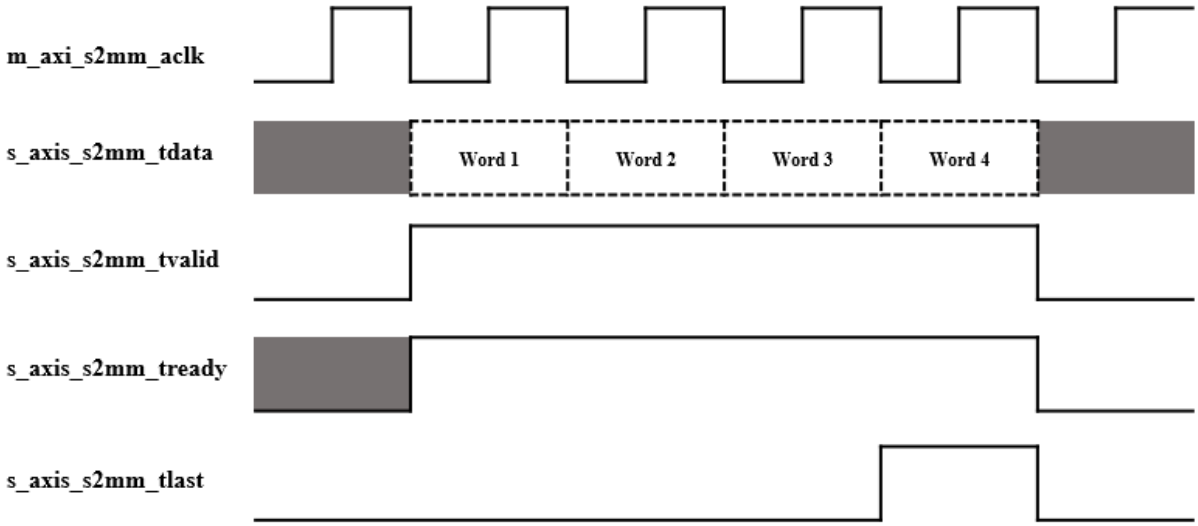


Figura 5.21 - Exemplo de sequência de escrita no canal

Inicialmente é verificado se o canal se encontra pronto para a receção de dados. Posteriormente é dada a indicação de valores validos e disponíveis no sinal de dados. Quando é transmitida a última *word* de dados é prontamente indicado no sinal de "last".

Finalizada a abordagem dos módulos e topologias definiram-se os registos de entrada e de saída relativos ao módulo DMA ADC que se encontram ilustrados na Figura 5.22.

```
module DMA_ADC(
    input iCLK_100M,     // Clock geral do sistema
    input iCLK_ADC_CFG, // Clock de aquisição
    input iCLK_DMA,      // Clock DMA
    input iRST,          // Reset geral do sistema
    input iENABLE,       // Enable geral do sistema
    input [15:0] iDATA_ADC,    // Dado do ADC
    input iCLK_ADC,            // Clock proveniente do ADC
    output oCLK_ADC            // Clock de saída para o ADC
    input iDMA_ENABLE,         // Enable DMA
    input iFILTER_ENABLE,      // Enable filtro
    input iFILTER_COEF,        // Sinal leitura coeficientes
    input [13:0] iCONFIG_DMA, // Número de words a transmitir DMA
    output [31:0] oADDR_BRAM, // Dado BRAM
    input [31:0] iDATA_BRAM,  // Endereço BRAM
    input iREADY,              // Sinal READY DMA
    output [31:0] oDATA,       // Dados DMA
    output reg orLAST,         // Sinal LAST DMA
```

Figura 5.22 - Registos de entrada e saída do módulo DMA ADC

5.4. *Software Desenvolvido*

Neste subcapítulo é abordada toda a implementação no que diz respeito à criação de *software*, essenciais no interface com os vários IPs. Foram desenvolvidas diversas classes c++, em que cada classe é responsável pela comunicação com um IP específico. Foi também desenvolvido um módulo responsável pela geração dos coeficientes do filtro implementado.

5.4.1. Geração dos coeficientes do Filtro FIR

A geração dos coeficientes relativo ao filtro digital prende-se com o cálculo dos coeficientes, através das fórmulas relativas ao método da janela de Kaiser, já abordadas nesta dissertação. Assim criou-se uma classe responsável por essa mesma tarefa. No construtor da classe são passados quatro parâmetros previamente definidos pelo utilizador, sendo estes as frequências de amostragem, de corte, da banda passante e por fim a amplitude. Posteriormente, é criada uma função pertencente à classe que retorna um apontador para o *array* de inteiros onde estão localizados os coeficientes calculados. Está também implementada uma função que retorna o número de coeficientes gerados. De salientar que o número de coeficientes limite implementado é de 54 e por isso, terá que haver por parte do utilizador um cuidado especial aquando a definição das frequências e amplitude respetivas. Na Figura 5.23 está ilustrado o código relativo à implementação da classe descrita anteriormente.

```
class Filter
{
private:
    double *WinCoeff, *WinCoeff_aux, *xi, *WinIdeal;
    int *WinCoeffFinal;
    double d_w, ws, wp;
    int M, n;

public:
    Filter(int fp, int fs, int fa, int A);
    ~Filter();
    int getCoeffLength() { return M+1; }
    int* CalcCoeff();
};
```

Figura 5.23 - Parâmetros da classe Filter

5.4.2. Interface BRAM

Foi instanciado um *block* BRAM *controller* IP no PS cuja função é traduzir as transações AXI para as portas de interface relativas ao bloco BRAM nativo. Assim todos os endereços de dados pertencentes ao bloco de memória são espelhados na memória DDR da plataforma. Para aceder à memória DDR usa-se a função *mmap()*. Foi criada uma classe em que o único método implementado é a escrita do dado pretendido no endereço de memória (offset de memória) especificado. Internamente, esse método soma o offset de memória recebido ao endereço base espelhado relativo à BRAM instanciada. É posteriormente escrito o dado no endereço resultante.

O objetivo da utilização da BRAM é a escrita em registos da FPGA dos coeficientes inerentes ao filtro FIR. Depois da geração dos coeficientes, é percorrido o vetor resultante e são transferidos os valores para o bloco de memória. A Figura 5.24 ilustra os parâmetros da classe relativa a esta classe.

```
class Bram_filter
{
private:
    int fd0;
    void* bram_filter_map;

public:
    Bram_filter();
    ~Bram_filter();
    void write(unsigned int addr, unsigned int wr);
};
```

Figura 5.24 - Parâmetros da classe Bram_filter

5.4.3. Interface GPIO

A interface do *software* relativa a este *core* pode ser realizada através do acesso direto ao espaço de memória onde estão concentrados os registos associados ao GPIO. O acesso direto à memória é feito através do recurso à função *mmap*() que mapeia um endereço físico num endereço virtual. A cada módulo instanciado é-lhe reservado um espaço de memória específico, podendo ser posteriormente acedido. Na Tabela 5-7 apresentam-se os registos relativos ao AXI GPIO e respetivos *offsets* em relação ao endereço de memória onde o módulo se encontra instanciado.

Tabela 5-7 - Registos AXI GPIO

Endereço de Memória (Offset)	Nome do Registo	Tipo de Acesso	Valor Default	Descrição
0x00	*GPIO_DATA*	*R/W*	*0x0*	*Canal 1 AXI GPIO Registo de Dados*
0x04	*GPIO_TRI*	*R/W*	*0x0*	*Canal 1 AXI GPIO 3-state Registo de Controlo*
0x08	*GPIO2_DATA*	*R/W*	*0x0*	*Canal 2 AXI GPIO Registo de Dados*
0x0C	*GPIO2_TRI*	*R/W*	*0x0*	*Canal 2 AXI GPIO 3-state Registo de Controlo*
0x0011C	*GIER*(1)	*R/W*	*0x0*	*Registo Enable de Interrupção Glocal*
0x0128	*IP IER*(1)	*R/W*	*0x0*	*IP Registo Enable de Interrupção*
0x120	*IP ISR*(1)	*R/TOW*(2)	*0x0*	*IP Registo de Estado da Interrupção*

Notas:
1. Registo de interrupção só estão disponíveis se o IP AXI GPIO tiver o parâmetro *Enable Interrupt* activo.
2. *Toggle-On-Write* (TOW) isto é, alterna o estado do bit quando é escrito o valor 1 no bit correspondente.

5.4.3.1. Registos de configuração e comparadores

O IP AXI GPIO possui dois canais separados. Deste modo instanciou-se um módulo IP GPIO no PS em que num canal são configurados dois registos correspondentes aos dois comparadores analógicos, e noutro canal são configurados os registos de configuração ilustrados na Figura 5.17.

```
class Gpio_comp
{
private:
    int fd0;
    void* gpio_map;

public:
    Gpio_comp();
    ~Gpio_comp();
    unsigned long read_comp(void);
    void write_config(unsigned long state);
};
```

Figura 5.25 - Parâmetros da classe Gpio_comp

Na Figura 5.25 encontram-se ilustrados os parâmetros da classe Gpio_comp. Essa classe possuí dois métodos “read_comp” e “write_config”. O primeiro método retorna o valor existente no registo de dados do canal 1. Na *main*, é feito *polling* ao valor retornado, sendo assim detetada a comutação de estado nos comparadores. No segundo método a *word* definida pelo utilizador, que é um parâmetro de entrada do método, é escrita no registo de dados relativo ao canal 2.

5.4.3.2. DAC

Foi criada uma classe que é responsável exclusivamente pelos conversores D/A existentes. Foi instanciado no PS um IP GPIO em que num canal se encontram os valores de atuação e no outro canal são definidos quatros registos relativos à habilitação de cada módulo correspondente instanciado na FPGA.

A Figura 5.26 ilustra os parâmetros relativos à classe criada.

```
class Gpio_dac
{
private:
    int fd0;
    void* gpio_map;

public:
    Gpio_dac();
    ~Gpio_dac();
    void write_dac( unsigned int wr);
    void sel_dac( unsigned int state);
};
```

Figura 5.26 - Parâmetros da classe Gpio_dac

5.4.3.3. UART

Como já previamente referido o IP GPIO relativo à UART implementa os dois canais, sendo que um tem definido 9-bit, 8-bit relativo ao byte a enviar e 1-bit controlo, e o outro tem definido 2-bit de habilitação dos respetivos módulos UART implementados na FPGA. Foram assim criados dois métodos pertencentes à classe em que cada método se encarrega de escrever no respetivo canal. No método de seleção da UART, o parâmetro de entrada é escrito no registo de dados do canal correspondente. Quanto ao método de transferência do dado o

parâmetro de entrada é o byte pretendido, sendo o bit de controlo definido dentro do próprio módulo. Em *hardware* qualquer comutação do bit de controlo leva ao envio do byte respetivo. Aquando da execução da aplicação todos os *bits* pertencentes aos registos encontram-se por defeito inicializados a 0. Assim, e com recurso a uma variável do tipo *static* que mantém guardado o estado, este comuta sempre que é chamado o método sendo deste modo enviado o valor para o terminal.

Na Figura 5.27 estão ilustrados os parâmetros relativos à classe descrita anteriormente.

```
class Gpio_uart
{
private:
    int fd0;
    void* gpio_uart;

public:
    Gpio_uart();
    ~Gpio_uart();
    void write_uart( unsigned int wr);
    void sel_uart(unsigned int state);
};
```

Figura 5.27 - Parâmetros da classe Gpio_uart

5.4.4. Interface DMA

Para a leitura por parte do *software* dos dados enviados pelo DMA, é necessário aceder ao barramento de leitura do próprio módulo IP e seguir uma sequência de instruções de modo a dar início à transferência de dados. Inicialmente, e sendo a topologia já adotada nos interfaces anteriores recorre-se à função *mmap()* que instancia num endereço virtual o espaço de registos relativos ao DMA *Engine*. Na Tabela 5-8 estão descritos os registos relativos ao IP AXI DMA.

Tabela 5-8 - Registos AXI DMA

Endereço de Memória (Offset)	Nome do Registo	Descrição
0x00	*MM2S_DMACR*	*MM2S Registo de Controlo DMA*
0x04	*MM2S_DMASR*	*MM2S Registo de Status DMA*

0x14	*MM2S_SA*	*MM2S Source Address*
0x28	*MM2S_LENGTH*	*MM2S Comprimento da transferência (Bytes)*
0x30	*S2MM_DMACR*	*S2MM Registo de Controlo DMA*
0x34	*S2MM_DMASR*	*S2MM Registo de Status DMA*
0x48	*S2MM_DA*	*S2MM Destination Address*
0x58	*S2MM_LENGTH*	*S2MM Comprimento da transferência (Bytes)*

A configuração e iniciação da operação do DMA relativa à leitura (relativo ao canal S2MM) é introduzido pela seguinte sequência de processos:

- Iniciar o canal S2MM através da alteração do bit *run/stop* relativo ao registo *S2MM_DMACR* para 1. Esta alteração deve alterar o estado do bit *halted* pertencente ao registo *S2MM_DMASR* para 0 indicando que o canal S2MM se escontra a correr.
- Escrever um endereço de destino válido no registo *S2MM_DA*. Uma vez que o AXI DMA não foi configurado para realinhar os dados (*S2MM_DRE* = 0) e como foi definido um comprimento de dados de 32-bit (*S2MM_TDATA_WIDTH* = 32) então os dados só serão corretamente alinhados se estiverem localizados nos *offsets* das *words* (32-bit *offset*), isto é, 0x0, 0x4, 0x8, e assim sucessivamente.
- Escrever o número de bytes do *buffer* de receção no registo *S2MM_LENGTH*. A escrita deste valor inicia a escrita no interface S2MM AXI4 do número de *bytes* recebidos da FPGA. O número de *bytes* escritos no registo *S2MM_LENGTH* terá de ser igual ou superior ao número de *bytes* recebido pelo DMA caso contrário implicará a ocorrência de inúmeros erros indefinidos.
- Esperar para que o bit *IDLE* pertencente ao registo *S2MM_DMASR* passe para o estado 0, indicativo de que o processo foi concluído.
- Parar o processo, alterando o valor do bit *run/stop,* correspondente ao registo *S2MM_DMACR*, para 0.

Posto isto, é criada uma classe com um único método cuja função é iniciar o processo de leitura, descrito anteriormente. O parâmetro de entrada é um apontador para um vetor, o qual será preenchido com os valores transmitidos pelo DMA. A Figura 5.28 ilustra os parâmetros da classe.

```
class Dma_adc
{
private:
    int fd0;
    int fd1;
    void* dma_map;
    void* ddr_map;

public:
    Dma_adc();
    ~Dma_adc();
    void read(unsigned int* RxBuffer);
};
```

Figura 5.28 - Parâmetros da classe Dma_adc

6. Resultados

Este capítulo é dedicado à apresentação dos resultados que advêm da aplicações dos casos de uso já previamente descritos e que serviram para validar as funcionalidades da plataforma.

6.1. *Casos de Teste*

Neste subcapítulo serão realizados os casos de teste especificados no 0. Servirão para atestar as diferentes funcionalidades da plataforma e para comprovar o conceito de fácil implementação em contexto de complexidade de cálculos elevados como acontece para os controlos testados.

Para a aplicação destes casos de teste tiveram de ser definidos diversos aspetos relativos à plataforma. A aquisição é realizada através do ADC sendo a plataforma configurada para o modo de transferência de uma *word* por pacote à frequência que a plataforma conseguir atingir, de modo a que o sinal esteja disponível em *software* o mais rápido possível. É também definida a filtragem do sinal adquirido através da configuração do filtro FIR. Para a geração dos coeficientes do filtro definiu-se uma frequência de amostragem a 5MHz (frequência máxima do ADC no modo selecionado), de paragem a 1MHz e de 20KHz à frequência de banda passante. A amplitude estipulada foi de 120. Estes parâmetros resultam na geração de 40 coeficientes, que são prontamente transferidos para a BRAM. Posteriormente é enviado um sinal para a FPGA para que se inicie o processo de preenchimento dos registos relativos aos coeficientes do filtro.

Relativamente ao microsensor capacitivo usado no decorrer dos testes, incorpora uma microestrutura bidirecional tendo portanto dois pontos de atuação. A esses pontos estão ligados dois dos respetivos DACs existentes na placa de atuação, não esquecendo os *switches* existentes entre a estrutura e os conversores.

6.1.1. Controlo ON-OFF

O primeiro passo para a implementação deste controlo é a definição da tensão de atuação (V_{HIGH}). A escolha de um valor elevado leva a um deslocamento mais rápido por parte da estrutura, porém implica que haja um *ripple* maior. A escolha de um valor inferior leva a que a estrutura se desloque de maneira mais lenta porém o erro em relação à referência será menor.

É também necessário definir a tensão de referência que correspondente ao deslocamento no qual se pretende que a estrutura estabilize. Para efeitos de teste foram implementadas duas situações. Na primeira situação a referência é fixa e constante ao longo do tempo e na segunda situação a referência varia de acordo com um sinal previamente definido e de frequência constante.

Pela linha de comandos são introduzidas as tensões anteriormente descritas seguindo-se da implementação do próprio controlo que está inserido num ciclo infinito. Na Figura 6.1 encontra-se o código onde é aplicado o controlo ON-OFF.

```
dma_adc->read(&d_adc);
voltage = adc_to_voltage(d_adc);

if(voltage < v_ref && v_ref > 0)
{
      sw->write_pp(ESQ);
}
else if(voltage > v_ref && v_ref > 0)
{
      sw->write_pp(NONE);
}
else if( voltage < v_ref && v_ref < 0)
{
      sw->write_pp(NONE);
}
else if( voltage > v_ref && v_ref < 0)
{
      sw->write_pp(DIR);
}
else if( voltage == v_ref )
{
      sw->write_pp(NONE);
}
```

Figura 6.1 – Código controlo ON-OFF

A implementação do controlo encontra-se dividida em dois cenários. O primeiro onde a referência é positiva, sendo a atuação controlada pelo *switch* “ESQ”, e o segundo cenário onde a referência é negativa (*switch* “DIR”). Em ambos os casos se o módulo da tensão

medida for inferior à referência então o *swicth* entra no estado em que é aplicada a tensão de atuação. Caso o módulo da tensão seja superior, o *switch* comuta de estado para a massa.

Um dos pontos importante na implementação deste controlo é a frequência máxima de comutação dos *switches*, que quando mais elevado melhor será a reposta e mais exata será a posição da estrutura. Para a medição dessa frequência interromperam-se as conexões aos elétrodos e alternou-se sistematicamente o estado de um *switch*. A frequência do sinal à saída do *switch* corresponde à frequência máxima de comutação do controlo que é de 284KHz.

Como já referido anteriormente testaram-se duas situações. Para a primeira situação definiu-se uma tensão de atuação na ordem dos 5.7V e uma tensão de referência contante de 0.7V. Para a aquisição dos sinais analógicos aqui apresentados utilizou-se a placa de aquisição NI USB-6281 da National Instruments. Na Figura 6.2 mostram-se os resultados desta implementação.

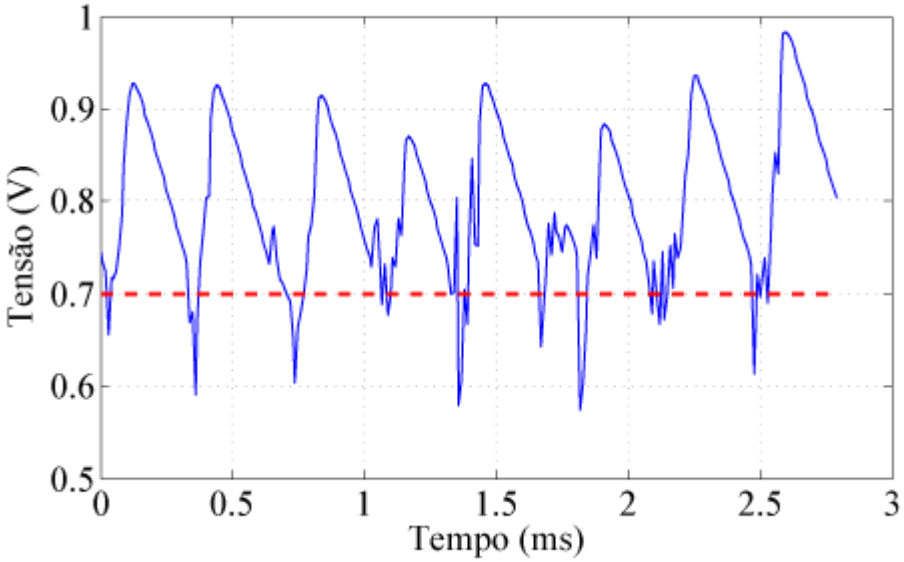


Figura 6.2 - Resultado Controlo ON-OFF Referência 0.7V

Como se pode verificar a estrutura mantem-se aproximadamente ao nível da referência. Os picos de deslocamento medidos (*ripple*) têm a ver sobretudo com a dinâmica da estrutura e atrasos na malha de realimentação que originam *ripple* [35]. Já relativamente ao tempo que a estrutura demora a descer para valores perto da referência está diretamente relacionada com a dinâmica da estrutura MEMS utilizada (sistema mecânico de segunda ordem).

Na segunda situação a tensão de referência não é fixa. A tensão de atuação definida foi novamente de 5.7V. É introduzido um sinal periódico e contínuo que a estrutura terá de

seguir. A cada N de ciclos de iteração é alterada a tensão de referência para valores disponíveis num vetor previamente gerado e onde se encontra o sinal sob a forma de “UM”. A Figura 6.3 mostra o resultado obtido.

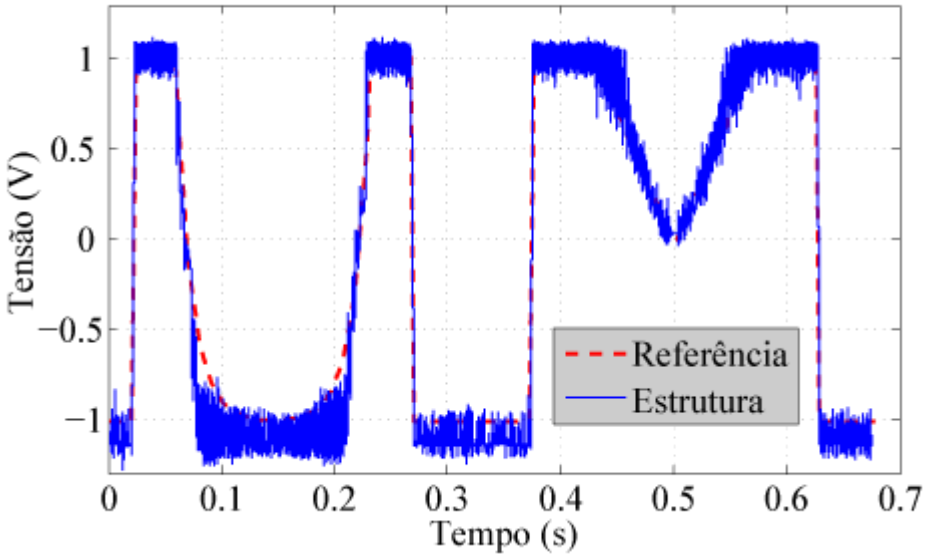


Figura 6.3 - Resultado Controlo ON-OFF referência "UM"

Como se pode verificar, a tensão à saída do circuito *readout* segue a tensão de referência como esperado. Apesar do ripple apresentado, os resultados são aceitáveis e cumprem com o pressuposto, isto é, facilmente se alteram as condições de operação do controlador, através da programação em alto-nível no Linux.

6.1.2. Feedback Linearization

O segundo caso de teste abordado está relacionado com a implementação do controlo *feedback linearization* descrito no 0. A variável de controlo, e ao contrário do que sucede no controlo ON-OFF é o próprio deslocamento da estrutura. Surge assim a necessidade de conversão do valor de tensão adquirida (que corresponde ao deslocamento) em deslocamento. Para tal, é imperativo chegar a uma função onde se relacione o deslocamento da estrutura em função da tensão medida à saída do circuito *readout*. Importante referir que os parâmetros necessários para a aplicação deste controlo dizem respeito apenas à estrutura em questão, sendo que para a utilização de uma outra estrutura terão de ser feitas novas medições e novos cálculos.

Na Tabela 6-1 encontra-se os parâmetros inerentes à estrutura utilizada e já previamente estipulados [9].

Tabela 6-1 - Parâmetros da estrutura

Parâmetros	
Massa (m)	$149.4*10^{-9}Kg$
Constante de elasticidade (K)	*2.6083N/m*
Distância inicial (d_0)	*3.1456μm*
Capacidade inicial (C_0)	*0.24771pF*

Estando a estrutura numa posição de equilíbrio, onde a aceleração gravítica não tem efeito, sabe-se que aplicando uma força externa com direção igual à do deslocamento, esta será igualada em módulo pela força elástica da mola. Surge assim a seguinte igualdade:

$$F_{ext} = F_{elástica} \Rightarrow a_{ext} * m = K * x$$

Onde x corresponde ao deslocamento que a estrutura sofre quando sofre uma aceleração a_{ext}. Para a construção de uma função que relaciona o deslocamento com a tensão de saída, recorreu-se a uma plataforma vibratória onde se produziram diferentes acelerações. Calculando o deslocamento produzido e guardando as tensões à saída do circuito de leitura correspondentes obtém-se a relação pretendida. Posto isto, restou apenas constituir um gráfico com os valores medidos ilustrado na Figura 6.4.

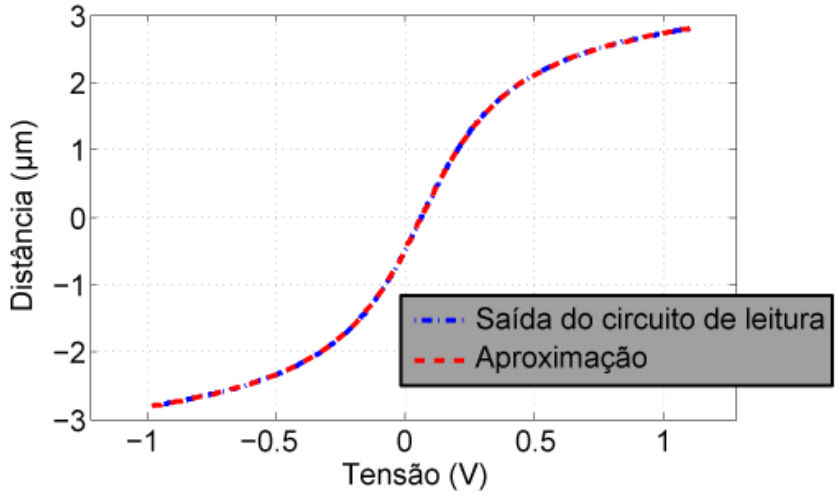


Figura 6.4 - Relação entre tensão e deslocamento da estrutura

Após convertida a tensão em deslocamento, segue-se a implementação do controlador baseado em *feedback linearization*. Para aplicação do mesmo, resta apenas definir o ganho proporcional. Após diversas implementações, optou-se por um ganho de 5, sendo este o que apresenta menor ganho em regime permanente, para além de apresentar um *overshoot* aceitável. Tendo em conta o diagrama de blocos ilustrado na Figura 4.21, chegou-se ao seguinte código mostrado na Figura 6.5.

```
erro = x_ref - x;
v_dac = (2*m*pow((d0-x),2)/(c0*d0)) * ((kp*erro+(k*x))/m);
if(v_dac < 0)    v_dac = 0;
v_dac = pow(v_dac, 0.5);
d_dac = voltage_to_dac(v_dac);
if(d_dac > 0xFFFFF) d_dac = 0xFFFFF;
```

Figura 6.5- Excerto de código relativo ao *feedback linearization*

Inicialmente é calculado o erro correspondente entre o deslocamento pretendido e o deslocamento medido. Posteriormente é aplicado o controlo, resultando num valor de tensão a aplicar ao microsensor, que dependendo da posição de referência, se seleciona o elétrodo respetivo. Importante observar que se limitou a atuação para o valor máximo admitido pelo DAC (10V). Com esta implementação foi atingida uma frequência de comutação de 100KHz fruto do tempo de cálculo e do atraso na comunicação PS/PL.

Tal como se sucedeu no controlo ON-OFF foi testado um sinal de referência com as siglas UM, obtendo-se o resultado ilustrado na Figura 6.6.

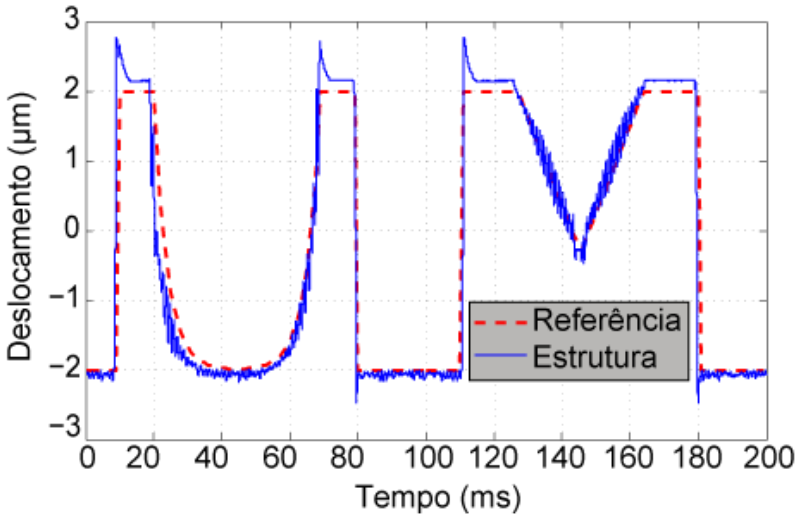


Figura 6.6 - Resultado *Feedback Linearization* referência "UM"

Como se pode observar a estrutura segue aproximadamente a referência pretendida. Verifica-se a existência de um *over-shoot* significativo quando se dá uma variação brusca do sinal de referência sendo normal atendendo às características que este tipo de controlo contém. Um outro aspeto a retirar é o erro em regime permanente observado quando a estrutura estabiliza sendo também uma característica intrínseca ao controlo.

6.1.3. Acelerómetro baseado em modulação Sigma Delta

Neste último caso de estudo implementou-se um acelerómetro baseado em modulação Sigma Delta. Para o teste deste modelo foi considerado como parâmetro de entrada a aceleração gravítica. Inicialmente definiu-se a tensão de atuação e estipulou-se o *offset* intrínseco ao sistema. Foi definida uma tensão de atuação de 7V, sendo esta a tensão mínima necessária para produzir uma força electroestática maior que a força gravítica máxima. É deste modo garantido que a estrutura se mantem sempre no ponto de referência, permitindo a medição correta da aceleração a toda a gama. Relativamente à tensão *offset*, esta corresponde à tensão disponível à saída do circuito de leitura quando não existem forças externas a atuar sobre a microestrutura. Esse valor de tensão medido foi de 70mV.

Uma vez definidos os parâmetros iniciais, percorreu-se toda a gama de acelerações gravíticas e procedeu-se ao cálculo da média do *bitstream* respetivo, obtendo-se uma relação entre as duas grandezas. Na Figura 6.7 está ilustrado o excerto de código relativo à aquisição dos estados binários e respetiva atuação.

```
while ( j<500000 )
{
    dma_adc->read(&d_adc);
    voltage = adc_to_voltage(d_adc);

    if(voltage < v_ref)
    {
        sw->write_pp(ESQ);
        vector[j] = 1;
    }
    else if(voltage > v_ref)
    {
        sw->write_pp(DIR);
        vector[j] = 0;
    }
    j++;
}
j = 0;
```

Figura 6.7 - Excerto de código relativo ao controlo digital em modelação sigma-delta

Como se pode observar na Figura 6.7 em cada ciclo de aquisição são armazenados 500000 estados binários que posteriormente são guardados em ficheiro para posterior análise. Nas Figura 6.8, Figura 6.9 e Figura 6.10 estão ilustrados os *bitstreams* resultantes da medição de diferentes acelerações gravíticas (-1g., 0g, 1g respetivamente).

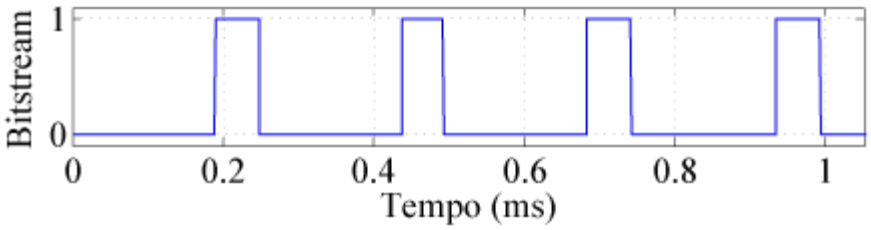


Figura 6.8 - *Bitstream* resultante de uma aceleração gravítica de -1g

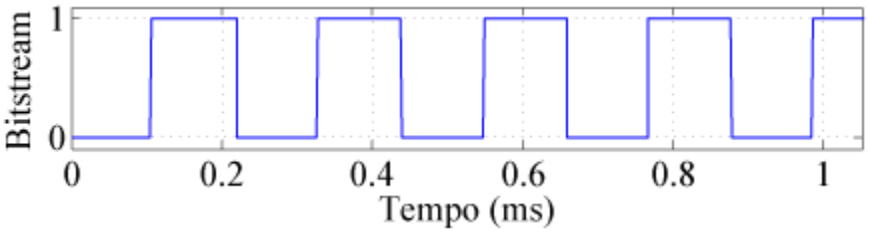


Figura 6.9 - *Bitstream* resultante de uma aceleração gravítica de 0g

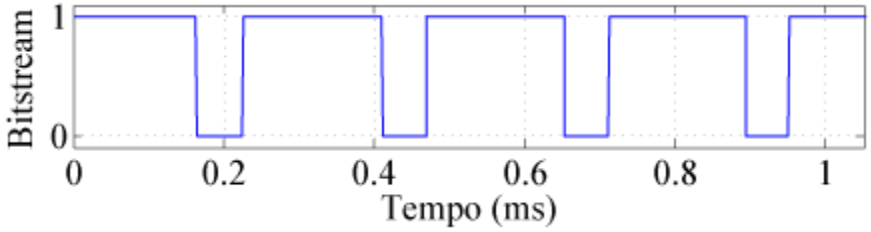


Figura 6.10 - *Bitstream* resultante de uma aceleração gravítica de 1g

Como se pode observar nas figuras anteriores, dependendo da aceleração a que a microestrutura está sujeita o tempo em que o *bitstream* se encontra no estado ‘1’ e no estado ‘0’ varia, conseguindo-se deste modo a relação pretendida.

Adquirindo os *bitstreams* relativos à medição de toda a gama de acelerações gravíticas obtém-se um gráfico onde se relacionam as duas grandezas, podendo este ser observado na Figura 6.11.

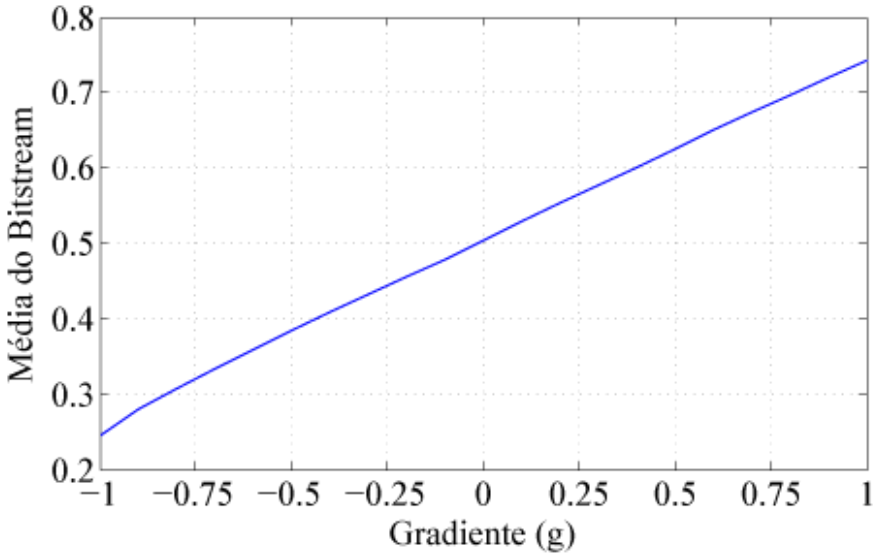


Figura 6.11 - Relação entre a aceleração e a média do *bitstream* resultante

Pela análise do gráfico observa-se que existe efetivamente uma relação entre a aceleração e a média obtida, sendo inclusive aproximadamente linear a toda a gama conseguindo-se deste modo provar o conceito de aplicação.

7. Conclusões e Trabalho Futuro

Como resultado do desenvolvimento desta dissertação foi apresentada uma plataforma de testes baseada em FPGA para microsensores capacitivos.

Com recurso à placa de desenvolvimento ZedBoard, foi possível a construção de uma plataforma que satisfaz os requisitos e as restrições estipuladas, resultando num sistema onde é possível implementar uma vasta gama de aplicações, recorrendo à linguagem de programação c++, não sendo necessário o conhecimento do *hardware* utilizado.

Os resultados obtidos através da implementação dos casos de uso foram bastantes positivos. Comprovou-se a viabilidade e flexibilidade da plataforma nos mais variados campos de aplicações. Porém, existe um atraso real que pode limitar algumas aplicações devido à limitação encontrada na transmissão de dados através dos barramentos utilizados.

Uma outra limitação prende-se com o fato do sistema operativo utilizado não ser um sistema de tempo real, apresentando muitas limitações aquando da necessidade de assegurar tempos de execução, o que pode colocar em causa o correto funcionamento do sistema. A implementação de um *soft core* com capacidade de hospedar um sistema de tempo real poderá resolver a limitação e inclusive melhorar a performance da plataforma.

Referências

[1] F. S. Alves, R. A. Dias, J. Cabral, J. Gaspar, and L. A. Rocha, "High resolution pull-in inclinometer," in *The 17th International Conference on Solid-State Sensors, Actuators and Microsystem, 2013 Transducers & Eurosensors XXVII:*, 2013, pp. 928–931.

[2] S. Dalola, V. Ferrari, and D. Marioli, "Micromachined piezoresistive inclinometer with oscillator-based integrated interface circuit and temperature readout," *Meas. Sci. Technol.*, vol. 23, no. 3, p. 035107, Mar. 2012.

[3] D. Xia, C. Yu, and Y. Wang, "A digitalized silicon microgyroscope based on embedded FPGA," *Sensors (Basel).*, vol. 12, no. 10, pp. 13150–66, Jan. 2012.

[4] H. Rodjegard, D. Sandstrom, P. Pelin, N. Hedenstierna, D. Eckerbert, and G. I. Andersson, "A digitally controlled MEMS gyroscope with 3.2 deg/hr stability," *13th Int. Conf. Solid-State Sensors, Actuators Microsystems, 2005. Dig. Tech. Pap. TRANSDUCERS '05.*, vol. 1, 2005.

[5] D. Keymeulen, C. Peay, D. Foor, T. Trung, A. Bakhshi, P. Withington, K. Yee, and R. Terrile, "Control of MEMS Disc Resonance Gyroscope (DRG) using a FPGA platform," in *IEEE Aerospace Conference Proceedings*, 2008.

[6] G. Qinglei, L. Huawei, M. Shifu, and H. Jian, "Design of a plane inclinometer based on MEMS accelerometer," in *Proceedings of the 2007 International Conference on Information Acquisition, ICIA*, 2007, pp. 320–323.

[7] Instrumentation and technology for MEMS and Sensors, "MEMS Characterization Platform, Solutions for Microsystems Characterization." [Online]. Available: http://www.itmems.it/MCP-Brochure_web.pdf.

[8] F. S. Alves, R. A. Dias, J. Cabral, L. A. Rocha, and J. Monteiro, "FPGA controlled MEMS inclinometer," in *IEEE International Symposium on Industrial Electronics*, 2013.

[9] V. A. Lourenço Lima, "Mecanismo de compensação de temperatura para sensores MEMS baseados na tensão de pull-in," Universidade do Minho, 2013.

[10] Xilinx, "ZedBoard, (Zynq Evaluation and Development Hardware) User's Guide," 2013. [Online]. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v1_9.pdf.

[11] Xilinx, "Zynq-7000 All Programmable SoC." [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[12] Xilinx, “Xilinx UG-585 Zynq-7000 Technical Reference Manual,” 2014. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf.

[13] Analog Devices, “1 ppm 20-Bit ± 1 LSB INL, Voltage Output DAC, AD5791,” 2013. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/AD5791.pdf.

[14] Analog Devices, “4.5Ω RON, Triple/Quad SPDT ±15V/+12V/±5V iCMOS Switches, ADG1433/ADG1434,” 2009. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/ADG1433_1434.pdf.

[15] L. A. Rocha, *Dynamics and Nonlinearities of the Electro-Mechanical Coupling in Inertial MEMS*. 2005.

[16] Texas Instruments, “16-BIT, 40/80 MSPS ADCs WITH DDR LVDS/CMOS OUTPUTS, ADS5560,” 2012. [Online]. Available: http://www.ti.com/lit/ds/symlink/ads5562.pdf.

[17] Future Technology Devices Internacional Ltd, “FT232R USB UART IC,” 2010. [Online]. Available: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf.

[18] Digilent, “Digilent Github U-Boot.” [Online]. Available: https://github.com/Digilent/u-boot-digilent.

[19] S. Kilts, *Advanced FPGA Design*. 2007, p. 355.

[20] Xilinx, “7 Series FPGAs Clocking Resources, User guide,” 2014. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf.

[21] A. V Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete Time Signal Processing*, vol. 1999. 1999, p. 870.

[22] Xilinx, “LogiCORE IP AXI GPIO v2.0, Product Guide,” 2014. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf.

[23] Xilinx, “Zynq-7000 All Programmable SoC: Concepts, Tools and Techniques (CTT),” 2012. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ug873-zynq-ctt.pdf.

[24] Xilinx, “LogiCORE IP AXI Block RAM (BRAM) Controller v3.0, Product Guide,” 2013. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_bram_ctrl/v3_0/pg078-axi-bram-ctrl.pdf.

[25] A. K. Agarwal, S. S. Sridharamurthy, and D. J. Beebe, "Programmable autonomous micromixers and micropumps," *J. Microelectromechanical Syst.*, vol. 14, no. 6, pp. 1409–1421, Dec. 2005.

[26] R. Müller-Fiedler and V. Knoblauch, "Reliability aspects of microsensors and micromechatronic actuators for automotive applications," *Microelectron. Reliab.*, vol. 43, no. 7, pp. 1085–1097, Jul. 2003.

[27] A. D. Ketsdever, R. H. Lee, and T. C. Lilly, "Performance testing of a microfabricated propulsion system for nanosatellite applications," *J. Micromechanics Microengineering*, vol. 15, no. 12, pp. 2254–2263, Dec. 2005.

[28] Xilinx, "Zynq-7000 All Programmable SoC Software Developers Guide," 2014. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf.

[29] Digilent, "Digilent Github Linux." [Online]. Available: https://github.com/Digilent/linux-digilent.

[30] Digilent, "ZedBoard Zynq™-7000 Development Board." [Online]. Available: http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,1028&Prod=ZEDBOARD.

[31] P. N. M. M. Morais, "Circuito de leitura digital para MEMS capacitivos," Universidade do Minho, 2013.

[32] Xilinx, "LogiCORE IP Multiplier v11.2," 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/mult_gen_ds255.pdf.

[33] Xilinx, "LogiCORE IP Adder/Subtracter v11.0," 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/addsub_ds214.pdf.

[34] Xilinx, "LogiCORE IP FIFO Generator v9.3, Product Guide," 2012. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v9_3/pg057-fifo-generator.pdf.

[35] L. Mol, E. Cretu, L. A. Rocha, and R. F. Wolffenbuttel, "Full-gap positioning of parallel-plate electrostatic MEMS using on-off control," *IEEE Int. Symp. Ind. Electron.*, pp. 1464–1468, 2007.

Anexos

A. Esquemático da placa de circuito impresso do sistema de atuação

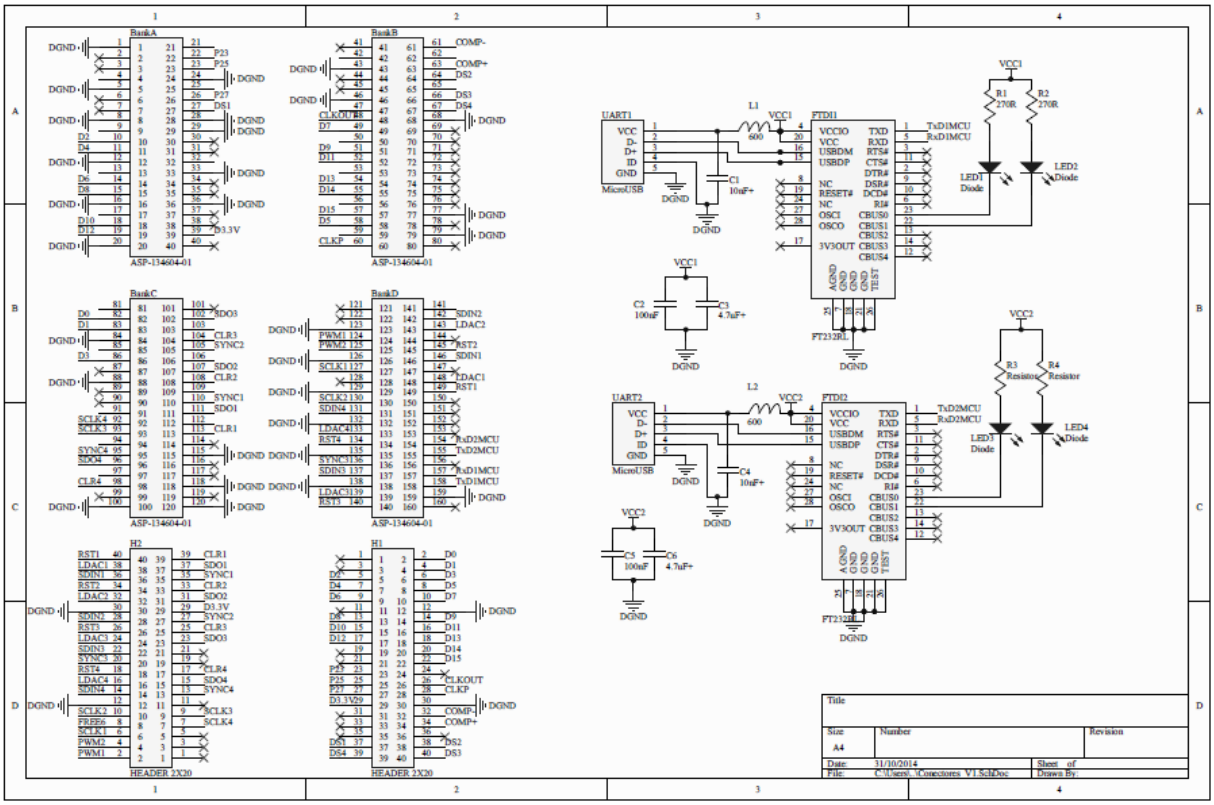


Figura A.1 - Esquemático da placa de circuito impresso

B. Layout da placa de circuito impresso

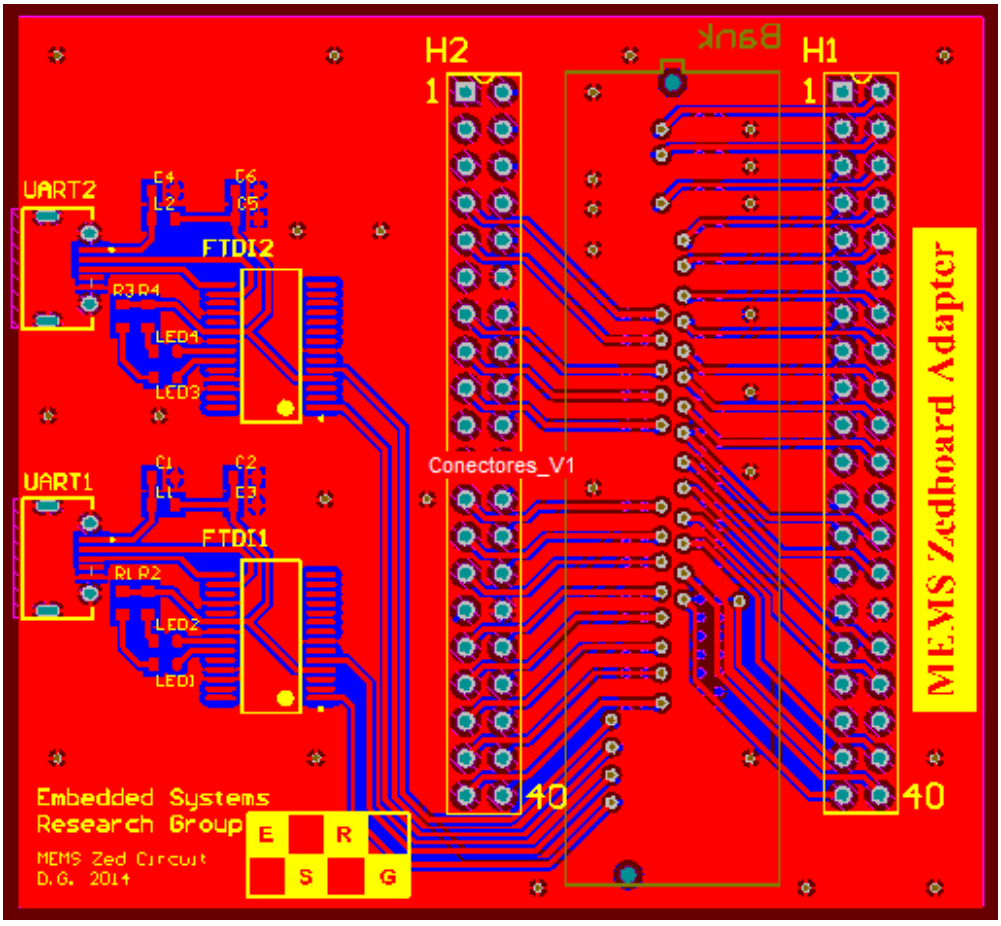


Figura B.1 - Layout da placa de circuito impresso (*top layer*)

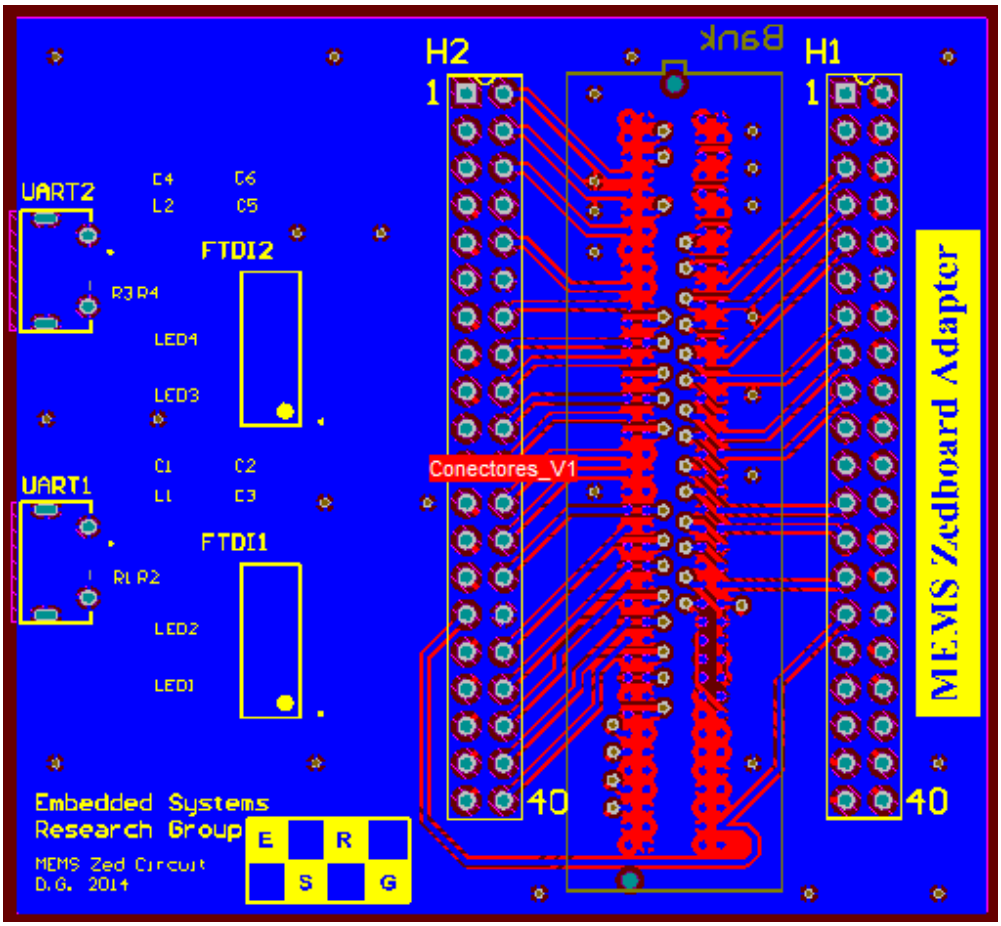


Figura B.2 - Layout da placa de circuito impresso (*bottom layer*)