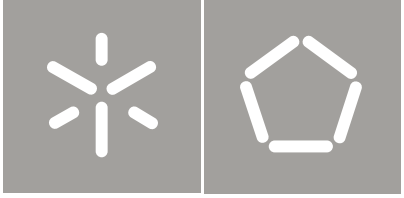


Universidade do Minho
Escola de Engenharia

Marcelo António C. Pereira e Sousa

Sistema Híbrido de Aquisição em Tempo-Real baseado em Linux

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao Grau de Mestre em Engenharia de Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Adriano José da Conceição Tavares

Dezembro de 2015

Declaração

Nome:
Marcelo António da Cruz Pereira e Sousa

Endereço Eletrónico: a61982@alunos.uminho.pt

Telefone/Telemóvel: 93 448 98 76

Número do Bilhete de Identidade: 14209719

Título da Dissertação: Sistema Híbrido de Aquisição em Tempo-Real baseado em Linux

Orientador: Professor Doutor Adriano José da Conceição Tavares

Designação do Mestrado:
Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Universidade do Minho, ____/_____/_____
Assinatura:_____

Agradecimentos

"Primeiro de tudo queria agradecer à minha família pelos momentos de apoio proporcionados, apesar da distância que nos separa. Assumiram um papel fundamental nesta meta final da minha carreira académica.

Aos meus amigos, colegas de curso e colegas de quarto quero agradecer-lhes por toda a ajuda e apoio não só profissionalmente, mas também emocionalmente recebidos no decorrer destes últimos anos. Foram essas pessoas que tornaram um ciclo de estudos académicos exigente num processo mais fácil.

Também gostaria de agradecer pelo serviço breve mas óptimo, por parte dos funcionários do Departamento de Eletrónica Industrial, que me ajudaram nesta dissertação através dos seus conhecimentos.

Estou agradecido ao meu orientador pela transmissão de conhecimentos que me guiaram ao longo do desenvolvimento desta dissertação, pela sua disponibilidade para a discussão de aspetos deste projeto e por todo o seu apoio.

Por fim, ao engenheiro, Vitor Silva, estou extremamente agradecido por todo o conhecimento que ele transmitiu, na forma de sessões de *brainstorm*, em todas as fases deste projeto final. A sua exigência fez-me dar o máximo para alcançar o maior nível de conhecimento possível nas áreas de especialização que eu escolhi. Graças a ele, eu consegui tirar o máximo proveito deste último ano da minha carreira académica.

A todos expresso a minha profunda gratidão por toda a ajuda e apoio recebidos"

Abstract

A large number of embedded systems for industrial applications are based on data acquisition system, due to the fact they must quantify the relevant information from the real world, in the proper format, to be analyzed and processed by the digital system, in order to be able to actuate on their environment. Since the real world is analog, the demand for real time, reliable, with higher resolutions and faster acquisition systems has been growing. The acquisition system that is meant to be developed tackles the requirements stated previously, therefore being able to be used in control and monitor systems in the several sectors of the modern industry, such as automotive, space industry, medicine, aviation, among others.

In this dissertation, a generic parametrizable real-time acquisition system was developed, based on Linux OS, in order to ensure the fulfillment of the previously mentioned real-time requirements, some tasks were offloaded into dedicated hardware implemented in a FPGA. This system provides a set of functionalities and services that allows to control it, hiding the details of the implementation from the user/designer. As a mean of validation of this system, it was developed an application scenario in the energy systems' domain where the system's functionalities are tested and verified.

Resumo

A maioria dos sistemas de monitorização e controlo presentes na indústria moderna são implementados tendo por base sistemas de aquisição em tempo-real, uma vez que é necessário quantificar as correspondentes grandezas físicas do mundo real para o formato adequado de modo a que as componentes digitais do sistema possam atuar sobre esses dados adquiridos. Uma vez que o mundo real é analógico, a procura por sistemas de aquisição tempo-real, com altas-resoluções, mais rápidos, seguros, fidedignos e com baixo consumo energético tem vindo a aumentar. Os vários benefícios tecnológicos do sistema operativo Linux, como a portabilidade, configurabilidade e modularidade, impulsionou a sua adoção por parte dos sistemas embebidos. O sistema a desenvolver tem de cumprir os requisitos estabelecidos, satisfazendo a maioria dos sistemas industriais com requisitos de tempo-real. De modo a assegurar que estes requisitos temporais são cumpridos são utilizadas técnicas de aceleração do processador em hardware recorrendo a FPGA. Este sistema de aquisição oferece um conjunto de serviços e funcionalidades que permitem abstrair a complexidade e os detalhes de implementação ao utilizador/designer de um sistema maior. Posteriormente, de forma a verificar e validar o sistema desenvolvido, este foi integrado num cenário da aplicação no domínio de sistemas de energia.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Neste documento é descrito o trabalho realizado no âmbito de dissertação do curso de Mestrado em Engenharia Eletrónica Industrial e Computadores, cujo resultado final é um sistema genérico parametrizável de aquisição em tempo-real. Nas secções que se seguem é descrito o contexto no qual esta dissertação se insere, qual a necessidade ou problema que pretende colmatar, quais os motivos que levaram o autor a desenvolvê-la, seguido dos objetivos e especificações estipulados e sua relevância-técnico-científica e, por fim, é descrito como este documento se encontra estruturado.

1.1 Contextualização

A aquisição de dados é um processo de amostragem de sinais, responsável por quantificar uma determinada grandeza física, proveniente do mundo exterior (ambiente de aplicação), no formato adequado de forma a ser analisada e processada por um sistema computacional. Os sistemas que se baseiam neste processo, denominados de sistemas de aquisição, permitem obter dados referentes a diversos tipos de grandezas físicas (e.g., tensão e corrente elétrica, pressão, temperatura, posição e porventura sinais de natureza digital) e, na maioria dos casos, o seu propósito não se limita apenas à própria aquisição de dados, mas também ao processamento e controlo conforme a aplicação alvo. Cada sistema de aquisição é caracterizado pela sua aplicação alvo, a qual dita os sinais a adquirir do mundo exterior, a quantidade de dados recolhidos, como disponibiliza tais dados para o utilizador/programador e que processamento exercita sobre a informação obtida.

Os sistemas de aquisição baseiam-se na ideia de sistema embebido, no sentido em que são sistemas computacionais desenhados para realizar determinada tarefa ou conjunto de tarefas utilizando apenas o hardware e software necessários. Tais sistemas, na maioria dos casos, fazem parte de um sistema maior, pelo que a sua presença pode não ser notada pelo utilizador. Para além disto, a maioria destes sistemas de aquisição possui requisitos temporais críticos no controlo de alguns processos na indústria moderna, como por exemplo, nos sistemas espaciais, sistemas militares ou sistemas médicos, os quais são classificados como sistemas *hard real-time*, em que o incumprimento de qualquer *deadline* pode comprometer a integridade do sistema ou ainda pôr em risco a vida do utilizador. Desta forma, soluções bem definidas têm de ser promovidas através a utilização de mecanismos e processos rígidos. Na Figura 1.1 estão ilustradas algumas áreas onde este tipo de sistemas é utilizado.

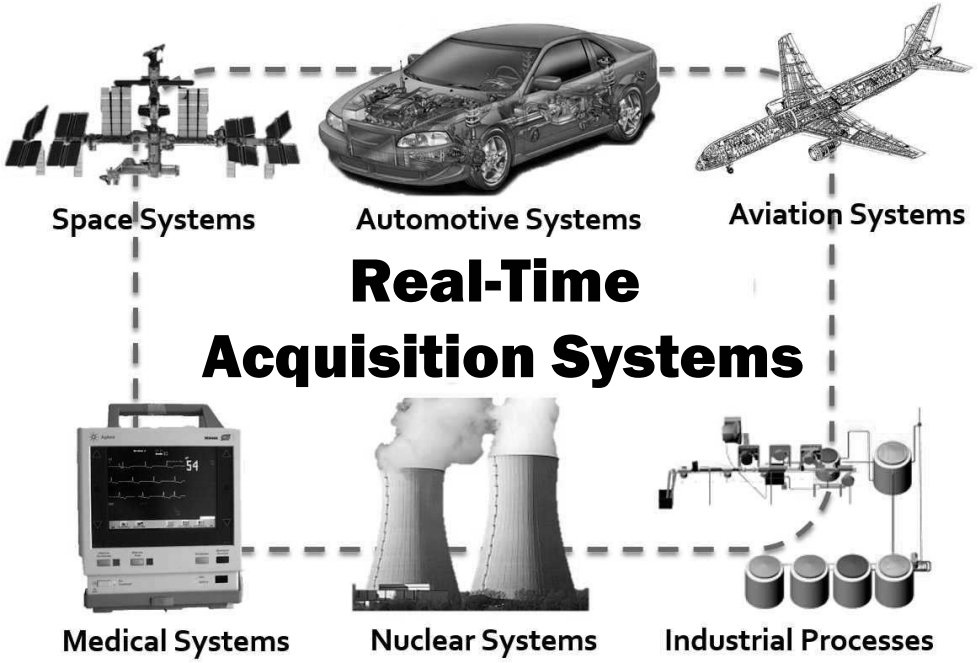


Figura 1.1: Setores da indústria onde sistemas de aquisição são usados.

Em vários setores da indústria existe uma procura crescente nos sistemas de aquisição por amostragens simultâneas, precisão e exatidão nos resultados, fiabilidade, segurança, e também por determinadas funcionalidades como capacidade de armazenamento e protocolos de comunicação (Universal Serial Bus (USB), Ethernet) para efeitos de comunicação com o exterior. O cumprimento destes requisitos aliado a um ambiente que providencie segurança no acesso aos dados, facilidade de controlo e gestão justa dos recursos, e o correto funcionamento do sistema podem

ser assegurados através do uso de um sistema operativo. A utilização do sistema operativo Linux em sistemas embebidos tem vindo a aumentar devido aos seus vários benefícios tecnológicos, como a sua escalabilidade (desde aplicações orientadas ao consumidor até sistemas com uma complexidade elevada), um vasto suporte de plataformas de hardware e *device drivers*, a portabilidade das aplicações e consequente reutilização, e a sua capacidade de integração de bibliotecas no domínio da aplicação.

Os sistemas de aquisição que usufruem de uma distribuição Linux, tornam-se uma solução capaz de colmatar os requisitos da maioria dos sistemas presentes na indústria moderna que beneficia também da virtualização de hardware garantida por este sistema operativo podendo assim, serem utilizados independentemente da plataforma. Contudo, o determinismo, previsibilidade, baixas latências de sistemas do tipo *hard real-time* tornam-se requisitos difíceis de satisfazer num ambiente de sistemas operativo, pelo que surge a necessidade da utilização dos modernos dispositivos com tecnologia Field-Programmable Gate Array (FPGA) de forma a mitigar tais problemas, uma vez que estes dispositivos permitem o *offload* por computação de tarefas críticas em hardware.

1.2 Motivação e Objetivos

O crescimento na utilização de sistemas embebidos potencia o crescimento e inovação em vários setores da indústria moderna, como por exemplo em sistemas de telecomunicações, transportes, sistemas espaciais, onde novas tecnologias estão continuadamente a emergir. O desenvolvimento destes sistemas requer uma *expertise* nas ferramentas de modelação e simulação de maneira a antecipar decisões que asseguram que a aplicação satisfaz os seus objetivos e requisitos e, levando assim, a um aumento da produtividade e diminuição do *time-to-market*. A adoção do sistema operativo Linux por parte dos sistemas embebidos tem vindo a aumentar devido às suas características tecnológicas, pelo que também possui o suporte de uma comunidade científica que trabalha continuamente para o seu melhoramento. A variedade de distribuições de Linux e os serviços oferecidos pelo sistema operativo permitem adquirir um conhecimento no domínio dos sistemas embebidos, pelo que despertou o interesse do autor.

O desenvolvimento deste tipo de sistemas com requisitos muito exigentes, requer a utilização de métodos de *co-design* de hardware-software e consequentemente

competências chave centradas neste domínio de forma a satisfazer os níveis de performance, consumo energético, fiabilidade e segurança do sistema. Estas competências aliadas à integração das duas áreas especialização do mestrado (Sistemas Embebidos e Eletrotecnia e Sistemas de Energia) tornaram o desenvolvimento do projeto atrativo para o autor.

1.3 Contribuições

Do ponto de vista da performance exigida nas plataformas de hardware, a crescente complexidade dos sistemas usados na indústria e da variedade dos dispositivos que realizam tarefas com restrições temporais originam um aumento no custo total do sistema e no *time-to-market*. Na maioria destas aplicações, os custos associados aos recursos de hardware e software podem ser reduzidos através da utilização de uma distribuição Linux adaptada aos requisitos do sistema. É expetável que a identificação de variáveis chave que afetam as tarefas com restrições temporais contribuam para a redução do esforço de engenharia empregue, e também no *time-to-market* requeridos para o desenvolvimento deste tipo de aplicações.

1.4 Estruturação do Documento

Nesta dissertação é descrito o desenvolvimento de um sistema genérico parametrizável de aquisição de dados baseado em Linux em cinco capítulos discutidos de seguida.

O segundo capítulo apresenta uma visão geral dos conceitos e tecnologias necessários para o co-desenvolvimento de *hardware-software* deste sistema embebido com requisitos temporais exigentes, o qual é um sistema complexo com suporte do sistema operativo Linux com algumas componentes da aplicação migradas para hardware nos dispositivos FPGA.

O capítulo seguinte descreve rigorosamente as características de algumas plataformas que podem ser empregues. Esta descrição compreende as principais componentes e funcionalidades de acordo com o sistema proposto, finalizando com o critério de seleção que levou à escolha da plataforma utilizada.

No capítulo quatro é descrito o *co-design* de hardware-software do sistema de aqui-

sição proposto, incluindo: a placa de aquisição de sinal responsável pela aquisição do sinal proveniente do mundo exterior no formato adequado para a componente digital do sistema; a caracterização do sistema de aquisição descrevendo o comportamento; o sistema de controlo, sendo o *core* do sistema de aquisição que realiza as respetivas ações para recolher e processar os dados adquiridos; e a aplicação do sistema que disponibiliza um conjunto de serviços do sistema de aquisição.

No capítulo cinco, o sistema de aquisição proposto é integrado em vários cenários de aplicação, cujos resultados permitem verificar e validar as funcionalidades, serviços e performances da tecnologia do sistema. Para cada cenário de aplicação é feita uma descrição do mesmo e como o sistema proposto interage com este.

O último capítulo apresenta uma breve conclusão acerca do sistema proposto, discutindo os conceitos e tecnologias abordadas e os resultados recolhidos dos testes realizados, o que foi concretizado e o que poderá ser feito como trabalho futuro.

Capítulo 2

Estado da Arte

Neste capítulo é apresentada uma visão geral dos conceitos e tecnologias abordados nesta dissertação. Uma vez que os objetos de estudo são sistemas de aquisição em tempo-real modulados como sistemas embebidos, são apresentados conceitos centrados neste domínio. Estes sistemas embebidos são desenhados com um sistema operativo de suporte, por isso é feita uma breve descrição de sistemas embebidos e da sua constituição, discutindo também o sistema operativo Linux e as suas principais características com respeito a sistemas embebidos com requisitos tempo-real. Para além disto, é apresentada uma descrição sucinta da tecnologia FPGA descrevendo os principais elementos que a constituem, e um típico *workflow* usado para criar aceleradores dedicados em hardware para melhorar a performance do sistema embebido. Por fim, são discutidas as principais componentes de um sistema de aquisição, desde a aquisição dos dados do mundo exterior até à sua disponibilização ao utilizador/programador.

2.1 Sistemas Embebidos

Um sistema embebido é um sistema computacional que é desenhado especificamente para realizar um certo conjunto de tarefas e é, geralmente, encapsulado pelo dispositivo que controla, tornando-o invisível para o utilizador. Devido à sua aplicação restrita, a sua performance, tamanho, consumo energético e custo total podem ser otimizados pelos *designers* do sistema. Sistemas embebidos existem em vários ambientes, tamanhos e formas, e englobam sistemas computacionais desde os mais simples orientados ao consumidor (e.g., MP3 *player*, televisão) até a siste-

mas com um maior nível de complexidade, como Asymmetric Digital Subscriber Line (ADSL) *routers* e impressoras. Desta forma, a complexidade de cada sistema embebido é ditada pela sua aplicação alvo, pelo que um sistema embebido pode ser uma simples aplicação em *bare-metal* bem como um sistema com elevada capacidade de processamento com um sistema operativo de suporte. No entanto, cada sistema embebido possui as seguintes características em comum:

- É orientado à aplicação uma vez que é especialmente desenhado para realizar um específico conjunto de tarefas, completamente por si próprio (*standalone*) ou parcialmente dependente de intervenção humana;
- Interage com elementos físicos do mundo exterior, como no controlo de motores elétricos ou no controlo da temperatura dentro de uma estufa;
- Possui uma interface dedicada para o utilizador (Entradas/Saídas), como por exemplo Graphical User Interface (GUI), interface Light Emitting Diode (LED), um *keypad* e/ou botões;
- Deve ser eficiente, com baixo consumo energético, baixo custo, rápido, pequeno, leve, baixo *code footprint*, contudo o *designer* tem de lidar com vários *trade-offs*;
- Deve ser um sistema fidedigno, na medida que cumpre os seguintes aspetos:
  - **Fiável:** capaz de operar corretamente em todas as situações;
  - **Sustentável:** capaz de manter o sistema operacional após a ocorrência de uma falha
  - **Disponível:** está pronto para utilização;
  - **Seguro:** não causa eventos desastrosos em caso de falha, e providencia autenticação e confidencialidade na comunicação.

Ao contrário de um sistema computacional de propósito geral, como um Personal Computer (PC), um sistema embebido executa um conjunto restrito de tarefas com requisitos muito específicos possuindo, assim, menos recursos de hardware, e pode variar desde um sistema extremamente simples (*bare-metal*) até um sistema mais complexo que tira proveito da flexibilidade proporcionada por um ambiente de sistema operativo, como discutido anteriormente. Contudo, a utilização de um sistema operativo traduz-se num aumento do custo total do sistema uma vez que requer mais recursos de hardware.

O desenvolvimento de um sistema embebido requer o *design* conjunto de hardware e software de forma a alcançar uma solução bem definida que satisfaça os requisitos da aplicação alvo. Por isso, torna-se necessário o domínio de ambas as componentes de hardware e software e a utilização de metodologias eficazes durante o *design* do sistema.

2.1.1 Constituição de um Sistema Embebido

Como discutido previamente, os sistemas embebidos possuem os mais distintos tamanhos, formas e ambientes de aplicação contudo, existem algumas partes que são comuns a todos. Na Figura 2.1 está ilustrada a visão geral de um sistema embebido genérico.

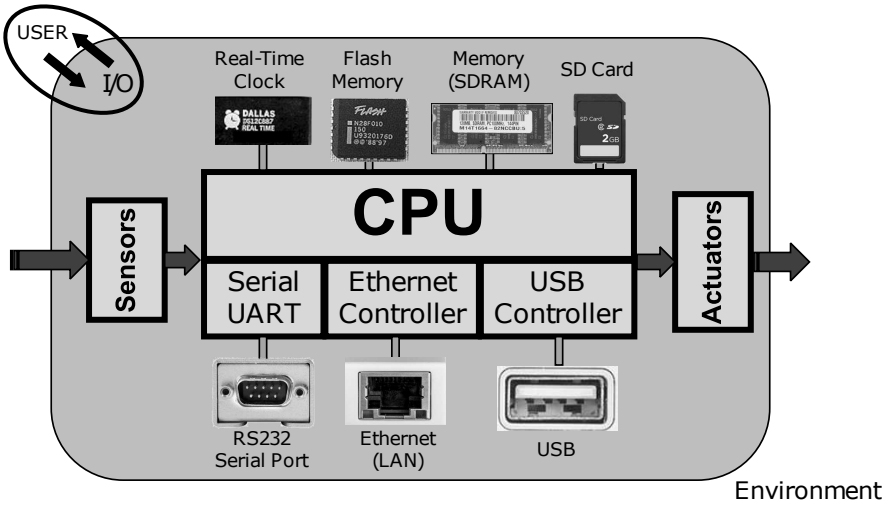


Figura 2.1: Diagrama simplificado de um sistema embebido genérico.

De uma forma genérica, um sistema embebido possui os seguintes elementos:

- **Central Processing Unit (CPU)** atua como o cérebro do sistema, controlando todas as operações e processando todas as informações que entram e saem do próprio sistema;
- **Memória Synchronous Dynamic Random Access Memory (SDRAM)** é um meio de armazenamento temporário e de controlo dos dados;
- **Memória *flash*** é uma memória não volátil que permite o armazenamento de programas e de dados extra;
- **Real-Time Clock (RTC)** controla a informação relativa à data e hora;

- **Secure Digital (SD) Card** é um elemento de armazenamento que funciona como complemento ou em associação à memória física do sistema;

- **Protocolos de Comunicação (USB, Ethernet, RS232)** servem para comunicação com o ambiente exterior.

Alguns sistemas embebidos são desenvolvidos em torno de um System-on-Chip (SoC). Este circuito integrado engloba vários componentes num único *chip* sendo capaz de possuir uma unidade de processamento potente com vários periféricos acoplados num baixo fator de forma, o que o torna uma solução atrativa no mercado das aplicações móveis. Na Figura 2.2 encontra-se representado um diagrama simplista de um SoC.

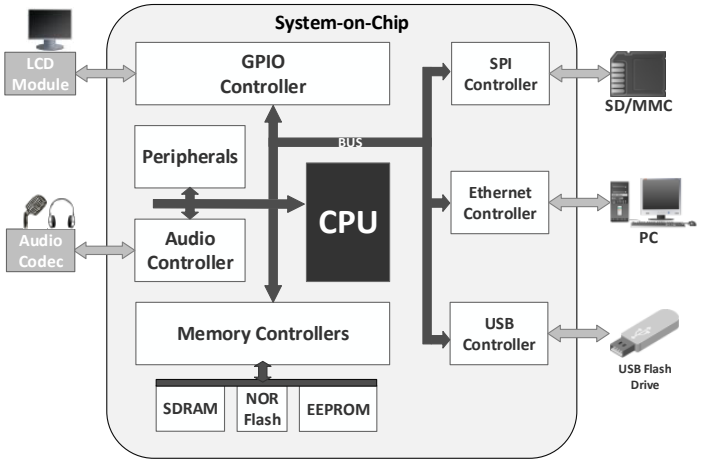


Figura 2.2: Diagrama simplificado de um System-on-Chip.

Por um lado, um microcontrolador contém um processador, as unidades de memória e os periféricos de entrada/saída embutidos no mesmo *chip*, pelo que destina-se a aplicações embebidas de baixa complexidade. Por outro lado, um SoC é tipicamente constituído por uma unidade de processamento com maior poder computacional, as unidades de memória, e vários periféricos que permitem executar um sistema operativo como o Linux. De salientar, que os dispositivos mais modernos possuem também uma área de lógica programável.

Como ilustrado na Figura 2.2, as características e constituição de um SoC tornam-no na solução ideal para sistemas embebidos que requerem um elevado poder computacional numa área reduzida com várias funcionalidades. De seguida, é apresentada uma breve descrição das principais partes que constituem um sistema embebido.

Unidade de Processamento Central

A unidade de processamento central é responsável pela execução de instruções de um programa, realizando as respetivas operações aritméticas, lógicas, de controlo e de entrada/saída. Ao longo dos anos, esta unidade tem sofrido alterações, no entanto os seus principais constituintes, ilustrados na Figura 2.3, permanecem inalterados: a unidade de controlo e o *datapath*.

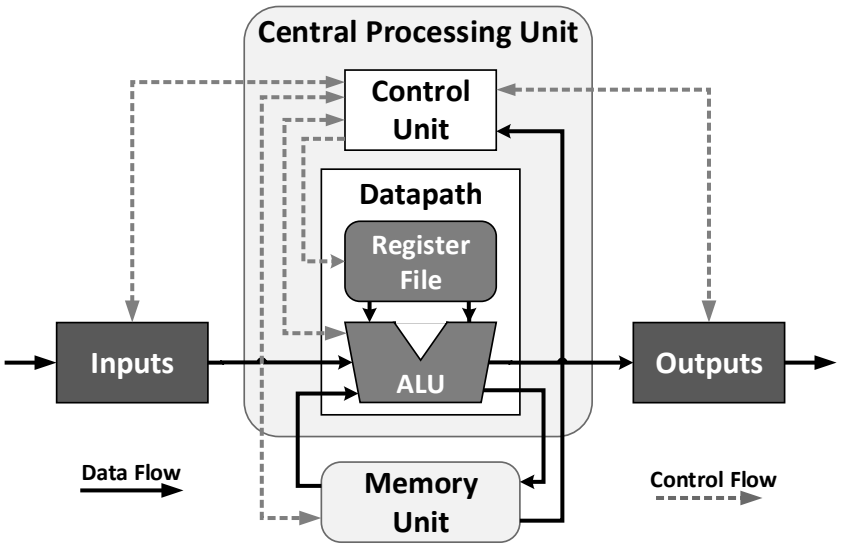


Figura 2.3: Diagrama simplificado da estrutura de um CPU.

O *datapath* é responsável por realizar todas as operações especificadas pela respetiva Instruction Set Architecture (ISA) e é, geralmente, composta por uma Arithmetic Logic Unit (ALU) que realiza as operações aritméticas e lógicas e, por um conjunto de registos que fornecem um meio de armazenamento interno para mover dados entre a memória e as unidades funcionais do CPU. Por outro lado, a unidade de controlo é uma máquina de estados que controla as operações do *datapath* de acordo com a instrução em execução que se encontra na memória interna do sistema. Este processo pelo qual o CPU lê a instrução da memória, determina quais as ações que desencadeia e executa as tais ações. Este ciclo de instrução (*fetch-decode-execute*) é continuamente repetido pelo CPU desde o arranque até o sistema ser desligado.

Todas as instruções e elementos de armazenamento (registos) disponíveis no CPU são definidos pelo ISA de forma a executar as respetivas operações. Desta maneira, o CPU pode ser classificado conforme este conjunto de instruções, como Complex Instruction Set Computer (CISC) como Reduced Instruction Set Computer (RISC). Na arquitetura CISC, o processador é capaz de executar um maior

número de instruções do que na arquitetura RISC, aferindo versatilidade e simplicidade no desenvolvimento do próprio compilador. Por sua vez, na arquitetura RISC o processador executa um conjunto menor de instruções o que origina uma maior performance do que na arquitetura CISC.

Interrupções

De forma a reagir a certos eventos é necessário interromper o CPU para lidar com o evento recebido. Isto é conseguido através interrupções. Nestes casos, o CPU não segue o fluxo normal de execução uma vez que quando um evento ocorre, este pausa e guarda o contexto atual de execução (guarda o *program counter* e outros registos na pilha), terminando com o *program counter* a carregar do endereço da Interrupt Service Routine (ISR) do vetor de interrupções. A ISR é responsável por desencadear as ações em resposta ao evento e quando termina, o CPU restaura o contexto de execução anterior da pilha na ordem inversa e retoma o fluxo normal de execução.

A interrupção pode ser desencadeada por software por resposta a um evento externo através do respetivo sinal de interrupção. Na Figura 2.4 encontra-se representado a transferência de controlo do CPU quando uma interrupção ocorre.

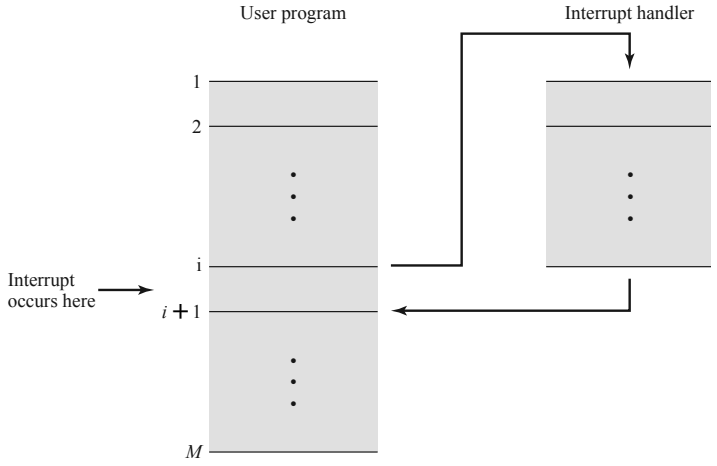


Figura 2.4: Transferência de controlo do CPU na ocorrência de uma interrupção [16].

Barramentos

Os barramentos estabelecem um meio de comunicação entre os elementos que constituem o sistema embebido através da utilização de um barramento de dados, um barramento de endereços e um barramento de controlo. As portas de entrada e saída de cada elemento são o elo de ligação por onde os dados são transmitidos entre todos os elementos conectados ao barramento, como o CPU, a memória, os dispositivos de Input/Output (I/O). O propósito dos barramentos do sistema é a comunicação entre os vários elementos do sistema, a qual é realizada através dos seguintes barramentos:

- **Barramento de Dados:** proporciona um caminho para a transferência dos dados entre os elementos do sistema, cuja largura determina a máxima quantidade de dados que pode ser transferida de uma vez;
- **Barramento de Endereços:** designa a origem ou destino dos dados a serem transferidos via o barramento de dados uma vez que cada elemento possui a sua própria gama de endereços. A largura deste barramento dita a capacidade máxima de endereçamento da memória;
- **Barramento de Controlo:** transmite informação de comando (especifica as operações a serem executadas) e de temporização (indica se o endereço e os dados são válidos) entre os elementos do sistema, que por sua vez é usada no controlo do acesso e utilização dos barramentos de dados e de endereços.

A Figure 2.5 apresenta um esquema simplificado da interconexão entre o CPU, as memórias, os dispositivos de I/O e os barramentos do sistema.

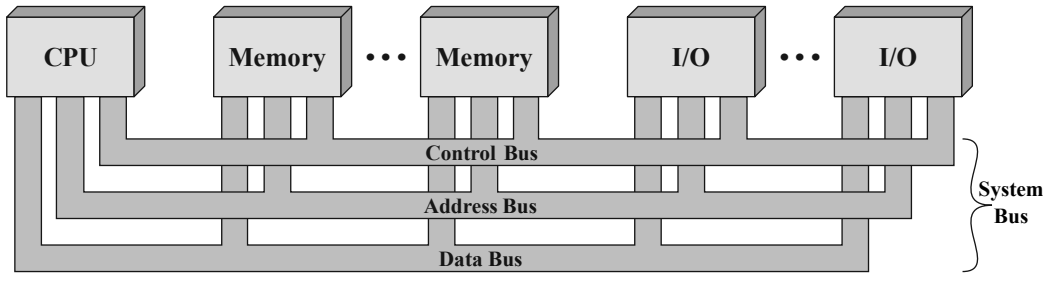


Figura 2.5: Esquema simplificado da interconexão dos elementos do sistema [16].

As principais operações realizadas pelo CPU envolvem a leitura e escrita para a memória principal do sistema. Na leitura dos dados, o CPU carrega o endereço da memória para o barramento de endereços e depois os sinais de controlo da memória

são carregados para o barramento de controlo. No próximo passo, o controlador de memória carrega os dados do respetivo endereço de memória para o barramento de dados, onde o CPU irá lê-los. No processo de escrita, o CPU carrega o endereço de memória para o barramento de endereços e os dados para o barramento de dados. Neste ponto, os sinais de controlo da memória são carregados para o barramento de controlo. Este processo termina com o controlador de memória a ler os dados do barramento e colocá-los no endereço presente no barramento de endereços.

Tecnologias de Memória

Nos sistemas computacionais, a memória é um dos principais componentes porque é responsável pelo armazenamento de dados e instruções no formado digital. Atualmente, a vanguarda das tecnologias de memória compreende:

- Read-Only Memory (ROM): memória não-volátil (i.e., mantém a informação quando o sistema é desliga) cujos dados se encontram *hardwired*, logo apenas é possível a leitura neste tipo de memórias;
- Random-Access Memory (RAM): memória volátil (i.e., perde a informação quando o sistema deixa de ser alimentado) que permite a leitura e escrita de dados.

Memória ROM

ROM é uma memória não volátil que contém um padrão permanente de dados que não é facilmente alterado ou reprogramado, por isso é, geralmente, programado apenas uma vez. Um exemplo de utilização deste tipo de memórias é a Basic Input/Output System (BIOS) de um PC que é lançada no arranque do computador. Os principais tipos de memória ROM são:

- **Programmable Read-Only Memory (PROM):** o processo de escrita é realizado eletricamente através de equipamento especial. Esta memória não pode ser alterada ou reprogramada depois de concluída a escrita dos dados;
- **Erasable Programmable Read-Only Memory (EPROM):** este tipo de memórias permite a reprogramação dos dados existentes no próprio semicondutor através de um dispositivo de emissão de luz ultravioleta que apaga os dados no início do processo de escrita;

- **Electrically-Erasable Programmable Read-Only Memory (EEPROM):** esta tecnologia permite também a reprogramação dos dados, mas ao contrário da EPROM, os processos de leitura e escrita são feitos eletricamente, removendo, assim, a necessidade de dispositivos auxiliares;

- ***Flash*:** o comportamento destas memórias assemelha-se ao das EEPROM, contudo os processos de escrita são mais rápidos. Para além disto, este tipo de memórias possuem uma maior capacidade de armazenamento, menor consumo energético e uma maior robustez, no entanto apresentam um maior custo. Existem dois tipos distintos de memória *flash*:

  - ***NOR flash:*** proporciona um rápido acesso na escrita e leitura em localizações específicas da memória. Por exemplo, esta tecnologia é usada para armazenar o programa da BIOS no arranque de um computador;

  - ***NAND flash:*** ao contrário das memórias NOR *flash*, neste tipo de memórias não é possível aceder aleatoriamente à memória. O acesso aos dados é feito por blocos, em que cada bloco contém uma grande quantidade de dados, por isso, estas memórias são usadas em USB *flash drives*, cartões de memória e em Solid-State Drives (SSD).

Memória RAM

Memória RAM é uma forma de armazenamento de dados que pode ser acedida (leitura e escrita) aleatoriamente a qualquer momento, em que cada localização endereçável possui um único mecanismo físico de endereçamento que permite um rápido acesso e manipulação. Este tipo de memória é uma das partes fundamentais de um sistema informático, uma vez que é utilizada para guardar os dados sobre os quais o CPU está trabalhar. Esta memória é não volátil o que significa que o seu conteúdo é perdido quando o sistema é desligado. Os principais tipos de memória RAM são:

- **Dynamic Random-Access Memory (DRAM):** este tipo de memórias possui uma maior densidade (mais células por unidade de área) e são menos dispendiosas (tecnologia mais simples), mas em contrapartida, o seu acesso é mais lento do o das memórias SRAM devido à necessidade de ciclos de "atualização";

- **Static Random-Access Memory (SRAM):** em contraste às memórias

DRAM, o acesso a estas memórias SRAM é mais rápido, a densidade é menor e são mais dispendiosas. Devido às características relativas de ambas as tecnologias de memória RAM, as DRAM são usadas na memória principal e as SRAM nas memórias *cache*.

Entradas/Saídas

As entradas e saídas são uma parte fundamental de um sistema embebido, visto que estabelecem a ligação entre o sistema computacional e o ambiente exterior, proporcionando assim, um meio de transferência de dados. Por exemplo, o sistema computacional pode interagir com o utilizador ou mesmo com outro sistema computacional através das suas entradas e saídas. Nas entradas podem ser conectados dispositivos como teclados, ratos, *keypads*, microfones e nas saídas altifalantes ou Liquid Crystal Displays (LCDs). Além destes dispositivos, existem outros que atuam como entrada e saída como as placas de rede.

A existência de entradas/saídas potencia a criação de determinados protocolos que permitem a comunicação entre processos (geralmente executados em sistemas diferentes) seguindo um conjunto específico de regras. Desta forma, é possível estabelecer um certo tipo de comunicação com o sistema embebido através dos respetivos periféricos de hardware como RS232, USB, Ethernet. Num grau de complexidade mais elevada, estes sistemas podem providenciar interfaces gráficas como Video Graphics Array (VGA) e High-Definition Multimedia Interface (HDMI).

2.1.2 Componentes de software

No topo da hierarquia das componentes de software de um sistema embebido, encontra-se a camada da aplicação que engloba a aplicação de software e as bibliotecas do domínio da aplicação. O sistema embebido é caracterizado por esta camada, uma vez que a aplicação representa o propósito pelo qual o sistema foi desenvolvido. Por baixo da camada de aplicação, encontra-se a camada do software do sistema (presente apenas em sistemas mais complexos) que envolve o sistema operativo e o programa de arranque do sistema. Esta camada funciona como uma camada de abstração do hardware que disponibiliza uma plataforma consistente para correr as aplicações. No último nível da hierarquia encontra-se a camada

de hardware. A Figura 2.6 apresenta uma arquitetura simplista de um sistema operativo.

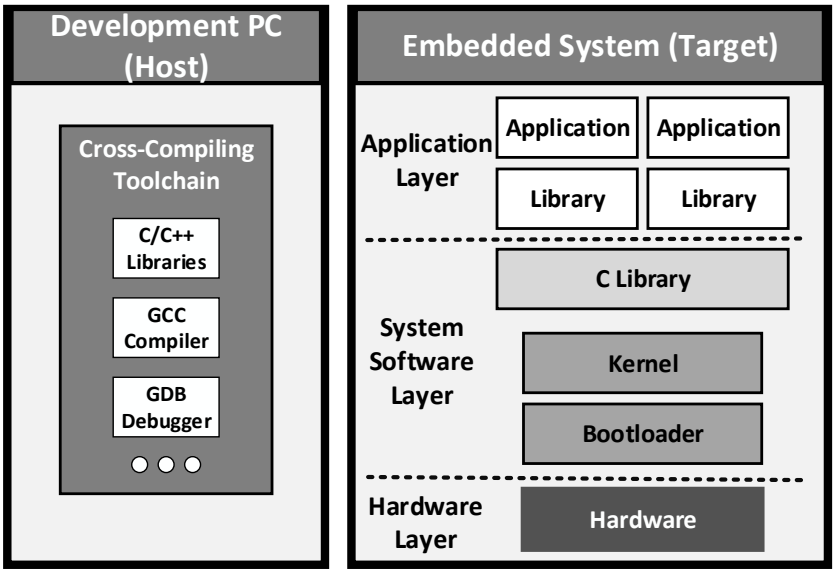


Figura 2.6: Diagrama simplificado da arquitetura de um sistema embebido.

A geração dos ficheiros necessários (e.g., *bootloader*, imagem do sistema operativo, aplicação) para a utilização de um sistema embebido completo, baseado em Linux, numa determinada plataforma de hardware é realizada num ambiente de *cross-compiling* criado na máquina *host*.

Arranque de um Sistema Embebido

No arranque do sistema embebido, vários componentes de hardware têm de ser inicializados para que o sistema esteja operacional. É importante notar que este processo é dependente do hardware e da arquitetura do CPU, uma vez cada plataforma possui um conjunto de ações e configurações pré definidas. Esta parte inicial de código faz parte do *bootloader* e é responsável pela correta inicialização dos recursos de hardware do sistema, incluindo a preparação do CPU.

Os *designers* de hardware organizam a estrutura da memória flash interna selecionando uma determinada gama de endereços, frequentemente, conforme o endereço pré definido do qual são carregadas as primeiras instruções a serem executadas. Posto isto, quando o sistema embebido arranca, o código especificado é carregado de um endereço previsível e o controlo do software pode ser estabelecido. Na Figura 2.7 está representada a estrutura típica da memória flash interna.

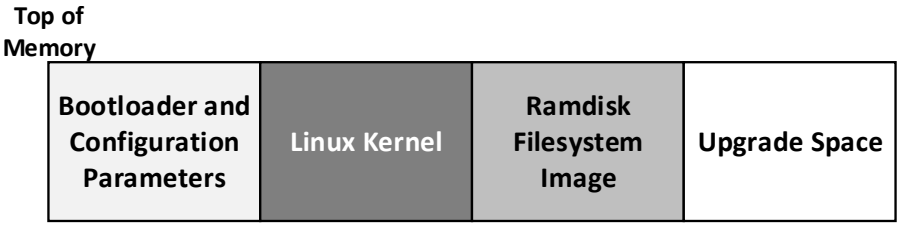


Figura 2.7: Típica estrutura da memória flash de um sistema embebido complexo.

No topo da memória flash é reservado espaço para o *bootloader* e os seus parâmetros de configuração. Em sistemas mais complexos, seguido do *bootloader*, é alocado espaço para a imagem do *kernel* e do sistema de ficheiros *ramdisk*. Ocasionalmente, estas imagens são comprimidas e descomprimidas pelo *bootloader* durante o processo de arranque. O último espaço da memória flash é dedicado a alguns dados que precisam de ser guardados entre ciclos de vida útil do sistema.

Em ambiente Linux, a imagem do sistema de ficheiros *ramdisk* armazenada na flash cria um *block device* na memória principal do sistema (RAM), sobre o qual é montada como um sistema de ficheiros. Isto qualifica-se como um boa solução caso o sistema de ficheiros seja raramente atualizado, visto que é uma tarefa difícil modificá-lo, contudo, quaisquer alterações feitas no sistema de ficheiros *ramdisk* em *run-time* são descartadas quando o sistema é desligado. Por outro lado, com o correto sistema de ficheiros (*rootfs*) armazenado na partição do elemento de armazenamento secundário (cartão de memória), o sistema operativo Linux arranca com um sistema de ficheiros temporário (*initrd*) como raíz. De seguida, monta o *rootfs*, invoca a rotina para colocar o "verdadeiro" sistema de ficheiros na raíz, removendo depois o *initrd* do sistema.

As tarefas mais importantes que o *bootloader* realiza durante o arranque são as seguintes [5]:

- Inicializa componentes de hardware críticos, como o controlador de memória, os controladores de I/O e os controladores gráficos;
- Inicializa a memória do sistema em preparação para passar o controlo para o sistema operativo;
- Aloca os recursos do sistema como a memória, serviços de interrupção aos controladores dos periféricos;
- Providencia um mecanismo para localizar e carregar a imagem do sistema operativo;

- Carrega e transfere o controlo para o sistema operativo, passando informações do arranque necessárias (tamanho total da memória, frequência de relógio, velocidades da porta série e outros dados referentes ao hardware).

Normalmente, o *bootloader* encontra-se dividido em dois estágios, o *first stage bootloader* que é responsável pela inicialização de algum hardware seguida do carregamento do *second stage bootloader* que mantém as funcionalidades do programa anterior complementando com outras como a linha de comandos. Este processo termina no segundo estágio, onde ocorre o carregamento do sistema operativo para a memória do sistema e a conclusão do processo de inicialização de hardware, seguidos da transferência de controlo do CPU para o sistema operativo. Cabe ao designer do sistema embebido configurar e compilar o *bootloader* de acordo com a arquitetura do sistema. Por isso, são descritos de seguida os principais *bootloaders open-source* utilizados em sistemas embebidos:

- **RedBoot:** é uma solução de arranque de um sistema embebido que oferece *firmware* para executar e depurar sistemas embebidos baseados em Linux numa vasta gama de arquiteturas de hardware, como ARM, MIPS, MN10300, PowerPC, Renesas SHx, e x86;

- **U-Boot:** é o *bootloader* mais utilizado no domínio dos sistemas embebidos, pelo que é compatível com várias arquiteturas de hardware, incluindo 68k, ARM, AVR32, Blackfin, MicroBlaze, MIPS, Nios, PPC e x86;

- **Barebox:** este *bootloader* deriva do U-Boot, e segue metodologias e técnicas desenvolvidas no kernel Linux. Contudo, possui um suporte de plataformas de hardware menor do que o U-Boot.

O *bootloader* é compilado num ambiente de *cross-development* de forma a ser utilizado na plataforma de hardware alvo. Na respetiva plataforma este programa é carregado para a memória flash do sistema e posteriormente executado.

Software de Sistema e Aplicação

Software encontra-se, geralmente, dividido em software de sistema e software de aplicação. Software de aplicação refere-se a todas as aplicações que realizam certas tarefas em prol do utilizador, como por exemplo, processadores de texto, videojogos. Software de sistema tem como função controlar e gerir os dispositivos e operações do sistema informático bem como disponibilizar um ambiente para

executar as aplicações, pelo que inclui o sistema operativo, as ferramentas de desenvolvimento de software (e.g., compiladores, *linkers*, *debuggers*), entre outros. Na Figura 2.8 está representada esta hierarquia de software.

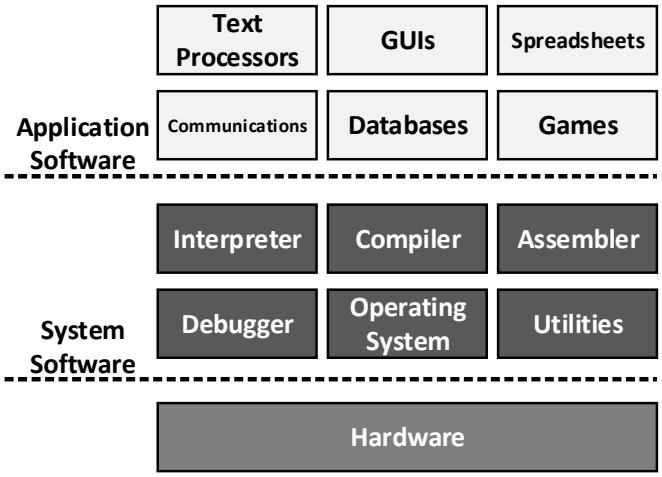


Figura 2.8: Típica hierarquia de software de um sistema embebido.

No início do desenvolvimento de sistemas informáticos, estes eram programados em código máquina o que dificultou muito o processo de desenvolvimento de sistemas mais complexos, no entanto, na altura, estes eram sistemas simples por isso era aceitável esta linguagem. Desenvolvimentos seguintes levaram à representação simbólica de código máquina através da linguagem *assembly*, que por sua vez levaram ao surgimento de linguagens de programação de alto nível similares às notações matemáticas. Estas representações mais abstratas combinadas com o respetivo software de tradução (compiladores e *assemblers*) que convertem automaticamente um programa numa sequência de instruções em código máquina, facilitaram significativamente o ciclo de desenvolvimento e aumentaram a produtividade dos programadores. Hoje em dia, a programação dos sistemas embebidos é feita através de notações de alto nível que possuem um grande poder de expressão em domínios específicos.

2.1.3 Conclusões

Um sistema embebido pode ser definido como um sistema computacional dedicado à realização de um conjunto específico de tarefas, como por exemplo em máquinas de lavar, sistemas de navegação e alarmes de segurança. A maioria destes sistemas embebidos possuem requisitos temporais críticos, cujos resultados são relevantes se ocorrerem dentro de uma específica *time frame*. Nos processos de controlo e

monitorização na indústria moderna são utilizados inúmeros sistemas embebidos com requisitos de tempo-real (e.g., em transportes, estações nucleares, hospitais) que têm de ser cumpridos de forma a não comprometer o sistema ou a vida do utilizador.

Sistemas embebidos possuem um papel importante no quotidiano, mesmo que a sua presença não seja notada, como no controlo do menu da televisão, do temporizador do microondas, do telemóveis, entre muitos outros. Esta é uma indústria em constante evolução onde as oportunidades de crescimento são numerosas.

2.2 Linux e Sistemas Embebidos

O movimento de software livre começou em 1984 quando Richard Stallman deixou o Massachusetts Institute of Technology (MIT) e fundou o projeto GNU com o intuito de libertar o conhecimento dos monopólios das empresas privadas. Por isso, em 1985, Stallman fundou a Free Software Foundation e escreveu a GNU General Public License (GPL) que permite aos utilizadores usar, estudar, partilhar e modificar todo o software que se encontre sobre esta licença. Nos pontos seguintes estão as principais características da GPL:

- É uma licença *copyleft* devido ao facto que todo o software derivado tem de ser distribuído sobre os mesmos termos desta licença;
- Garante liberdade ao utilizador, na medida em que pode executar, estudar, partilhar e modificar o código fonte do software;
- Garante os mesmos direitos a quem (indivíduos, organizações, empresas) o software é partilhado.

Em 1991, Linus Torvalds, um estudante da Universidade de Helsínquia na Finlândia preenche a lacuna existente no projeto GNU criando um simples, estável, *open-source kernel* baseado parcialmente no sistema operativo Unix e licenciado sobre os termos da GPL, denominado Linux. Subsequentemente, vários entusiastas tiraram proveito do crescimento da Internet nos anos 90 para criarem comunidades *online*, cujo objetivo era o desenvolvimento contínuo do *kernel* Linux. Graças a estas comunidades científicas, foram criadas as primeiras distribuições de Linux com todas as componentes de software necessárias para a instalação e utilização do sistema operativo Linux. Existem diversas razões pelas quais se deve a popularidade

do Linux, como por exemplo:

- **Modularidade e Estrutura:** cada funcionalidade é dividida em módulos distintos, possuindo a capacidade de adaptar o sistema às necessidades da aplicação e garantindo também estabilidade no caso da ocorrência de falhas não comprometendo o sistema integralmente;
- **Legibilidade:** o código fonte encontra-se legível, facilitando, assim, a compreensão do *kernel* e consequentemente as modificações no seu código;
- **Extensibilidade:** o processo de adicionar de novas funcionalidades é simples e se for pretendido realizar algumas modificações no código, estas são facilmente identificadas;
- **Recuperação de Erros:** na ocorrência de uma situação problemática, o software realiza as ações necessárias para recuperar da condição de falha notificando o sistema operativo, registando o erro e alertando o utilizador;
- **Longevidade:** o software corre sobre o Linux por longos períodos de tempo sem assistência, preservando a integridade do sistema.

A maioria dos programador chegam a um entendimento comum no que se relaciona ao facto do Linux encaixar nas suas descrições de qualidade e fiabilidade. Graças ao seu modelo de desenvolvimento *open-source*, este junta uma grande comunidade científica que contribui para este projeto, identificando problemas, discutindo soluções e corrigindo problemas, o que leva a um aumento na utilização de soluções ótimas.

Devido aos seus benefícios tecnológicos, a adoção dos Linux pelos sistemas embebidos tem vindo a aumentar em vários mercados e tecnologias. Este sistema operativo tem sido adotado por sistemas embebidos utilizados nas mais diversas aplicações como em automóveis, na qualidade da energia elétrica, em produtos orientados ao consumidor, entre muitas outras. De seguida, são apresentadas as principais características que distinguem o Linux dos outros sistemas operativos:

- É um sistema escalável, desde aplicações mais simples orientados ao consumidor até sistemas com um grau de complexidade maior;
- Suporta uma vasta variedade de plataformas de hardware, o que significa que o Linux pode ser executado em várias arquiteturas de hardware;
- Oferece uma grande quantidade de *device drivers*;

- Suporta vários protocolos de comunicação (e.g., Controller Area Network (CAN), USB, Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C)) facilitando a comunicação com outros sistemas computacionais ou outros dispositivos;
- Potencia a portabilidade das aplicações e consequente reutilização, devido ao seu vasto suporte de arquiteturas de hardware;
- Promove a integração de bibliotecas no domínio da aplicação;
- Possui uma grande comunidade ativa que contribui para aceleração de suporte de novas arquiteturas, plataformas e dispositivos.

Todas estas características fazem do Linux, o sistema operativo mais utilizado pelos sistemas embebidos atualmente.

2.2.1 *Kernel* Linux

Como ilustrado na Figura 2.9, acima da camada de hardware encontra-se o principal elemento de sistema operativo, chamado de *kernel*. Este elemento é responsável por controlar todos os recursos de hardware do sistema (e.g., o CPU, unidade de memória e I/O) bem como providenciar um conjunto de abstrações de alto nível independentes da arquitetura e do hardware (como as Application Programming Interfaces (APIs) Portable Operating System Interface (POSIX)) que permitam aos processos em modo de utilizador interagir com os recursos do sistema. Para além disto, o *kernel* Linux gere o acesso às entradas/saídas, controla o escalonamento de processos e é responsável por outras tarefas administrativas.

Existem dois conjuntos distintos de serviços dentro do *kernel* Linux que disponibilizam as funcionalidades requeridas pelas aplicações. As interfaces de baixo nível lidam com operações específicas do CPU, operações de memória e proporcionam uma API que controla o hardware do sistema. Acima destes serviços, encontram-se as interfaces de alto nível que disponibilizam certas abstrações que gerem processos, ficheiros, *sockets* e sinais.

Normalmente, no arranque do sistema, o *kernel* inicializa todos os dispositivos do sistema (como controladores de rede, dispositivos I/O) e os *handlers* de serviço à interrupção que respondem aos sinais gerados pelos periféricos. De seguida, são inicializadas as interrupções de software desencadeadas para processarem as

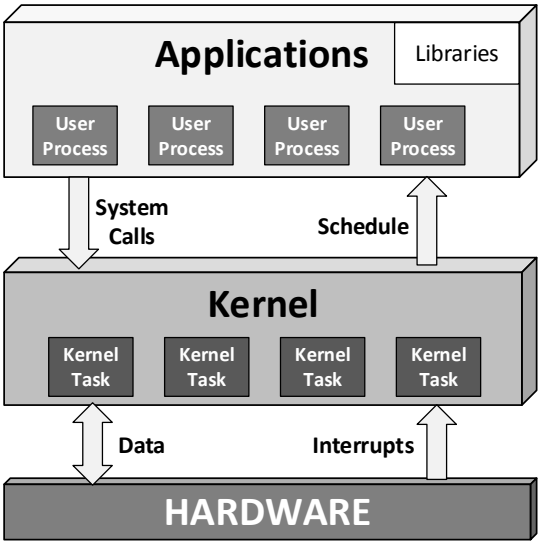


Figura 2.9: Diagrama simplificado da interação entre os elementos de um sistema embebido.

chamadas ao sistema lançadas pelos processos no espaço da aplicação.

2.2.2 *Kernel* Linux em Sistemas Embebidos

Ao contrário de sistemas *desktop*, um sistema embebido, na maioria dos casos, não usufrui de todos os serviços oferecidos pelo *kernel* Linux devido à sua especificidade num conjunto de tarefas. Desta forma, o *kernel* é customizado para satisfazer apenas as necessidades específicas do sistema removendo as funcionalidades não requeridas. Para além da parametrização do *kernel* consoante o hardware do sistema, podem ser adicionadas/removidas certas aplicações e bibliotecas extra e, graças à sua natureza *open-source*, podem ser modificadas de acordo com os requisitos do sistema. Nas secções seguintes é feita uma breve descrição sobre as principais características dos serviços do sistema operativo Linux: gestão de processos, gestão de memória, sistema de ficheiros e interfaces de hardware.

Gestão de Processos

O subsistema de gestão de processos é parte integral de qualquer *kernel*, e tem como objetivo alocar os recursos para os processos, habilitar processos para a troca de informação através de mecanismos Inter Process Communication (IPC), proteger os recursos de outros processos, habilitar a sincronização entre processos e

escalonar os processos. Para a identificação e diferenciação dos processos, o *kernel* atribui uma estrutura de dados a cada processo permitindo o controlo independente de cada processo.

Cada processo possui o seu próprio espaço de memória e proporciona os recursos necessários para executar um programa. Estas entidades são iniciadas com uma única *thread*, podendo ser criadas *threads* adicionais subsequentemente. Por sua vez, *thread* é uma entidade de um processo que pode ser escalonada para execução e possui um único fluxo de execução. Todas as *threads* de um processo partilham o mesmo espaço de memória.

Em *user space*, o *kernel* é responsável pelo escalonamento das *threads* através da atribuição de um certo período de tempo do CPU e/ou de prioridades. Caso um processo, o qual corre sempre em *user space*, precise de usar um recurso de hardware, executar ou criar novos processos ou comunicar com os subsistemas do *kernel*, este tem de lançar uma chamada ao sistema de maneira que o *kernel* concretize o pedido efetuado.

Gestão de Memória

A gestão de memória é um subsistema do sistema operativo que está encarregue da alocação de memória quando requisitada por um processo e pela sua libertação quando termina a execução do respetivo processo ou quando deixa de ser necessária. Este subsistema mantém guardado o estado de cada posição de memória, determinando assim como a memória é alocada entre processos.

A memória total disponível de um sistema engloba a memória *cache* (memória de rápido acesso e de menor tamanho que guarda dados frequentemente usados da memória principal), a memória principal (memória RAM que guarda os dados e instruções dos processos em execução) e a memória de armazenamento secundário (memória não-volátil usada pelo *kernel* no mecanismo de paginação e usada como elemento complementar de armazenamento).

O sistema operativo Linux virtualiza o acesso à memória física do sistema, permitindo aos processos acederem a uma gama de endereços pré-definida consoante a arquitetura do sistema. Uma vez que esta gama de endereços virtuais não reflete a memória física do sistema, o sistema operativo usa um conjunto de tabelas que traduzem o endereço virtual para endereço físico para que o hardware do sistema responda corretamente aos pedidos realizados.

O espaço de endereços virtuais bem como o espaço de endereços físicos são organizados através de um conceito de paginação, onde cada espaço de memória é divido em páginas com tamanhos definidos pela arquitetura do sistema. Cada página pode ter dois estados: estado válido quando está mapeada na memória principal do sistema; ou estado inválido quando está mapeada numa unidade secundária de memória do sistema. Ao longo do execução do sistema operativo, o kernel está constantemente a trocar páginas da memória RAM, cujo acesso é menos frequente, com a unidade de armazenamento secundária, contribuindo para um melhoramento na performance global do sistema. No entanto, quando uma página presente na unidade secundária de memória precisa de ser acedida por um processo, a Memory Management Unit (MMU) despoleta uma *page fault*, notifica o *kernel* que a página não está na memória principal, e carrega-a (*swapped*) de volta para a memória RAM.

As vantagens deste mecanismo compreendem a facilidade de acesso à memória partilhada, maior proteção da memória, controlo da *cache*, e o uso de uma maior gama de endereços de memória através do constante *swapping* de páginas da/para a memória de armazenamento secundário.

O *kernel* usa o CPU e a MMU para mover transparentemente as instruções e dados entre as várias entidades de memória discutidas. O acesso a estas memórias segue uma hierarquia cujos níveis são distinguidos pelo seu tempo de resposta: primeiramente encontra-se a memória *cache*, seguida da memória RAM e finalizando nas memórias de armazenamento secundário.

Sistema de Ficheiros

O sistema de ficheiros define a estrutura e regras utilizadas para gerir a informação presente no sistema, sem o qual tornaria a informação num bloco de dados indistinguíveis. Por isso, este subsistema atribuindo as respetivas informações a cada bloco de dados, torna a informação organizada e legível.

Em sistemas operativos Unix, todos ou quase todos os recursos de hardware são tratados com ficheiros, identificados por um nome. Além desta abstração proporcionada pelo Linux, existe um subsistema do *kernel*, o Virtual File System (VFS), que interage com os recursos do sistema em modo *kernel* fornecendo uma camada de abstração com a mesma interface para todos os ficheiros (incluindo *device drivers*) que permite a gestão dos mesmos recursos.

Interfaces de Hardware

O sistema operativo Linux possui certos componentes de software ao nível do *kernel*, denominados de *device drivers*, os quais constituem uma interface com os dispositivos I/O escondendo os detalhes da implementação do utilizador, acerca do acesso/controlo de estruturas de dados do *kernel* e dos recursos de hardware. As rotinas destes *drivers* podem ser lançadas pelo *kernel* quando requeridas ou como resposta a algum evento, como uma interrupção de hardware.

Principalmente, um *device driver* possui as seguintes características:

- Fornece um conjunto especifico de rotinas para comunicar com os respetivos recursos de hardware/software;
- É um componente modular que pode ser carregado/removido do sistema operativo dinamicamente;
- Gere o fluxo de dados e controlo entre os processos em *user space* e recursos do sistema.

Além disto, no contexto de gestão de memória pode ser implementado um mecanismo de mapeamento de memória que permite o acesso direto dos processos em *user space* à memória do *device* (periférico ou estrutura de dados do *kernel*). Este mecanismo consiste em associar uma gama de endereços no espaço virtual da aplicação à memória do *device*, de forma a que a escrita/leitura dos processos no espaço do utilizador se traduzam na memória do *device*. Contudo, a gestão de memória é feita ao nível de páginas, pelo que o tamanho da área mapeada tem de ser um múltiplo do tamanho da página do sistema.

2.3 Linux e Tempo-Real

A maioria dos sistemas embebidos baseados em Linux possuem requisitos difíceis de satisfazer num ambiente de sistema operativo, como determinismo, baixas latências e previsibilidade. Estes requisitos temporais se não forem cumpridos podem comprometer a integridade do sistema ou ainda a vida do utilizador.

O sistema operativo Linux foi originalmente desenhado sem a característica de tempo-real, mas sob um conceito de atribuição de *time slices* a cada processo. Este tipo de sistemas são caracterizados pela sua capacidade de processar eventos

numa forma temporal, obtendo a melhor performance e o maior *throughput* através gestão dos recursos.

Os sistemas de tempo-real podem ser classificados em duas categorias consoante as consequências em caso de falha:

- ***Hard real-time*** é caracterizado pelos resultados no incumprimento de uma *deadline*. Neste tipo de sistemas, se uma *deadline* não é cumprida os resultados comprometem a integridade do utilizador, como no sistema de controlo da injeção de combustível numa nave espacial;

- Por outro lado, em sistemas ***soft real-time***, a falha origina a diminuição da qualidade de experiência do sistema, não causando resultados catastróficos. Por exemplo, em *streaming* de áudio e vídeo os dados podem apresentar algum atraso, contudo não comprometem o sistema.

Para usar o sistema operativo Linux em aplicações com requisitos tempo-real, é necessário realizar algumas modificações que tornem o sistema capaz de reagir rapidamente face a um evento externo, com baixas latências e completamente preemptivo.

2.3.1 Escalonamento em Linux

O escalonador do Linux é um escalonador que se baseia em prioridades, determinando assim quais os processos a serem executados pelo CPU, de forma que cada processo progrida. Cada vez que o escalonador é lançado, este analisa cada processo na lista de execução e baseando-se na prioridade concebe-lhes o CPU. No entanto, isto vai contra o conceito de tempo-real, visto que tempo-real significa ter um sistema determinístico, previsível e com baixas latências, o que pode envolver sacrifícios na performance global do sistema.

2.3.2 Latência e *Jitter*

Um sistema operativo de tempo-real é definido pela capacidade de resposta do sistema a atender eventos internos ou externos. Nestes eventos podem ser incluídos interrupções externas por hardware, interrupções por software e *timers*. Esta capacidade de resposta pode ser medida através da latência, a qual dita o tempo

que leva entre a ocorrência do evento até à execução da primeira instrução do respetivo processo de atendimento. Outra forma pela qual a capacidade de resposta pode ser medida é o *jitter* que consiste na variação da periodicidade verdadeira de um sinal supostamente periódico. Para reduzir a latência e o *jitter*, e consequentemente melhorar o tempo de resposta, o sistema operativo tem de assegurar que os processos de tempo-real possam interromper a execução todos os outros (preempção). Na Figura 2.10 estão representadas as latências desde a ocorrência de um evento até à execução do respetivo processo de tempo-real.

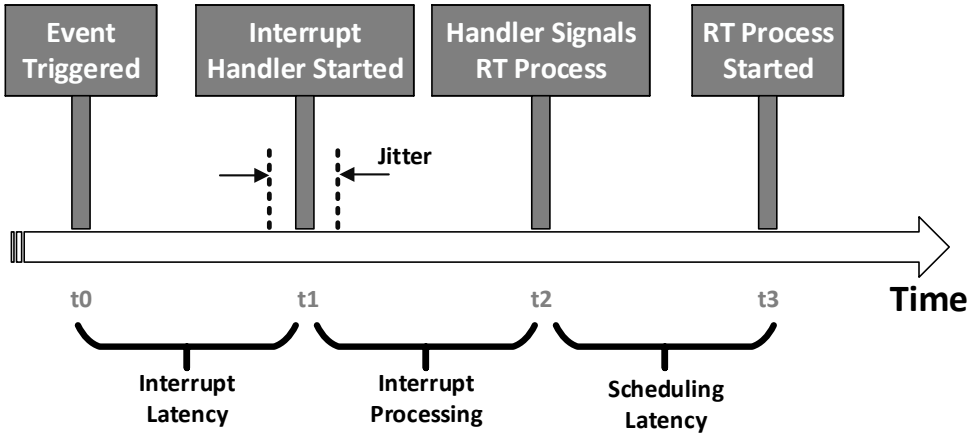


Figura 2.10: Latências presentes no Linux desde a ocorrência de um evento até à execução do respetivo processo de tempo-real.

O processo de medida da latência começa aquando a ocorrência de um certo evento interno ou externo, cujo instante é indicado por `t0` na Figure 2.10. Neste instante, o CPU é notificado pelo sinal de interrupção e lança o respetivo *handler* no instante `t1`. A duração da execução do *handler* de serviço à interrupção tem de ser mínima, delegando o resto da resposta ao evento para outra entidade, de forma a não manter as interrupções desabilitadas por um longo período de tempo. Depois de realizadas as ações de resposta ao evento, o respetivo processo de tempo-real é acordado no instante `t2`. De seguida, o escalonador transfere o controlo do CPU para esse processo, instante `t3`, cuja latência é afetada pelo número de processos em espera e pelas usas prioridades.

2.3.3 Preempção no *Kernel*

De forma a satisfazer os requisitos temporais críticos da aplicação, o *kernel* tem de ser capaz de mudar de contexto de execução o mais rápido possível assim que uma tarefa de prioridade maior do que a prioridade da tarefa a correr esteja pronta para

execução. Isto é possível tornando o *kernel* completamente preemptivo. Com esta funcionalidade, o kernel é capaz de interromper a execução de qualquer processo em prol da execução de um novo processo com prioridade maior. Na Figura 2.11 está representada a respetiva sequência de eventos.

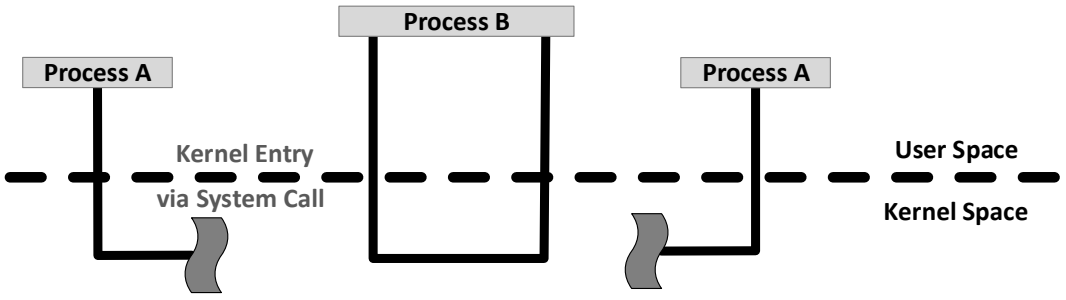


Figura 2.11: Exemplo de preempção no *kernel* Linux.

Em relação à Figura 2.11, inicialmente, o `Process A` encontra-se em execução e lançou uma chamada ao sistema. Enquanto o `Process A` está a ser executado em modo *kernel*, o `Process B` de maior prioridade é acordado por um certo evento. Posto isto, o escalonador verifica que se encontra um processo de maior prioridade pronto para ser executado e interrompe a execução do processo de menor prioridade, atribuindo o CPU ao `Process B`. De salientar que, o processo de preempção ocorreria mesmo que o `Process A` não estivesse concluído o seu processamento ou se encontrasse suspenso.

Modelos de Preempção

Originalmente, o *kernel* Linux era fundamentalmente não-preemptivo, mas à medida que a exigência dos requisitos das aplicações foi aumentando, a necessidade por uma solução em termos das funcionalidade de preempção foi também crescendo. Daí adveio o desenvolvimento do modelo de preempção atual usado no sistema operativo Linux. Este modelo tornou o *kernel* parcialmente preemptivo, pelo que esta funcionalidade é desativada em secções críticas de código onde é necessário proteger estruturas de dados de serem corrompidas por outros processos.

Na versão 4.04 do *kernel* Linux (versão utilizada nesta dissertação) existem três modelos de preempção:

- `PREEMPT_NONE`: *No forced preemption (server)* - Em média, a latência global é aceitável, mas pode sofrer atrasos significativos, o que torna-a na solução ideal para maximizar o *throughput* global de um sistema preemptivo;

- `PREEMPT_VOLUNTARY`: *Voluntary kernel preemption (desktop)* - Neste modelo, o *throughput* global é sacrificado por latências mais baixas, através da inserção de pontos de preempção estratégicos no *kernel*;
- `PREEMPT_DESKTOP`: *Preemptible kernel (low-latency desktop)* - Com este modelo habilitado, o *kernel* torna-se completamente *preemptible*, exceto no processamento de secções de código críticas. Como expectado, a latência global diminui à custa do *throughput* global do sistema.

Cabe ao *designer* do sistema avaliar as funcionalidades de cada modelo de preempção e escolher a solução que melhor se adapta à sua aplicação. No caso de nenhum modelo satisfazer os requisitos temporais, a solução passa pelo *patching* do *kernel* Linux, como é descrito de seguida.

Real-Time Kernel Patch

Até à data da escrita desta dissertação, a versão *mainline* do *kernel* Linux ainda não possui o suporte para sistemas do tipo *hard real-time*, por isso é necessário recorrer a modificações no código fonte do *kernel* através de *patches*. A *patch real-time* é o resultado cumulativo de vários programadores, cujos objetivos eram melhorar o determinismo, previsibilidade e latências do *kernel* Linux. Quando o *patch* é aplicado, são adicionadas várias funcionalidades que podem ser acedidas através do menu de configuração do *kernel*.

Esta *patch real-time* adiciona um modelo de preempção extra, `PREEMPT_RT_FULL`, que habilita a preempção em cada ponto no *kernel* exceto em áreas onde a função *preempt_disable()* é chamada. Através desta funcionalidade, o *kernel* Linux torna-se capaz de lidar com aplicações com requisitos temporais exigentes, conferindo um nível maior de previsibilidade e determinismo ao *kernel*. No Anexo A é descrito este procedimento.

Funcionalidades do *Kernel* Tempo-Real

Quando o modelo de preempção tempo-real é habilitado, várias modificações são aplicadas ao código fonte do *kernel*, das quais se destaca a substituição de *spinlocks* por uma variante baseada em prioridades que pode ser interrompida, designada de *mutex*. Com esta modificação nos mecanismos de exclusão mútua, a latência global

do sistema é reduzida à custa do aumento do *overhead* associado ao processamento destes mecanismos, reduzindo o *throughput* global do sistema.

Assim, o *handler* de serviço à interrupção fica apenas responsável pelo reconhecimento do sinal de interrupção, pelo que o restante trabalho é realizado É também importante salientar, que a *patch real-time* transforma todas os *handlers* de serviço à interrupção em entidades escalonáveis com as suas próprias prioridades podendo, assim, serem manipuladas pelo programador como qualquer outra *thread* no sistema. Deste modo, as latências associas ao tempo que as interrupções se encontram desativadas são reduzidas.

Desta forma, o *patching* do *kernel* Linux origina uma performance inferior, mas em contrapartida, confere uma maior previsibilidade e redução das latências do sistema. Logo, cabe ao programador escolher o nível de preempção que pretende no *kernel* Linux.

2.4 Tecnologia Field-Programmable Gate Array

Field-Programmable Gate Arrays são circuitos integrados de silício que podem ser reprogramados para realizarem várias tarefas sem a complexidade inerente ao desenhar circuitos eletrónicos. Esta categoria de hardware reprogramável tem sido desenvolvida ao longo dos anos, pelo que o melhoramento da tecnologia dos semicondutores potenciou o aumento da capacidade e funcionalidades destes dispositivos modernos. Consequentemente, a tecnologia FPGA chegou a um ponto onde suporta processadores, barramentos, controladores de memória e uma vasta quantidade de blocos dedicados em hardware como somadores/subtratores, multiplicadores e blocos FFT.

A adoção dos modernos dispositivos com tecnologia FPGA, surgiu devido à crescente exigência nos requisitos das aplicações de complexidade elevada. Deste modo, as aplicações podem explorar o paralelismo e flexibilidade proporcionados pela tecnologia FPGA com o intuito de obter uma melhor performance, determinismo, previsibilidade e baixas latências exigidos por sistemas de tempo-real.

Assim, o programador do sistema embebido deve seguir certas metodologias de desenvolvimento de forma a acelerar a sua produtividade e de reduzir o *time-to-market*. Nas próximas subsecções, são discutidos a arquitetura de uma FPGA, as linguagens empregues na programação de hardware e um típico *design flow*

utilizado.

2.4.1 Arquitetura de FPGA

O *fabric* dos dispositivos FPGA consiste num vasto número de blocos de lógica programável em conjunto com uma hierarquia de interconexões programáveis que fazem a ligação entre os blocos lógicos, como ilustrado na Figura 2.12.

Cada bloco lógico programável fornece os recursos lógicos básicos (Look-Up Tables (LUTs), lógica combinacional e registos na forma de *flip-flops*) que combinados com as interconexões podem formar elementos funcionais lógicos. Adicionalmente, estes dispositivos possuem blocos de memória RAM, blocos dedicados (e.g., somadores, multiplicadores) de hardware embebidos no *fabric* da FPGA e, na vanguarda desta tecnologia, têm a capacidade para criar um microprocessador através de blocos de lógica programável ou possuem blocos dedicados *hardwired* (*hard cores*) que funcionam como tal.

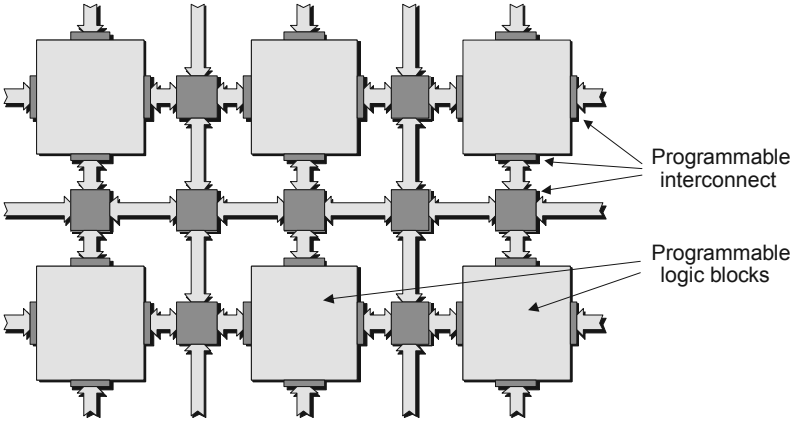


Figura 2.12: *Fabric* da FPGA [7].

Atualmente, os principais fabricantes de dispositivos FPGA são a Xilinx e a Altera, pelo que cada um possui a sua própria designação de cada elemento lógico. Nesta dissertação são utilizadas as nomenclaturas da Xilinx devido à utilização das suas ferramentas e plataformas de hardware.

2.4.2 Linguagens de Descrição de Hardware

As linguagens de descrição de hardware foram criadas para descrever o comportamento do hardware na tecnologia FPGA. Estas linguagens de programação con-

sistem em modelar e descrever a estrutura de circuitos lógicos através de notações textuais, permitindo também a análise, simulação e teste dos elementos funcionais lógicos criados.

Verilog e VHSIC Hardware Description Language (VHDL) são as linguagens de descrição de hardware mais utilizadas na síntese de lógica e simulações. Na comunidade das Hardware Description Languages (HDLs) ainda é discutível qual é a melhor solução na descrição do comportamento de hardware, contudo existe consenso na facilidade de uso de Verilog e nas potencialidades do VHDL.

As linguagens de descrição de hardware suportam diferentes níveis de abstração, representados na Figura 2.13, incluindo a descrição mais detalhada ao nível de portas lógicas, as representações funcionais através da transferência de dados entre registos (Register Transfer Level (RTL)), e a descrição do comportamento de hardware através de algoritmos similares à linguagem de programação C.

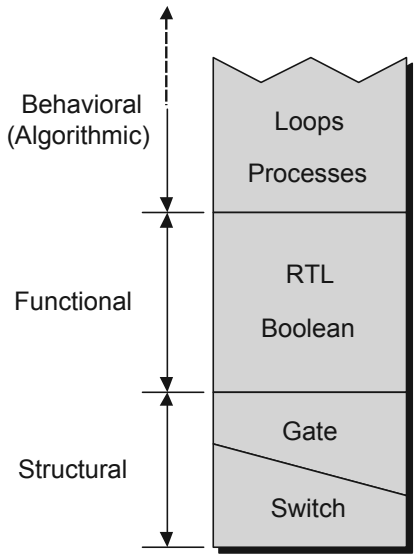


Figura 2.13: Diferentes níveis de abstração das linguagens de descrição de hardware [7].

Estas linguagens de descrição de hardware permitem a combinação de níveis de abstração com o objetivo de melhorar a performance do sistema.

2.4.3 FPGA *Design Flow*

O *design flow* é um processo iterativo que envolve desenhar, implementar e verificar o comportamento do hardware até alcançar um ponto que satisfaz os requisitos es-

tipulados. Um *design flow* de FPGA geralmente adotado encontra-se representado na Figura 2.14.

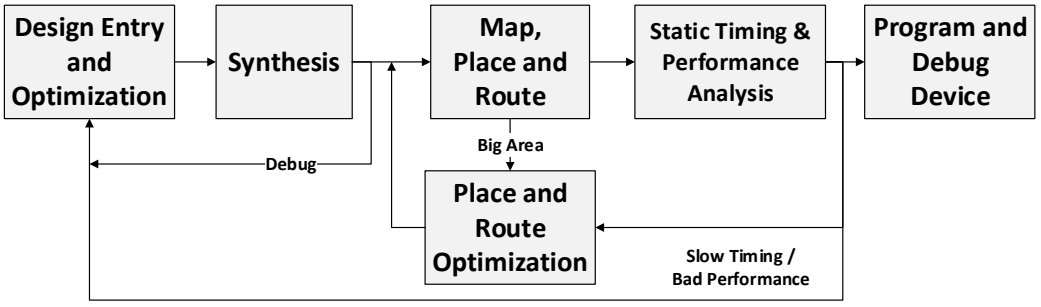


Figura 2.14: *Design flow* normalmente adotado em FPGAs.

Na primeira fase, é criado e posteriormente otimizado o design dos elementos lógicos funcionais através de ferramentas disponibilizadas pelos forncedores Electronic Design Automation (EDA). Durante a síntese, a informação comportamental no ficheiro HDL é traduzido numa *netlist* (representa as conetividades dos blocos lógicos) e o *design* é otimizado para a plataforma selecionada.

A implementação começa com o mapeamento da lógica sintetizada na plataforma especificada, onde os componentes lógicos dedicados são inferidos. Esta fase termina quando o design sintetizado é mapeado com sucesso na plataforma alvo. Na última fase do *design flow*, é gerado o bitstream que especifica toda a informação dos blocos de lógica programável e as respetivas interconexões, usados para programar a FPGA.

Além disso, ao longo do *design flow*, é possível realizar simulações a diferentes níveis de modo a verificar e validar as funcionalidades do sistema: simulação RTL, simulação funcional pós-síntese e pós-implementação. No caso de não se verificarem, terão de ser feitas otimizações ao nível do design ou mapeamento até alcançar os resultados esperados.

2.5 Sistemas de Aquisição de Dados

Os sistemas de aquisição são responsáveis pela ligação entre o ambiente exterior (mundo analógico) e os sistemas digitais, geralmente apresentados na forma de sistemas embebidos, no controlo e monitorização de processos industriais. Estes sistemas integram as potencialidades e flexibilidade de um sistema embebido, pelo

que a sua configuração deve ser feita através da combinação dos requisitos da aplicação com o hardware e software apropriado, de modo a ser utilizado eficazmente. Os elementos básicos de um sistema de aquisição são:

- **Transdutores** - quantificam a grandeza física a medir num sinal elétrico;
- **Acondicionamento de sinal** - trata o sinal proveniente do mundo exterior de acordo com as características e especificações do hardware de aquisição de sinal;
- **hardware de aquisição de sinal** - é o *core* do sistema pois faz a interface entre o mundo analógico com o digital através da utilização de conversores analógico-digitais. Para além disto, pode executar vários procedimentos, como a atuação direta através dos sinais analógicos gerados, o envio dos dados para um PC onde está a aplicação, o processamento dos dados por parte de um sistema embebido e posterior atuação através de conversores digital-analógicos;
- **Aplicação de monitorização/controlo** - é responsável pela execução das tarefas no ambiente em que está inserida, de acordo com as especificações e objetivos estipulados.

A empresa de referência no domínio de sistemas de aquisição para fins académicos ou industriais é a National Instruments pelo que possui um maior número e variedade de oferta deste tipo de sistemas. A gama de produtos que a NI oferece explora as potencialidades de uma ferramenta de programação gráfica, o LabView, que permite desenvolver sistemas com maior rapidez, automatizar várias medições e tomar decisões baseadas nos dados adquiridos.

2.5.1 Transdutores

A aquisição de dados começa com a quantificação de um fenómeno físico ou elétrico como, a temperatura de uma sala, a intensidade luminosa de uma luz ou a força aplicada a um objeto. Um transdutor converte um certo fenómeno num sinal elétrico mensurável na forma de tensão, corrente, resistência ou noutra grandeza elétrica que varia conforme o fenómeno exterior. Desde modo, estes dispositivos permitem a análise de certas grandezas ao longo do tempo.

2.5.2 Acondicionamento de Sinal

Na maioria dos casos, o sinal de saída proveniente dos transdutores é muito difícil (e.g., presença ambientes ruidosos) ou muito perigoso (e.g., rede elétrica) de medir diretamente com um dispositivo de aquisição, podendo resultar em leituras imprecisas e potencialmente perigosas. Por isso, de maneira a garantir precisão, exatidão, e segurança são necessários os circuitos de acondicionamento de sinal. Estes circuitos passam principalmente, pela amplificação, atenuação, filtragem, isolação e excitação com o intuito de tratar o sinal de saída dos transdutores no formato adequado para o hardware de aquisição de sinal.

2.5.3 Hardware de Aquisição de Sinal

O hardware de aquisição de sinal é o *core* do sistema, uma vez que funciona como meio de comunicação entre o sistema computacional (mundo digital) e o ambiente exterior (mundo analógico). Normalmente, este *core* é constituído por um Analog-to-Digital Converter (ADC), para converter a grandeza física medida no formato digita cuja informação é processada posteriormente pela plataforma de desenvolvimento, contudo pode realizar o seguinte conjunto de funções:

- Os sinais analógicos de entrada são convertidos em sinais digitais usando ADCs;
- Os dados de entrada são transferidos para um sistema embebido para serem analisados e processados;
- Os dados de entrada são enviados para um PC, onde são utilizados em aplicações de monitorização/controlo;
- Os dados de entrada são processados e posteriormente convertido de volta em sinais analógicos, usando Digital-to-Analog Converters (DACs), para atuarem no ambiente exterior;
- Os dados de entrada digitais, que contêm informação sobre um certo dispositivo, são usados diretamente para efeitos de controlo.

O hardware de aquisição de sinal encontra-se disponível em várias formas sobre diversos fabricantes, podendo oferecer vários níveis de funcionalidade como o número de canais, taxa de amostragem, resolução, precisão e custo.

2.5.4 Componente de Software

A chave para tirar partido das potencialidades de um sistema embebido de forma eficaz, encontra-se no *co-design* de hardware-software de acordo os requisitos específicos da aplicação. Assim, a componente de software possui um papel fundamental no controlo do hardware de aquisição de sinal de forma a realizar as tarefas alvo corretamente. Esta função passa por desenhar os respetivos *device drivers* do respetivo hardware utilizado bem como uma fornecer um conjunto de serviços, com um sistema operativo de suporte, que permitam a execução de determinadas tarefas.

Para além disto, esta componente engloba a aplicação de monitorização/controlo que, normalmente, realiza tarefas como:

- Monitorização em tempo-real;
- Análise de dados;
- Algoritmos de controlo;
- Human Machine Interface (HMI).

Normalmente, a aplicação de software corre sob um sistema operativo num PC *host*, garantindo flexibilidade visto que mantendo o protocolo de comunicação é possível alterar o hardware de aquisição de sinal sem a necessidade de modificar a aplicação. Outra possibilidade consiste em integrar a aplicação na plataforma, contudo requer um maior esforço de engenharia devido à variabilidade de hardware e da quantidade de ferramentas de software existentes.

Capítulo 3

Plataformas de Desenvolvimento

Neste capítulo são discutidas as plataformas de hardware usadas nesta dissertação, descrevendo as suas principais características face ao sistema proposto. As plataformas híbridas utilizadas consistem num SoC customizável que integra o sistema de processamento com dois *hard-cores* da arquitetura ARM e uma área de silício programável de tecnologia FPGA num único chip. Como discutido anteriormente, a utilização de uma plataforma com tecnologia FPGA permite o aumento da performance, do determinismo, de reduzir as latências da aplicação e oferece um nível elevado de flexibilidade de forma a satisfazer os requisitos exigentes e complexos dos sistemas embebidos. As plataformas de hardware descritas requerem um ambiente de desenvolvimento embebido tendo como alvo dispositivos da família Zynq-7000 da Xilinx, por isso foram utilizadas as ferramentas de software deste fabricante. No final deste capítulo são apresentados os critérios utilizados que levaram à escolha da plataforma de hardware como o sistema embebido alvo.

3.1 Plataforma ZedBoard Zynq-7000

A plataforma de desenvolvimento ZedBoard, ilustrada na Figura 3.1, oferece um grande número de componentes de hardware usados para criar um SoC customizado suportado pelo *kernel* Linux, que incluem:

- **Zynq-7000 XC7Z020-1CSG484:** FPGA+CPU SoC que combina dois processadores ARM Cortex-A9 MPcore na unidade de processamento e uma área programável da tecnologia de FPGA 7-Series;

- **Memória Dual Data Rate (DDR) SDRAM:** Duas memórias SDRAM DDR3 de 256 MB responsáveis pela execução das aplicações com um sistema operativo de suporte ou de aplicações *bare-metal*;
- **Interface para Cartão de Memória SD:** Esta interface permite a utilização de um elemento de memória de armazenamento secundário que pode ser utilizado na programação da FPGA, no arranque o *kernel* Linux, para montar o sistema de ficheiros de Linux e também pode ser utilizado como memória complementar do sistema;
- **USB-to-UART:** Este periférico permite a conexão com um PC *host* para efeitos de comunicação;
- **USB-JTAG:** É completamente suportado e integrado nas ferramentas da Xilinx, incluindo as ferramentas para a programação e de depuração do sistema;
- **USB-OTG:** Permite a conexão com dispositivos USB externos, como USB flash drives, câmaras digitais, ratos ou teclados;
- **Fontes de Relógio:** Oscilador *on-board* de 100 MHz fornece o sinal de relógio da área programável e o sinal de relógio da unidade de processamento gerado por um oscilador de uma frequência fixa de 33.33 Mhz;
- **Interfaces Gráficas:** Incluem uma interface HDMI que suporta formatos de vídeo até uma resolução de 1080p a uma taxa de 225 MHz, e uma saída de vídeo de 12-bits através do conetor VGA;
- **Entre Outros:** Adicionalmente, esta plataforma possui outras unidades de memória e um outro conjunto de periféricos como Ethernet, microfone, *headphone* e portos GPIO. Mais detalhes sobre esta plataforma de hardware encontram-se no respetivo manual.

As ferramentas usadas para a criação de um ambiente embebido baseado em Linux nesta plataforma são descritas posteriormente neste capítulo.

3.1.1 Unidade de Processamento e Área Lógica

A unidade de processamento é equipada com dois processadores ARM Cortex-A9 MPcore que operam a uma frequência máxima de 667 MHz permitindo obter

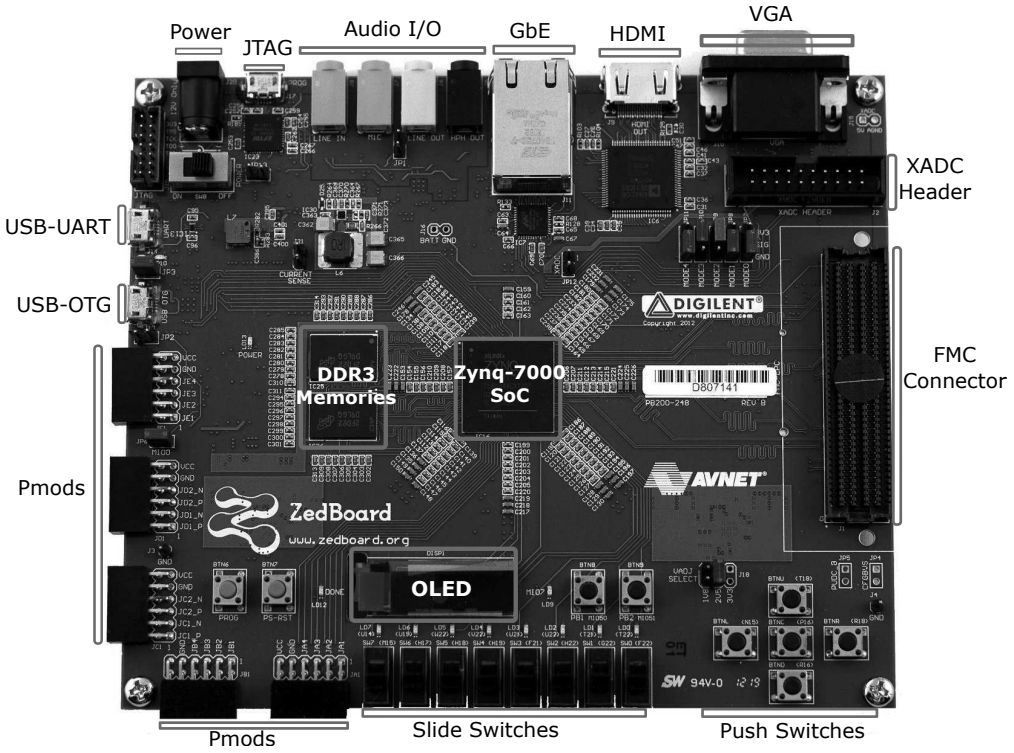


Figura 3.1: Plataforma de desenvolvimento ZedBoard Zynq-7000 [2].

um nível elevado de performance e eficiência energética no domínio das aplicações. Cada *core* baseia-se num processador de 32-bits de arquitetura RISC. Na Figura 3.2 está representada uma visão geral do SoC presente nestas plataformas de hardware.

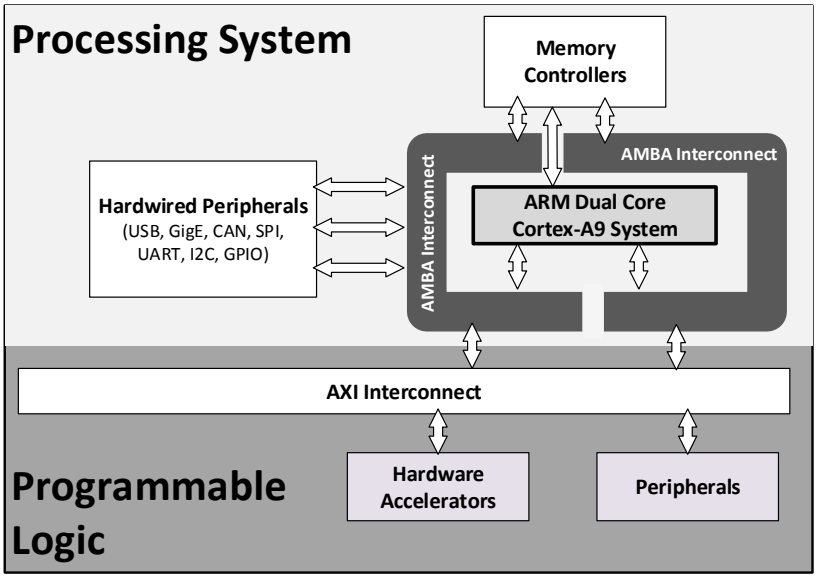


Figura 3.2: Zynq-7000 SoC's simplified overview.

No que diz respeito ao sistema proposto, destacam-se as seguintes características desta unidade de processamento:

- Cada core do processador integra uma MMU, com *caches* de nível 1 separadas para instruções e dados;
- Interface Accelerator Coherency Port (ACP) que mantém os acessos da área programável ao espaço de memória do CPU coerentes;
- *Cache* de nível 2 unificada entre ambos os cores do processador, para acelerar o acesso aos dados da memória;
- Controladores de memória *flash* e DRAM que permitem o acesso a memórias *off-chip* como memórias NOR e NAND *flash*, cartões de memória SD, e memórias DRAM DDR3;
- 256 Kb de memória RAM on-chip utilizada em acessos seguros e rápidos;
- Componente Direct Memory Access (DMA) que suporta múltiplos tipos de transferência de dados sem a contribuição do CPU: memória-memória e memória-periférico;
- Controlador de interrupções flexível que permite responder a eventos de periféricos de hardware externos;
- Dois *timers*/*counters* triplos e três *watchdog timers* proporcionando funcionalidades como contadores de eventos, medidores de tempos de execução e sinalização em caso da falha do sistema.

Em relação à capacidade de endereçamento de periféricos de hardware, esta unidade permite a utilização de duas interfaces SPI, I2C, CAN, Universal Asynchronous Receiver/Transmitter (UART), em conjunto com duas interfaces de Secure Digital Input Output (SDIO), USB e Gigabit Ethernet (GigE) com o mecanismo de DMA embutido, e a utilização de vários portos de entrada/saída para uso genérico (GPIO).

Por outro lado, o *fabric* FPGA da plataforma proporciona a criação de aceleradores dedicados em hardware através da programação dos seus elementos lógicos.

Esta unidade de processamento, desenhada para atingir 2.5 Dhrystone MIPS (DMIPS) operando a uma frequência de 667 Mhz, em conjunto com a lógica programável FPGA tornam-se uma ótima solução para aplicações com requisitos de tempo-real.

3.2 Plataforma Zynq-7000 AP SoC ZC702

Tal como a plataforma Zedboard, a plataforma Zynq-7000 AP SoC ZC702, ilustrada na Figura 3.3 oferece um conjunto semelhante de hardware, como:

- **Zynq-7000 XC7Z020-1CLG484:** SoC híbrido que combina o CPU com a tecnologia FPGA;
- **Memória DDR SDRAM:** Quatro memórias SDRAM de 256 MB que constituem a memória principal do sistema;
- **Suporte para Cartões de Memória SD:** Interface que permite o acesso a cartões de memória SD;
- **USB-to-UART *Bridge*:** Proporciona um meio para comunicar com a plataforma;
- **Fontes de Relógio:** Relógio do sistema, relógio da unidade de processamento e um relógio programável acessível ao utilizador. O relógio do sistema é gerado por um oscilador Low-Voltage Differential Signaling (LVDS) de 200 MHz. O relógio do sistema é fornecido por um oscilador com uma frequência fixa em 200 MHz. E o relógio acessível ao utilizador é gerado por um oscilador programável cuja frequência pode variar entre 10 MHz e 810 MHz;
- **Interface Gráfica:** Interface HDMI que suporta todos os formatos de vídeo até uma resolução de 1080p a uma frequência de atualização de 225 MHz;
- **FPGA Mezzanine Cards (FMCs):** Estes *slot cards* oferecem portos I/O de alta velocidade de forma a serem conectados a módulos FMC já existentes ou a outros dispositivos externos;
- **Outros:** Esta plataforma possui também 16 MB de memória Quad SPI Flash e 1 KB de memória EEPROM, bem como outros periféricos como USB-OTG, Ethernet e portos I/O. Para mais detalhes sobre a plataforma the Zynq-7000 SoC ZC702 é necessário consultar o seu manual.

Como discutido previamente, a unidade de processamento e a área lógica desta plataforma de hardware fazem parte do SoC também presente na Zedboard, logo as ferramentas utilizadas para criar o respetivo ambiente embebido foram as mesmas.

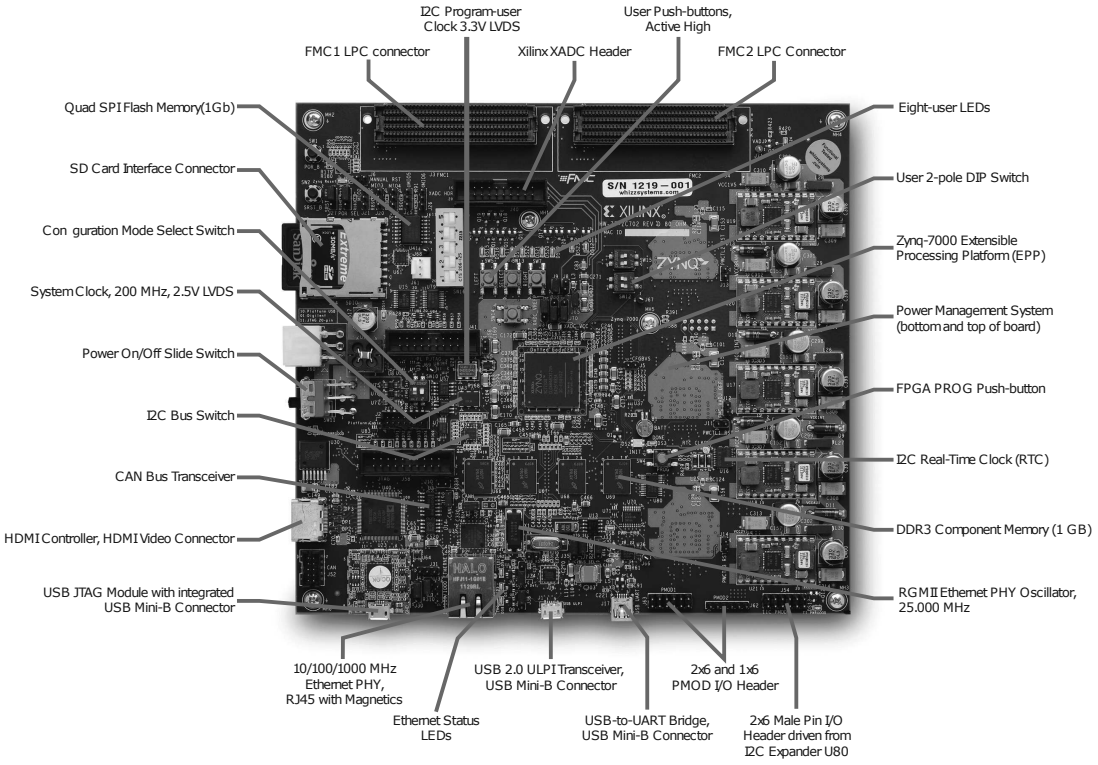


Figura 3.3: Plataforma Zynq-7000 AP SoC ZC702 [18].

3.3 Ferramentas de Desenvolvimento

De modo a criar um ambiente embebido que forneça um meio para desenvolver um sistema embebido completamente funcional baseado em Linux, foram utilizadas as ferramentas da Xilinx e a ferramenta Buildroot. As ferramentas da Xilinx incluem o Vivado Design Suite e o Software Development Kit (SDK), as quais disponibilizam um ambiente de desenvolvimento, simulação e depuração que visa o design, simulação e implementação de aceleradores de hardware, a customização da unidade de processamento da plataforma (envolve a habilitação/desabilitação dos periféricos) e a geração do *bitstream*, usado para programar a FPGA, no Vivado Integrated Development Environment (IDE), e o desenvolvimento e depuração de aplicações software no SDK. Por outro lado, o Buildroot é utilizado para criar um ambiente de *cross-compiling* composto pela *toolchain* da arquitetura da plataforma de hardware que permite a compilação da imagem do *kernel* Linux e a geração do sistema de ficheiros. No Anexo B é descrito a sequência de passos a executar de forma a gerar estes ficheiros.

A combinação das ferramentas da Xilinx com o Buildroot permitem criar um

ambiente de desenvolvimento embebido num PC *host* de formar a gerar o conjunto de ficheiros necessários para executar um sistema baseado Linux completamente funcional na plataforma de hardware alvo.

3.4 Conclusões

As plataformas de hardware discutidas provaram ser soluções escaláveis com um nível elevado de performance e eficiência energética no domínio de aplicações específicas com requisitos muito exigentes. Apesar de ambas as plataformas conterem o mesmo *System-on-Chip*, possuindo as mesmas características *on-chip*, e os mesmos periféricos que possibilitam a utilização de um sistema operativo de suporte, a plataforma Zedboard destaca-se devido às suas rápidas capacidades de expansão através de portos General Purpose Input/Output (GPIO) permitindo a conexão de certos periféricos requeridos em sistemas de aquisição (como um ADC e um monitor LCD). Além disto, a plataforma Zedboard apresenta-se como uma solução menos dispendiosa do que a plataforma ZC702.

Capítulo 4

Architectural Design do Sistema de Aquisição em Tempo-Real

Neste capítulo são descritas a modelação e respetivas técnicas de *design* empregues na implementação do sistema proposto, abordando a implementação das funcionalidades mais importantes e as interações entre as entidades que as representam. Este sistema visa ser aplicado no domínio dos sistemas de energia elétrica com o objetivo de monitorizar as formas de onda das tensões e correntes de um sistema elétrico trifásico.

Modelado na forma de um sistema tempo-real baseado em Linux, este sistema, na vertente de sistema embebido, permite a aquisição de dados que descrevem grandezas físicas do mundo exterior. O sistema preocupa-se em assegurar a necessária precisão, simultaneidade e gama de frequências de amostragem, com o objetivo da informação resultante ser utilizada em aplicações genéricas, com restrições temporais elevadas, como são exemplo, alguns dos sistemas de monitorização, atuação e controlo nos sistemas de energia. Para além disto, o sistema de aquisição proposto oferece um determinado conjunto de serviços e funcionalidades ao utilizador que permitem configuração e controlo internas ao próprio sistema.

Este sistema de aquisição em tempo-real, foi desenvolvido na forma genérica e mapeado para plataforma de desenvolvimento selecionada no Capítulo 3. Fazendo o mapeamento adequado dos pinos, em número e ordem de grandeza, estabelecendo as ligações com os periféricos que este sistema controla, o mesmo pode ser implementado numa qualquer plataforma que lhe proporcione um ambiente de sistema operativo baseado em Linux. Estes módulos foram desenvolvidos especificamente

para o sistema de aquisição e compreendem as seguintes funcionalidades:

- Transdutores e acondicionamento de sinal;
- Conversor Analógico-Digital;
- Plataforma de desenvolvimento Zedboard Zynq-7000;
- Aplicação de software.

A Figura 4.1 apresenta uma visão geral do sistema, identificando os principais elementos e as ligações entre eles.

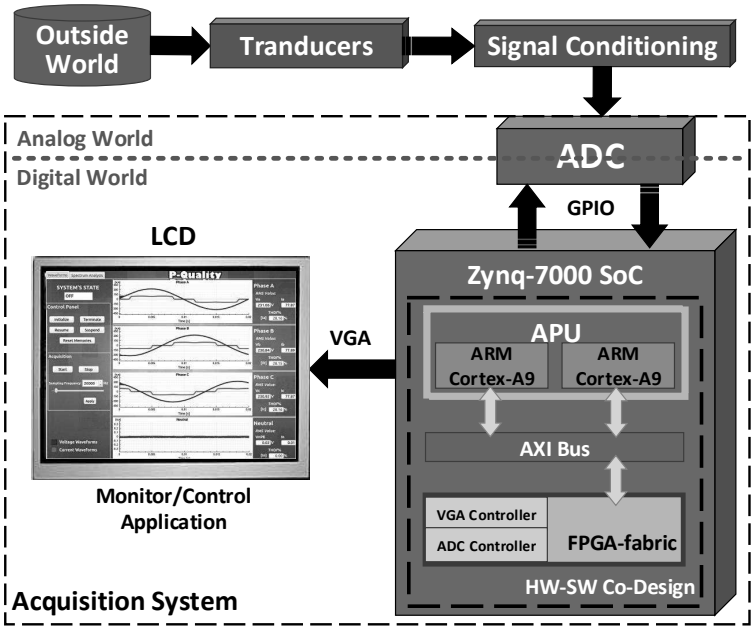


Figura 4.1: Visão geral do sistema de aquisição em tempo-real proposto.

4.1 Transdutores e Acondicionamento de Sinal

A função dos transdutores de sinal em conjunto com o adequado acondicionamento, são a de traduzir uma grandeza física existente no mundo exterior numa grandeza elétrica proporcional, compatível com as ligações de entrada no conversor analógico digital (ADC).

A aquisição dos sinais de um sistema elétrico trifásico é feita através de quatro transdutores de tensão e quatro transdutores de corrente, que operam segundo o efeito Hall. Estes transdutores operam sobre tensões/correntes contínuas/alternadas usando um transformador que proporciona isolamento galvânico entre os

subsistemas. Possuem um terminal identificado através a letra “M”, onde flui uma corrente elétrica proporcional à grandeza física a ser medida. Cabe ao *designer* do sistema dimensionar a resistência elétrica para esse terminal, de acordo com os valores proporcionarias *versus* precisão de medida, requisitos que a aplicação deste sistema enfrenta.

Uma vez que a aplicação de software deve monitorizar as formas de onda das tensões e correntes de um sistema elétrico trifásico, este sistema deve ser capaz de se conectar ao sistema elétrico alvo. Na forma genérica, sistema elétrico trifásico de utilização industrial e/ou doméstico, pode ser resumido em três tensões sinusoidais, desfasadas no tempo em 120°e pelo potencial do neutro para a terra, e por quatro correntes nas três fases e a resultante corrente no condutor neutro.

Para a medição das formas de onda da tensão, foi utilizado o transdutor de tensão LV 25−P do fabricante LEM Co., ilustrado na Figura 4.2. Este transdutor é capaz de converter tensões alternadas e contínuas até 500 V. O seu princípio de funcionamento baseia-se na relação de transformação de 2500:1000, típica um transformador para a corrente alternada.

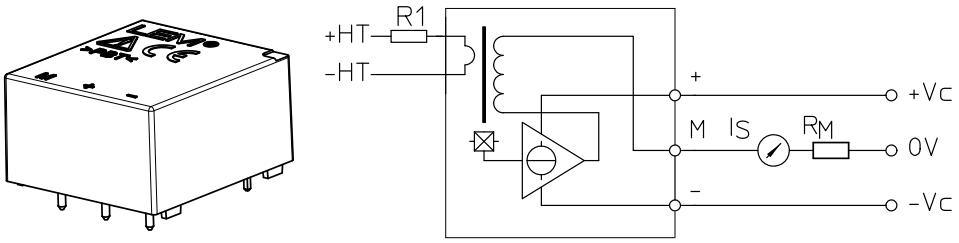


Figura 4.2: Transdutor de tensão LV25-P [6].

Os valores das resistências R_1 e R_M devem de ser calculados de acordo com as correntes nominais nos enrolamentos do primário e secundário e da tensão máxima admitida. Os cálculos referentes ao dimensionamento destas resistências, representados na Equação 4.1 e na Equação 4.2, foram realizados consoante as seguintes especificações:

- Tensão máxima admitida por fase: 360 V;
- Corrente nominal nos enrolamentos do primário: 10 mA.

$$R_1 = \frac{V_{FN}}{I_{PN}} = \frac{360}{0.010} = 36\ k\Omega \tag{4.1}$$

$$P_R = R_1 I_{PN}^2 = 36000 \times 0.010^2 = 3.6\ W \tag{4.2}$$

Para evitar o sobreaquecimento da resistência de entrada e consequente *offset* no valor medido, foram utilizadas duas resistências de potência de 18 $k\Omega$.

No que diz respeito à resistência no terminal "M", esta foi dimensionada de forma a respeitar a gama de valores nas entradas do ADC ($\pm 5\ V$) e em concordância com a máxima tensão anteriormente estabelecida para a entrada do transdutor de tensão (360 V). Deste modo, os cálculos (Equações 4.3 e 4.4) foram realizados garantido que a tensão máxima admitida na entrada corresponda ao valor máximo admitido na entrada pelo ADC.

$$I_{SN} = K_m I_{PN} = \frac{2500}{1000} \times 0.010 = 25\ mA \tag{4.3}$$

$$R_M = \frac{V_{MAX}}{I_{SN}} = \frac{5}{0.025} = 200\ \Omega \tag{4.4}$$

$$P_{R_M} = R_M I_{SN}^2 = 200 \times 0.025^2 = 0.125\ W \tag{4.5}$$

De acordo com a Equação 4.5, deve ser também usada uma resistência de potência que seja capaz de dissipar a potência calculada. Analisando o *datasheet* do transdutor, verifica-se que estes valores respeitam os valores da resistência e corrente nominal estipulados pelo seu fabricante.

Para a monitorização de uma instalação elétrica, é também necessário medir a corrente consumida pelas cargas em cada fase usando para isso os transdutores de corrente. Do mesmo modo que na situação anterior, foi escolhido o transdutor de corrente LA 100-P do fabricante LEM Co., ilustrado na Figura 4.3, que permite medir corrente contínua e alternada no mesmo princípio básico de um transformador, com uma relação de transformação ('K_m') de 1:2000, que suporta uma corrente máxima de 100 A. A principal diferença entre ambos os transdutores, corrente e tensão, encontra-se no acoplamento dos enrolamentos primário, com o sistema elétrico. Não existe um contacto direto entre a grandeza a medir e o transdutor de corrente e como tal não se dimensiona a resistência de polarização na entrada do transdutor.

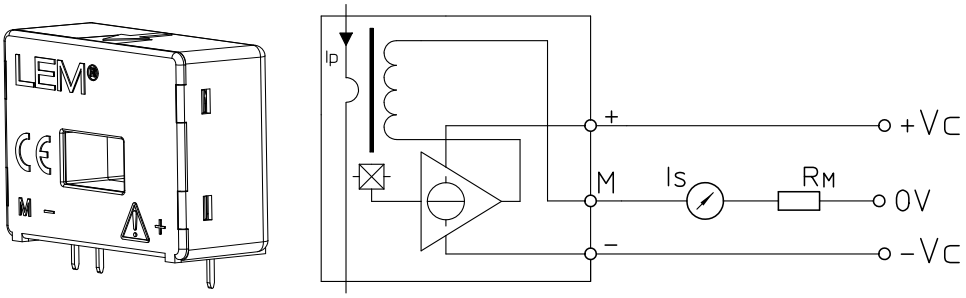


Figura 4.3: Transdutor de corrente LA 100-P [6].

O valor da resistência de medida, R_M, foi calculada da mesma forma que no transdutor de tensão, utilizando a relação de transformação de 1:2000 e uma corrente máxima admitida de 100 A nos enrolamentos do primário do transformador nos cálculos realizados na Equação 4.6.

$$I_S = K_m I_P = \frac{1}{2000} \times 100 = 50\ mA \tag{4.6}$$

O valor da resistência R_M deve respeitar a gama de valores impostas pelo fabricante, de 0 Ω a 110 Ω, resultando na Equação 4.7.

$$V_{MAX} = 5\ V \Rightarrow R_M = \frac{V_{MAX}}{I_S} = \frac{5}{0.05} = 100\ \Omega \tag{4.7}$$

$$P_{RM} = R_M I_{SN}^2 = 100 \times 0.050^2 = 0.25\ W \tag{4.8}$$

Devido aos cálculos realizados na Equação 4.8 foi utilizada uma resistência capaz de dissipar a potência calculada.

4.2 Conversor Analógico-Digital

O ADC é o componente responsável pela interface entre o mundo analógico e mundo digital, convertendo um sinal analógico num valor digital correspondente.

Fazendo face aos requisitos da aplicação de software, foi escolhido o ADC MAX11046 do fabricante Maxim Co. que permite uma amostragem bipolar, a simultaneidade na amostragem dos 8 canais, garantindo o desfasamento entre tensões e correntes, e uma resolução de 16-bits no resultado de conversão. O diagrama funcional deste

conversor encontra-se representado na Figura 4.4. As principais características deste dispositivo são as seguintes:

- **Amostragem Bipolares:** simplifica a aquisição de sinal, visto que os sinais podem ser convertidos diretamente a partir da saída dos transdutores;
- **8 Canais:** possibilita a aquisição de todas as componentes de uma instalação elétrica trifásica necessárias para uma correta avaliação da sua qualidade de energia;
- **Circuito *Track/Hold On-Chip* para cada Canal:** possibilita a discretização de sinais periódicos com frequências que excedam a gama de taxas de amostragem do ADC, numa largura de banda de 4 MHz;
- **Tempo de Conversão:** 3 μs a executar a amostragem simultânea nos oito canais, permitindo alcançar frequências de amostragem até 250 000 amostras por segundo em cada canal;
- **Impedância de Entrada:** possui uma elevada impedância na entrada de 1 $G\Omega$ que atenua as interferências presentes nos canais do ADC e reduz o efeito de carga na corrente da resistência R_M.

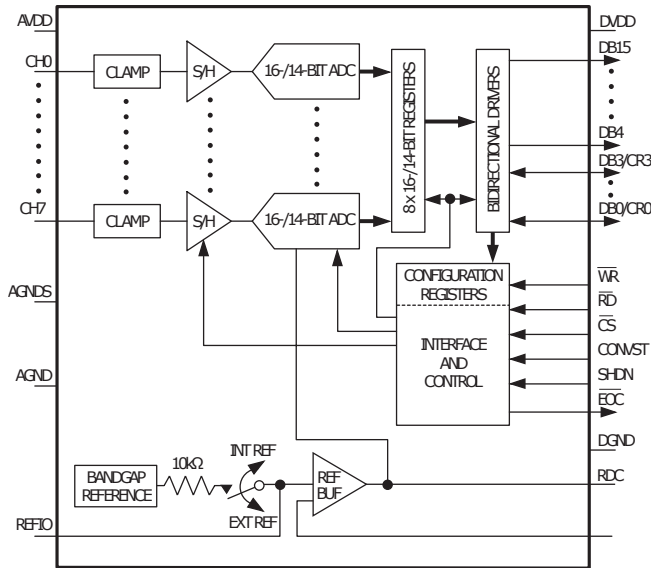


Figura 4.4: Diagrama funcional do MAXIM ADC MAX11046 [8].

No ADC MAX11046, os 16 bits resultantes da conversão de cada canal são acedidos por endereçamento paralelo, fazendo uma leitura sequencial conforme representado na Figura 4.5. Contudo, o ADC deve ser configurado previamente de acordo com as especificações da aplicação.

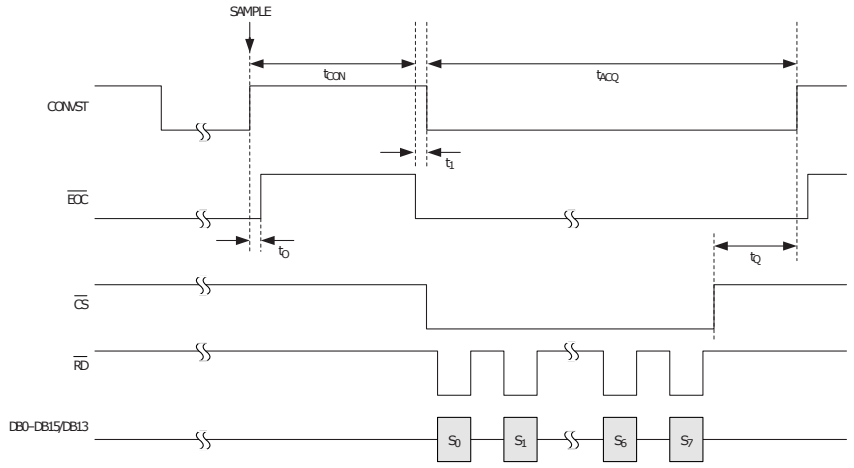


Figura 4.5: Timing Diagram of MAXIM ADC MAX11046 [8].

O valor do registo de configuração do ADC é definido através da interface digital paralela e bidirecional `CR0-CR3` (`DB0-DB3`). Nos processos de leitura e escrita (configuração) do ADC é necessário lidar com os seguintes sinais de controlo: *chip select* ($\overline{\texttt{CS}}$), *read* ($\overline{\texttt{RD}}$), *write* ($\overline{\texttt{WR}}$), *end of conversion* ($\overline{\texttt{EOC}}$), e *conversion start* (`CONVST`). O resultado da conversão de cada canal é apresentado na saída digital paralela do ADC, `DB0-DB15`, consoante o diagrama apresentado na Figura 4.5.

Para a comunicação do hardware de aquisição de sinal com a plataforma de desenvolvimento foi desenhada uma PCB, apresentada na Figura 4.6, contudo, foi utilizada uma placa já implementada que serve o mesmo propósito.

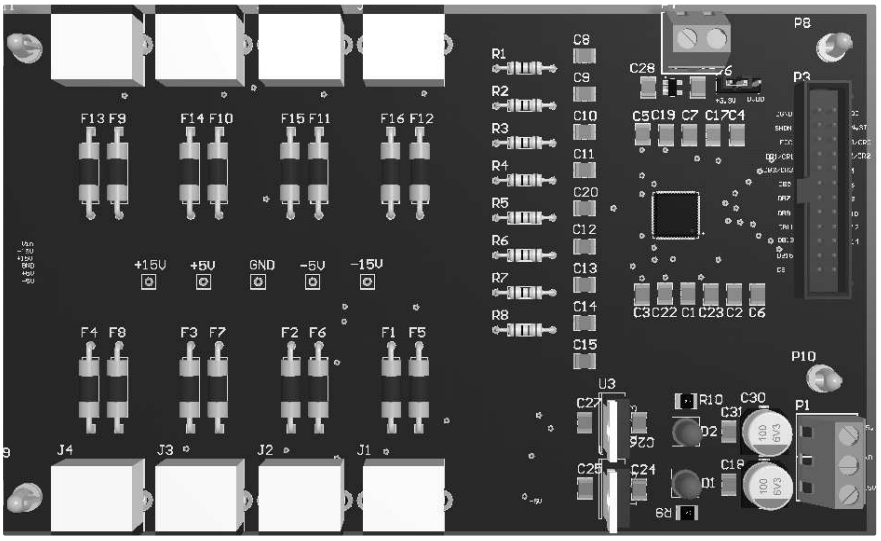


Figura 4.6: Modelo 3D da PCB desenhada na ferramenta Altium.

4.3 Ambiente de Desenvolvimento Embebido baseado em Linux

A plataforma Zedboard Zynq-7000, com uma unidade de processamento baseada em ARM e uma área de lógica programável, permite a criação de aceleradores de hardware dedicados ou a utilização de IP *cores* já existentes no repositório da Xilinx criando um sistema de elevada performance, que promove a satisfação dos objetivos e requisitos que a aplicação de software enfrenta.

As várias características e funcionalidades do Linux levaram à sua adoção nesta plataforma, como a capacidade de abstrair a programação dos detalhes de implementação e hardware da plataforma alvo, integração de múltiplas bibliotecas de software do domínio de aplicação e promoção da portabilidade sobre múltiplas plataformas do tipo CPU+FPGA.

A configurabilidade proporcionada pela plataforma aliada à modularidade e estruturação do *kernel* Linux tornam possível o desenvolvimento de um sistema minimalista que contenha apenas os recursos necessários à sua correta execução.

No que diz respeito às ferramentas de desenvolvimento, o Vivado Design Suite da Xilinx Inc., proporciona um ambiente de desenvolvimento de um SoC com a instanciação de uma ou mais unidades de processamento, as interfaces e os periféricos desejados, que por sua vez podem ser adicionados ou desenvolvidos utilizando a ferramenta IP Integrator. Neste conjunto de ferramentas da Xilinx, encontra-se o Software Development Kit no qual podem ser desenvolvidas e depuradas aplicações de software.

Para executar um sistema embebido completamente funcional com o suporte do sistema operativo Linux é necessário gerar os seguintes ficheiros:

- ***Boot Image*** é composta por:
  - ***First-stage bootloader*** é responsável pela inicialização de algum hardware do sistema e do carregamento do *second-stage bootloader*;
  - ***Bitstream*** contém os dados de configuração usados para programar a área FGPA da plataforma;
  - ***U-boot*** (*second-stage bootloader*) é responsável pela inicialização do resto do hardware do sistema bem como pelo carregamento da imagem do sistema operativo para a memória.

- **Device tree** contém toda a informação necessária sobre o hardware do sistema;
- **Imagem do *Kernel***;
- ***Root file system***.

Existem diversos métodos para gerar estes ficheiros através de ferramentas como Petalinux, Buildroot e Vivado Design Suite. O ambiente de desenvolvimento embebido para a plataforma Zedboard, foi criado utilizando o Buildroot para gerar o sistema de ficheiros e imagem do *kernel* Linux, e as ferramentas da Xilinx para gerar os restantes ficheiros, como ilustrado na Figura 4.7.

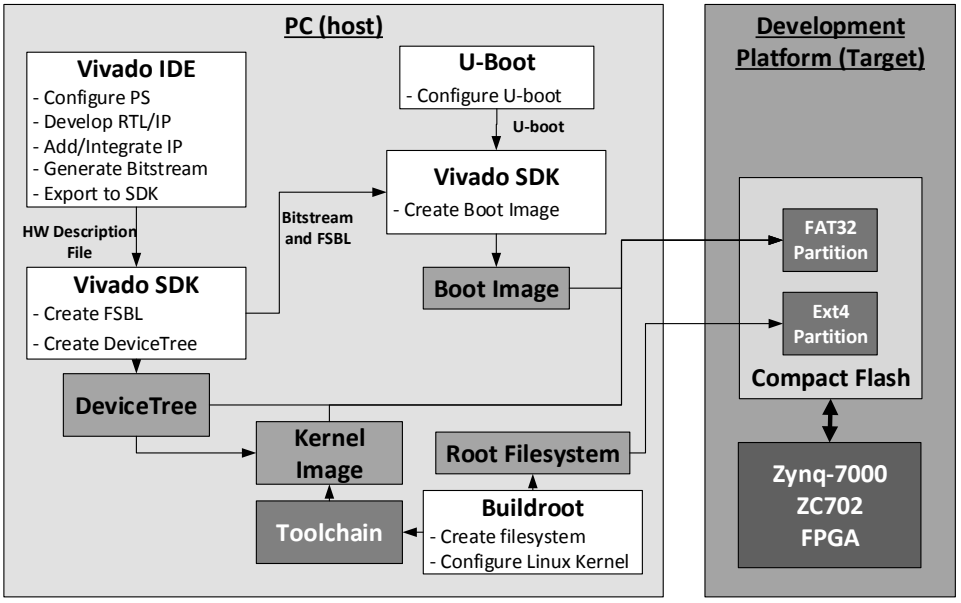


Figura 4.7: Criação de ambiente de desenvolvimento embebido.

4.4 Caracterização do Sistema de Aquisição

O sistema de aquisição em tempo-real proposto foi modulado na vertente de sistema embebido baseado em Linux, com o propósito específico de adquirir de forma permanente o fluxo constante de amostras digitais, que descrevem a evolução temporal das grandezas físicas no mundo exterior. Assim sendo, o conjunto de operações que este sistema desempenha pode ser resumido à inicialização, estímulo, controlo de execução e leitura do dispositivo de aquisição, disponibilizando os resultados à aplicação de software alvo. A Figura 4.8 descreve a pilha tecnológica do sistema de aquisição que sustenta a implementação das funcionalidades e a execução da aplicação do sistema alvo.

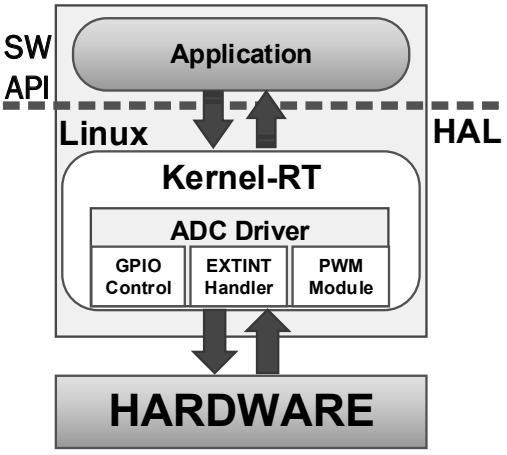


Figura 4.8: Principais componentes do sistema de aquisição.

Para realizar toda a comunicação com o ADC foi implementado um *device driver*, que presta serviços entre a aplicação e o dispositivo de aquisição. Esta comunicação destina-se a controlar os sinais do ADC para o processo de amostragem, ler os dados amostrados e também a configurar o registo de configuração do ADC. Para tal, o *device driver* foi dividido em três módulos chave, responsáveis por: controlar o porto GPIO para comunicar com o ADC; responder à interrupção externa despoletada pelo ADC no fim da conversão dos dados de entrada; e definir a frequência de amostragem do ADC.

Este sistema de aquisição foi implementado recorrendo à linguagem de programação C na implementação do *device driver* no modo *kernel*, e à linguagem de programação orientada a objetos C++ no espaço da aplicação, cujo diagrama Unified Modeling Language (UML) pode ser visualizado na Figura 4.9.

Neste diagrama UML é possível observar as várias entidades que compõem o sistema de aquisição. A classe "CAcquisitionSystem" representa o sistema de aquisição prestando um conjunto de serviços que permitem a sua parametrização e administração, a qual é composta pela: classe "Kernel" que representa a interface entre a aplicação e o sistema operativo; a classe "sampleADC" contém os dados referentes a uma amostragem simultânea dos oito canais do ADC; e pela classe "controlData", a qual possui informação face ao estado do sistema ("running", "suspended", stopped"), e aos índices de leitura e escrita. Por sua vez, esta classe "CAcquisitionSystem" está incluída na classe "CSystem". Esta classe é a entidade central do sistema, pelo que é também composta pela classe "CThread". A classe "CThread" usa a API da POSIX para criar e controlar uma *thread*, cujas estruturas de dados específicas estão representadas na estrutura "STh-

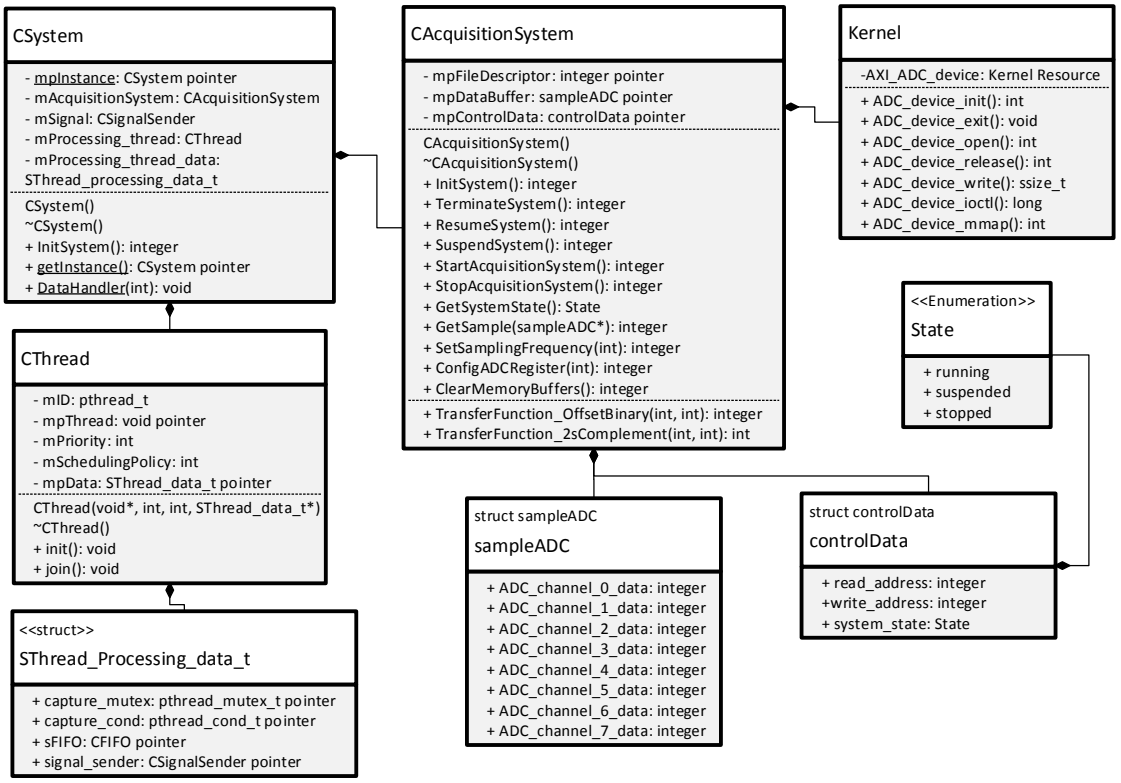


Figura 4.9: Diagrama UML do sistema de aquisição.

read_Processing_data_t". Para integrar a aplicação alvo no sistema, é necessário definir a estrutura específica de "CThread", e a respetiva *thread*, e adicionar estas estruturas à classe "CSystem".

No espaço de utilizador, encontra-se a aplicação do sistema de aquisição, que invoca os serviços do *device driver*. Esta aplicação também tem como finalidade realizar uma recolha periódica dos resultados de conversão do ADC, aplicar a respetiva função de transferência e guardar os dados resultantes numa estrutura de memória First-In First-Out (FIFO). A aplicação específica do sistema alvo irá consumir estes dados armazenados e realizar operações específicas da sua implementação em particular.

O mapeamento deste diagrama de classes na pilha tecnológica do sistema de aquisição baseado em Linux, está representado na forma de diagrama de interações e troca de dados na Figura 4.10. Inicialmente, os dados são obtidos do ADC, através do recurso GPIO, no *handler* de serviço à interrupção desencadeado sempre que o dispositivo de aquisição termina uma conversão. Este traduz os pedidos de leitura da aplicação numa sequência de operações consecutivas que permite ler as amostras convertidas do conjunto de registos do ADC, e subsequentemente

armazená-las numa região de memória pré-determinada.

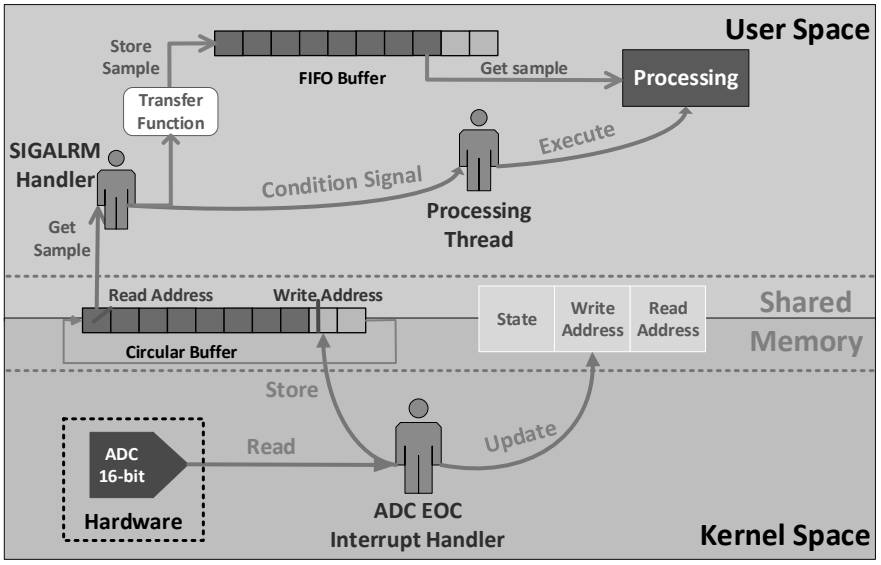


Figura 4.10: Diagrama de interações do sistema de aquisição.

Como melhoria de desempenho na sua forma original *software-only*, o sistema foi otimizado recorrendo a técnicas de programação de software como o *zero-copy*, na transferência de dados entre *kernel-* e *user-spaces*, evitando a cópia entre *high* e *low memory* do Linux. O *device driver* é a entidade com privilégios suficientes para a criação deste *buffer*. A utilização desta técnica contribuiu significativamente para o aumento da performance do sistema.

4.5 Controlo do Sistema de Aquisição

O *core* do sistema desenvolvido é o subsistema de controlo, que fornece um conjunto de serviços e funcionalidades que permitem a parametrizar e administrar do sistema de aquisição de dados. Este subsistema estabelece a interface entre a aplicação alvo e o sistema de aquisição. A Figura 4.11 apresenta o sistema de aquisição numa pilha tecnológica.

Tendo em vista a melhoria na satisfação de requisitos temporais, o controlo do ADC através do periférico GPIO, via *device-driver* no Linux, foi migrado para um periférico dedicado no FPGA-*fabric*, e *kernel* do sistema operativo Linux foi customizado para tempo-real através de *patching*.

Por outro lado, a componente de software do sistema é puramente independente

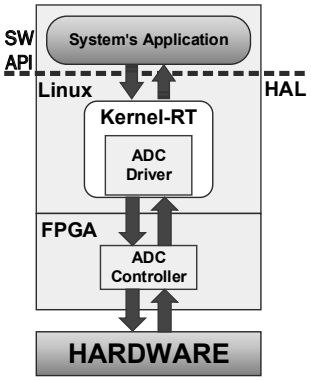


Figura 4.11: Partes constituintes do *core* do sistema de aquisição.

da arquitetura alvo, visto que se baseia no paradigma de programação multi-tarefa da POSIX, em conjunto com as funções da biblioteca C.

O sistema de controlo encontra-se dividido em três módulos fundamentais: o *Controlador do ADC*, sintetizado no FPGA-*fabric*, o respetivo *device driver* em modo *kernel* e a aplicação do sistema de aquisição a correr em modo utilizador.

A utilização do *Controlador do ADC* explora o paralelismo natural da tecnologia FPGA, promovendo a performance global do sistema e contribuindo para uma melhoria na satisfação dos requisitos temporais do sistema.

A programação no modo *kernel* garante o acesso aos recursos da plataforma alvo, permitindo o controlo dos periféricos, na forma de código pré-definido pela API do *kernel* do Linux mediante uma conduta de acesso rígida que se destina a preservar a integridade do sistema.

Num nível de abstração mais elevado, a programação no modo de utilizador (onde as aplicações são executadas) oferece um mecanismo de proteção da memória do sistema que regula o acesso dos processos no espaço de utilizador, para evitar o corromper da memória e de outros recursos a em utilizados pelos processos ou pelo *kernel*, garantindo a estabilidade do sistema.

O sistema de controlo foi desenhado com o intuito do sistema de aquisição possuir as seguintes funcionalidades:

- Inicializar/terminar sistema;
- Começar/parar o processo de amostragem do ADC;
- Ler os dados amostrados;

- Definir a frequência de amostragem;
- Modo *shutdown*.

O *device driver* em conjunto com o sistema operativo de suporte constituem uma camada de abstração de hardware, da qual a aplicação beneficia em espaço de utilizador ao aceder aos recursos de hardware sem a necessidade de conhecer detalhes da implementação. No topo desta camada encontra-se a API em software que fornece o conjunto de serviços destinados a controlar o sistema de aquisição.

A aplicação de software, faz uso os serviços fornecidos do *device driver* para interagir com o *Controlador do ADC*.

4.6 Controlador do ADC

O controlo do ADC foi simplificado fazendo uso do recurso FPGA, mapeando as funcionalidade específicas de escrita e leitura dos registos do ADC num periférico IP especializado. Nesta secção é descrita a tarefa responsável pelo controlo do conversor analógico-digital, descrito em 4.2, e pelo armazenamento temporário dos resultados de conversão. O objetivo deste *offloading* foi de aumentar a performance do sistema na resposta a eventos repetitivos de curta duração, adiando a interrupção do sistema até este dispositivo atingir uma quantidade pré-determinada e parametrizável de amostras no armazenamento temporário. A Figura 4.12 representado o *top-level* do RTL-IP *core* desenvolvido para o controlo deste dispositivo de hardware.

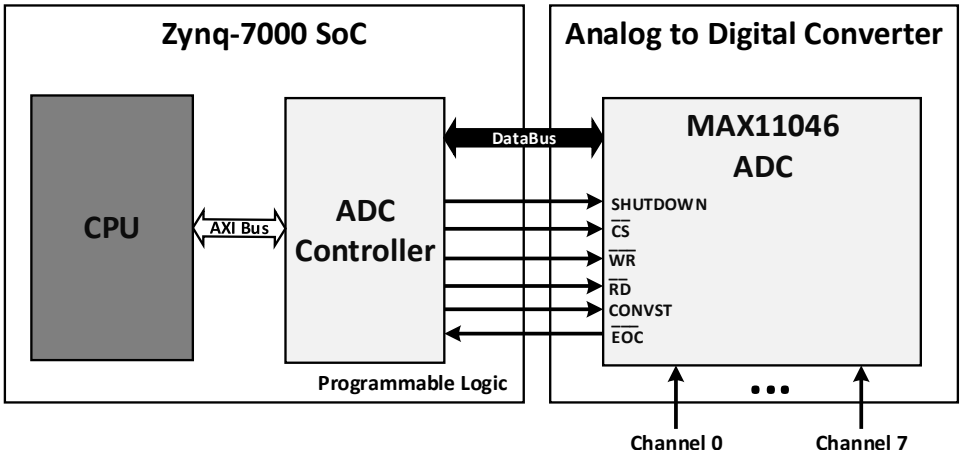


Figura 4.12: *Top-level* do periférico de hardware.

À semelhança de outros IP *cores* como os da Xilinx e/ou da Altera (principais fabricantes de dispositivos FPGA), este periférico necessita de ser conectado ao barramento do sistema para poder ser acedido pelo sistema de controlo. A interface com o barramento é um processo complexo, pelo que o recurso às ferramentas da Xilinx facilitam este processo, abstraindo o grau elevado de complexidade associado à comunicação através de um *wrapper*.

No Vivado IDE é possível criar um periférico customizado através da ferramenta "Create and Import Peripheral" com as seguintes características:

- Interface(s) AXI *master*/*slave*;
- Registos de configuração que podem ser acedidos por software através dos seus endereços (até 4096 registos);
- Suporte a interrupções.

Como resultado, obteve-se um controlador HDL dedicado, capaz de fazer interface entre o barramento AXI e o dispositivo ADC, gerir localmente a interrupção do ADC, através de interfaces do tipo registo. O conjunto de registos de configuração/operação são utilizados para controlar o modo de operação do controlador ADC, definir a frequência de amostragem e habilitar a leitura das amostras convertidas, em resposta ao fim de conversão do ADC. A ação de desencadear uma interrupção no sistema, sinaliza o sistema operativo que os dados da memória de armazenamento temporário devem ser lidos.

O *Controlador do ADC* foi implementado recorrendo à linguagem Verilog HDL, sendo formado de vários módulos que executam determinadas tarefas especificas para o correto funcionamento do dispositivo. A estrutura interna deste periférico encontra-se representada na Figura 4.13.

Os registos estabelecem a interface entre a aplicação de software e a unidade de controlo interna ao *Controlador do ADC*. As funções de cada registo de configuração usado são descritas na Tabela 4.1. Cada um destes é acedido para leitura/escrita pelo *device driver* do sistema operativo Linux.

Como descrito na tabela anterior, o registo `Reg0` é utilizado para controlar o modo de operação do controlador em hardware. Na Tabela 4.2 estão representados os campos do registo e as suas funções.

O conjunto principal de comandos para `Reg0` são descritos na Tabela 4.4.

Este *Controlador do ADC* foi divido em vários módulos de modo a realizar as

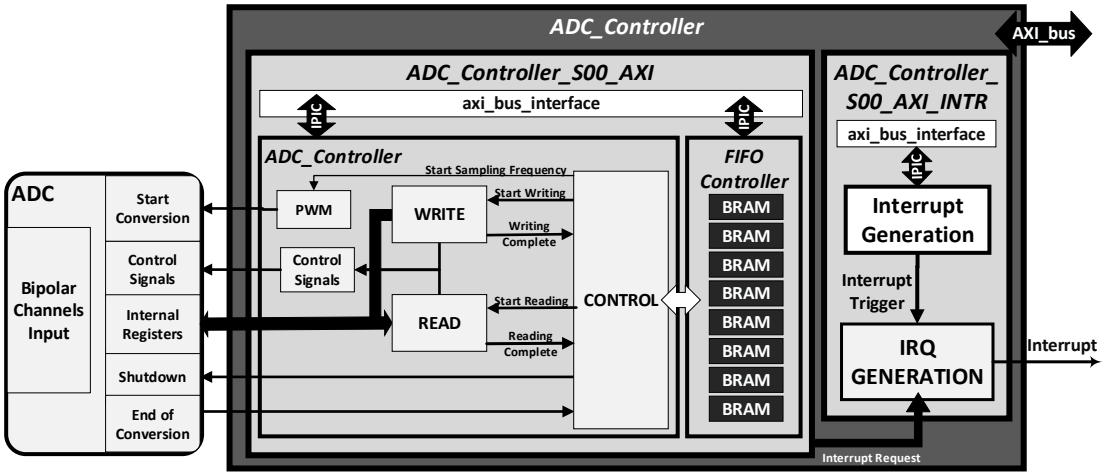


Figura 4.13: Estrutura interna da *Controlador do ADC*.

Tabela 4.1: Registos disponíveis no *hardware driver*.

Registo	Acesso	Descrição
Reg0	Escrita	Dita o modo de operação da tarefa em hardware
Reg1	Escrita	Mantém o valor usado na configuração do registo do ADC
Reg2	Escrita	Define a frequência do Pulse-Width Modulation (PWM) utilizado para iniciar a conversão no ADC
Reg3	Escrita	Mantém informação sobre o estado da interrupção
Reg4	Escrita	Contém os dados armazenados na Block RAM (BRAM)

funcionalidades do sistema de aquisição estipuladas: módulo de controlo, módulo de escrita, módulo de leitura, e o módulo de interrupção. Para além destes, foi implementado um módulo específico para controlar o acesso aos elementos de armazenamento do próprio periférico.

4.6.1 Controlo do ADC

O *Módulo de Controlo* é o núcleo do periférico controlador de ADC, sendo responsável por executar o conjunto pré-determinado de ações de controlo no ADC, de acordo com estimulo presente no registo `Reg0`. Este módulo foi modelado tendo como baseado o conceito tradicional de uma *thread* em software, pelo que possui a capacidade de estar: não criada (estado `RESET`); criada e à espera de um sinal (estado `READY`); suspensa (estado `HOLD`); e em execução (estado `OPERATION`).

O fluxo de execução deste controlador depende dos bits ativos no registo `Reg0`,

Tabela 4.2: Registo de operação da tarefa em hardware, `Reg0`, cujos campos são descritos na Tabela 4.3.

Reg0	Enable	Shutdown	...	Clear	Write	Read
	31	30	3	2	1	0

Tabela 4.3: Descrição dos campos do registo de controlo do controlador em hardware, `Reg0`.

Campo	**Bits**	**Descrição**
Enable	31	Habilita o controlador em hardware
Shutdown	30	Suspende a execução do controlador em hardware
...	29-3	Não usados
Clear	2	Limpa as memórias do periférico
Write	1	Inicia o processo de escrita no registo do ADC
Read	0	Inicia o processo de amostragem

representado na Tabela 4.2. De acordo com o nível lógico de cada bit, o controlador em hardware pode ser habilitado/desabilitado, pode executar a operação alvo (estado `OPERATION`), pode iniciar o processo de configuração do ADC (estado `INIT`), ou pode suspender a execução do controlador em hardware (estado `HOLD`). Estas ações traduzem-se no início dos processos de amostragem, de configuração do registo do ADC ou do modo *shutdown* do ADC, respetivamente. O diagrama da máquina de estados deste módulo encontra-se representado na Figura 4.14.

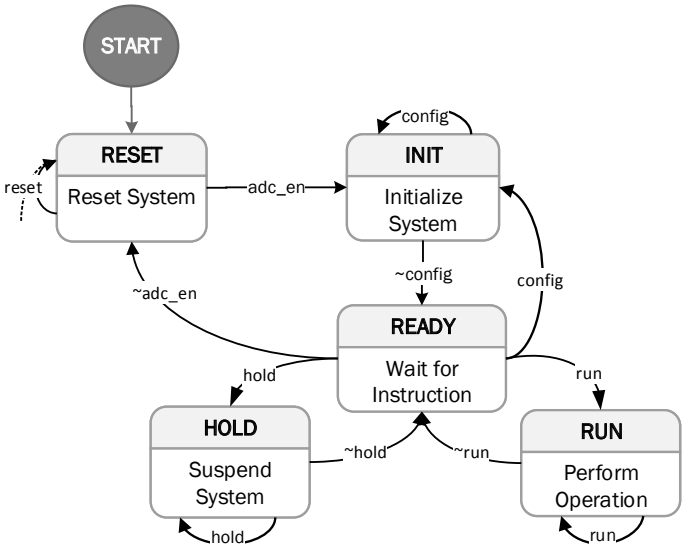


Figura 4.14: Diagrama da máquina de estados do *Módulo de Controlo*.

Quando o sistema é iniciado (`Reg0` = 0x00000000), o Módulo de Controlo espera no estado `RESET` até receber um comando para habilitar o controlador em hardware,

Tabela 4.4: Descrição dos comandos do registo de controlo `Reg0`.

Valor Reg0	Bits Ativos	Descrição
0xC0000000	31, 30	Suspende a execução do controlador em hardware
0x80000002	31, 1	Escreve no registo de configuração do ADC
0x80000004	31, 2	Limpa as memórias FIFO do periférico
0x80000001	31, 0	Inicia o processo de amostragem do ADC a uma frequência definida pelo registo `Reg3`

pelo que define os valores *default* no estado `INIT` e espera no estado `READY`. Estes comandos são enviados pelo *device driver* em software que por sua vez são definidos pela aplicação do sistema.

Por exemplo, se for pretendido suspender a execução do controlador em hardware, o valor 0xC0000000 tem de ser escrito no registo `Reg0`. Neste caso, um pedido para suspender irá direcionar controlador para o estado `HOLD`, até que o respetivo bit `shutdown` seja desativo, retornando assim ao estado `READY`. O mesmo acontece para os processos de escrita e leitura

Em relação aos processos de escrita e leitura o registo `Reg0` terá de conter os valores de 0x80000002 ou 0x80000001, respetivamente. Para configurar o registo do ADC o bit `write` tem de ser ativado, o que irá despoletar um sinal para iniciar o *Módulo de Escrita*, passando para o estado `INIT`. No outro lado, para começar o processo de amostragem e posterior armazenamento, o bit `read` tem de ser ativado, o que irá iniciar o *Módulo de Leitura* e esperar no estado `OPERATION` até que o sistema seja parado.

Outro ponto a destacar são as prioridades dos processos deste módulo. O processo de *shutdown* possui maior prioridade do que o processo de escrita, que por sua vez tem maior prioridade do que o processo de leitura. Isto significa que se ocorresse uma falha no sistema e os bits fossem todos habilitados, o processo de *shutdown* iria executar.

4.6.2 Escrita do ADC

O processo de escrita consiste na configuração dos registos internos do dispositivo ADC, a sua implementação encontra-se representada através de máquina de estados na Figura 4.15.

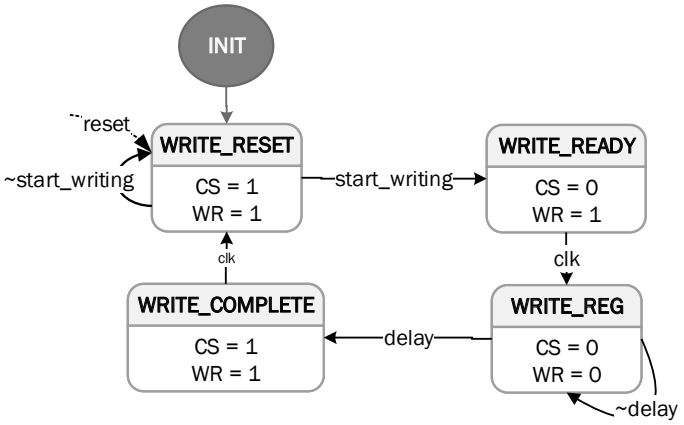


Figura 4.15: Diagrama de máquina de estados do *Módulo de Escrita*.

O *Módulo de Escrita* espera no estado inicial `WRITE_RESET` pelo sinal do *Módulo de Controlo* discutido anteriormente. Quando o processo de escrita é iniciado, estado `WRITE_READY`, este módulo endereça os sinais de controlo do ADC de forma a que o valor de configuração do registo do ADC seja mantido o tempo suficiente no barramento bidirecional do dispositivo. Depois de configurado, este módulo sinaliza que terminou o processo no estado *WRITE_COMPLETE*, retornando de seguida para o estado `WRITE_READY`.

4.6.3 Leitura do ADC

O processo de leitura é constituído pelo *Módulo de Leitura*, o *Módulo PWM* e pelas memórias FIFO. O *Módulo PWM* é responsável por fornecer o sinal `CONVST` ao ADC de acordo com o valor presente no registo `Reg2`, ditando a frequência de amostragem. Quando inicializado, o *Módulo de Leitura* lê as amostras convertidas pelo barramento de saída do ADC. Quando o dispositivo de aquisição sinaliza o fim da conversão (transição ascendente do sinal $\overline{\text{EOC}}$), este módulo lê as oito amostras de cada canal, uma de cada vez. O diagrama da máquina de estados deste módulo encontra-se representado na Figura 4.16.

Quando o módulo termina a leitura das amostras, desencadeia um sinal que vai ser utilizado para iniciar o processo responsável pelo armazenamento das amostras na memória.

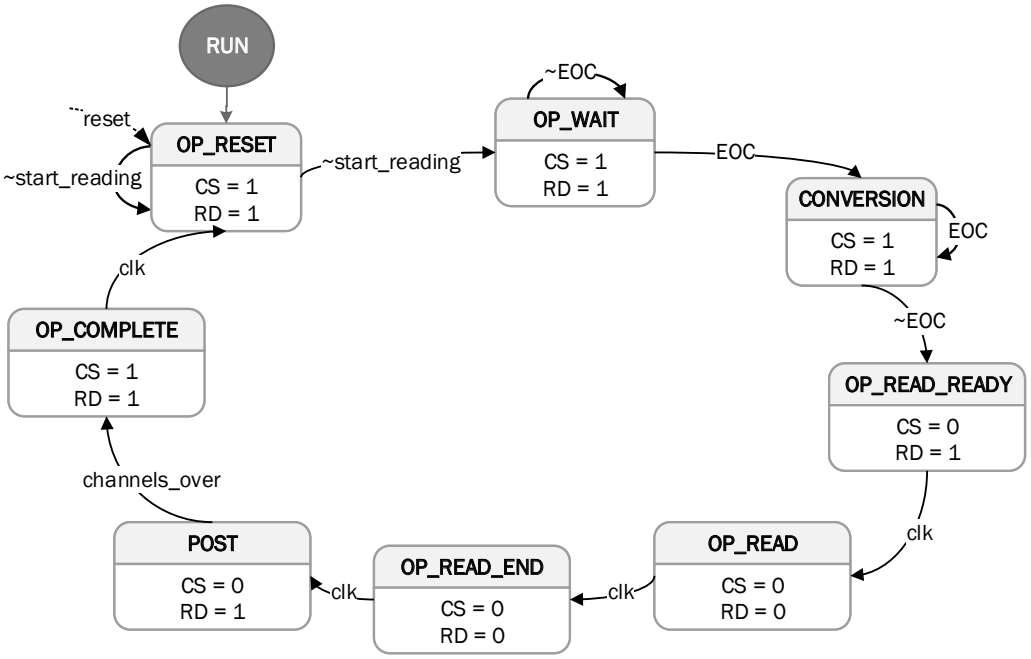


Figura 4.16: Diagrama da máquina de estados do *Módulo de Leitura*.

4.6.4 Armazenamento de Dados

O processo de armazenamento consiste em guardar todos os dados adquiridos dos canais do ADC na respetiva memória FIFO. Foram instanciados oito blocos de memória BRAM *dual-port*, destinados a armazenar até 1024 amostras de cada canal do ADC, permitindo a leitura e escrita simultâneas. Estes elementos de armazenamento foram desenhados para se comportarem como *buffers* circulares devido à facilidade de desenvolvimento.

O processo de escrita na FIFO, ilustrado na Figura 4.17, inicia quando é sinalizado pelo *Módulo de Leitura*. Neste instante, este módulo armazena as amostras nas respetivas memórias e espera até o próximo conjunto de amostras adquiridas pelo ADC. Este processo termina quando as memórias estão cheias ou quando recebe um sinal para limpar o estado delas.

Por outro lado, o processo de leitura das FIFOs é iniciado em três instantes: quando o sistema é iniciado; quando as memórias são libertadas; ou no final de cada processo de leitura. Cada execução do processo termina armazenando o respetivo valor no registo `R4`. Desta forma, a próxima amostra encontra-se sempre pronta para ser lida através do barramento AXI. Este processo garante que é lida uma amostra de cada memória FIFO, cada vez que o registo é acedido. No final da leitura das amostras simultâneas de todos os canais, este módulo prepara os dados

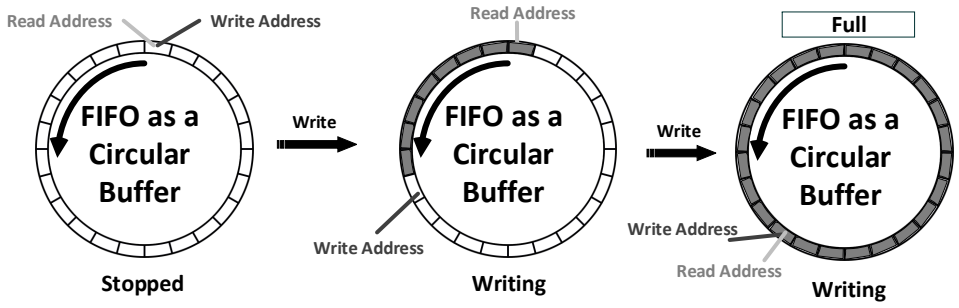


Figura 4.17: Processo de escrita do *Controlador das FIFOs*.

seguintes. Da mesma forma que o processo de escrita, a leitura das memórias termina quando não existem amostras para consumir (é lida a última amostra consumida) ou quando recebe um sinal para reiniciar o estado das memórias.

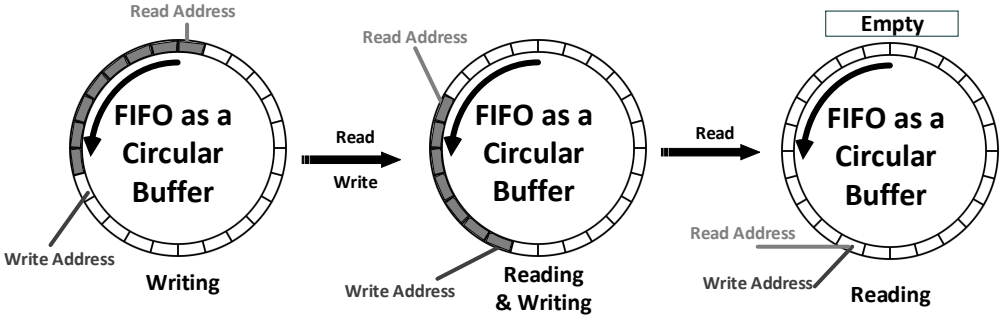


Figura 4.18: Processo de leitura do *Controlador das FIFOs*.

4.6.5 Gestão da Interrupção

O processo de interrupção é gerido pelo *Módulo de Interrupção*, o qual é responsável por notificar o sistema operativo para consumir as amostras existentes na memória do periférico.

A unidade de controlo do *Módulo de Interrupção* desencadeia uma interrupção quando a ocupação na memória do periférico atinge uma percentagem pré-estabelecida evitando o congestionamento no dispositivo e consequente perda dados. Este módulo é também responsável pelas *flags* de interrupção presentes no registo `Reg3`: *flag* de interrupção (`INTR_F`) e *flag* de fim de processamento (`INTR_P`).

Este módulo habilita `INTR_F` quando as memórias chegam a um certo estado e espera até que o respetivo *handler* de serviço à interrupção termine o seu processamento. Quando o sistema operativo atende a interrupção através do respetivo *handler* no *device driver*, as *flags* `INTR_F` e `INTR_P` têm de ser limpas de modo

a notificar este módulo para cessar a geração da interrupção. Após processar os dados, o *handler* de serviço à interrupção deve habilitar `INTR_P`. Neste momento, o *Módulo de Interrupção* retornou ao estado inicial pelo que pode reiniciar o processo de geração da interrupção, ilustrado na Figura 4.19.

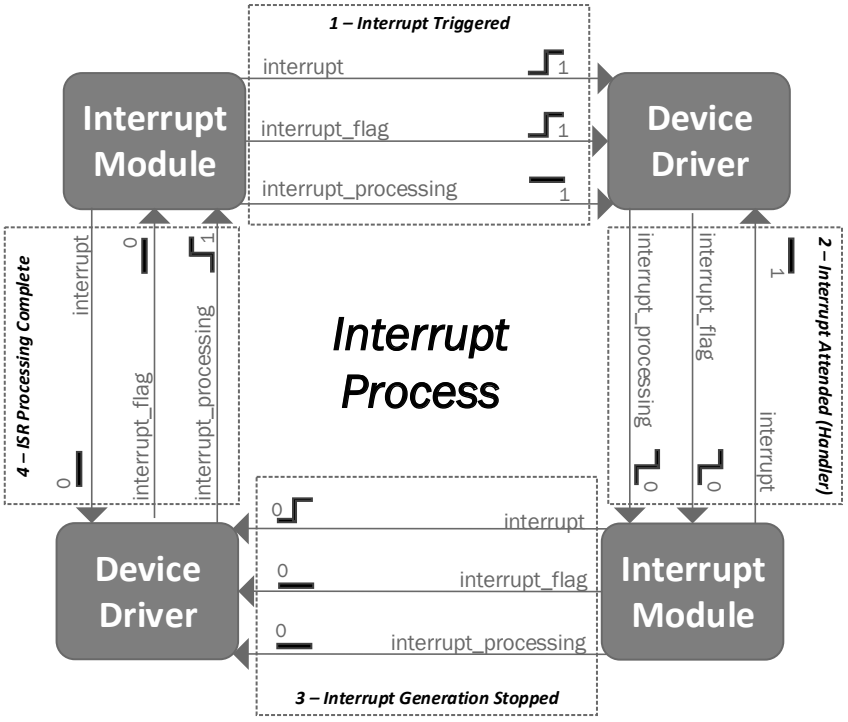


Figura 4.19: Sequência de passos do processo de interrupção.

Todos os periféricos conectados ao barramento AXI são mapeados no espaço de endereçamento I/O pelo que possuem um endereço físico único, atribuído automaticamente pelo Vivado IDE e que é utilizado, ao nível do *kernel*, para escrita e leitura do conjunto de registos e armazenamento temporário. Como ilustrado na Figura 4.20, por um lado, o periférico de hardware está conectado ao barramento AXI, onde comunica com a unidade de processamento, e por o outro lado interage com o dispositivo de aquisição.

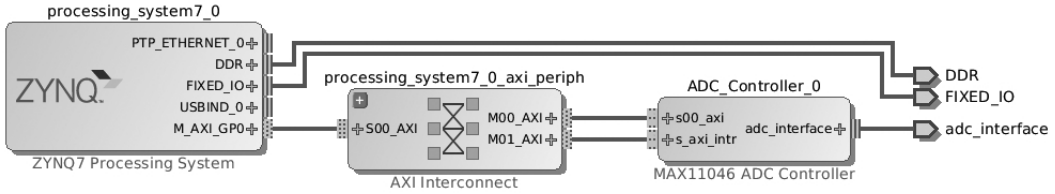


Figura 4.20: Conexão do periférico de hardware à unidade de processamento do SoC.

Para além disto, é necessário gerar o *device tree source file* com a informação

correta dos periféricos utilizados, de maneira a serem detetados pelo sistema operativo. Este ficheiro, quando compilado, permite recolher informações importantes de um periférico de hardware em *run-time* (e.g., endereço, tamanho, linha de interrupção).

4.7 *Software Device Driver*

Um *device driver* proporciona um conjunto de serviços à aplicação que é executada no espaço utilizador, permitindo o controlo dos recursos de hardware do ADC. Uma vez que a maioria das operações de controlo do ADC foram mapeadas no hardware do FPGA, o *device driver* tem apenas como objetivo estabelecer a ligação entre o periférico IP e a aplicação de software. Este *driver* é um *character driver* que oferece um conjunto de serviços disponíveis no modo de utilizador responsáveis por abrir/fechar o *driver*, escrever no periférico de hardware e de mapear uma região de memória numa gama de endereços do espaço virtual da aplicação. Internamente, possui um conjunto de funcionalidades que realizam a inicialização, o acesso e o controlo do periférico IP *Controlador do ADC*. Nesta secção são discutidas todas estas funções abordando as respetivas funcionalidades e relevância para o sistema de controlo.

4.7.1 Inicialização do *Driver*

Esta função de inicialização é executada pelo sistema operativo, em modo *kernel*, no momento em que o módulo é inserido no *kernel* dinamicamente ou estaticamente. Na sua inserção, o *kernel* prepara todos os recursos necessários para a sua inicialização, como a alocação de um espaço de memória onde o driver vai ser carregado.

Como discutido anteriormente, a tarefa do driver de software foi simplificada devido à migração de algumas funcionalidades para o hardware do FPGA. Assim sendo, nesta função de inicialização é realizado o registo do periférico no sistema, registo da função de serviço à interrupção e o pedido do recurso de memória para a partilha de dados com a aplicação. As principais tarefas realizadas nesta função estão representadas na Figura 4.21.

A tarefa responsável pela criação de uma região de memória partilhada, utilizada

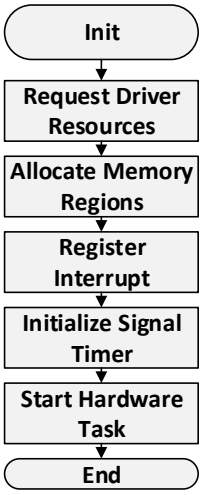


Figura 4.21: Principais ações desencadeadas na inicialização do *device driver*.

como um mecanismo otimizado para transferência de dados com a aplicação, aloca um *buffer* de memória com 1024 posições, cada uma contendo oito amostras de 32 bits. Para além disso, aloca também uma estrutura de dados usada no controlo dos acessos à memória partilhada, a qual é composta pelo endereço de escrita, endereço de leitura e o estado do sistema (*running*, *stopped* and *suspended*).

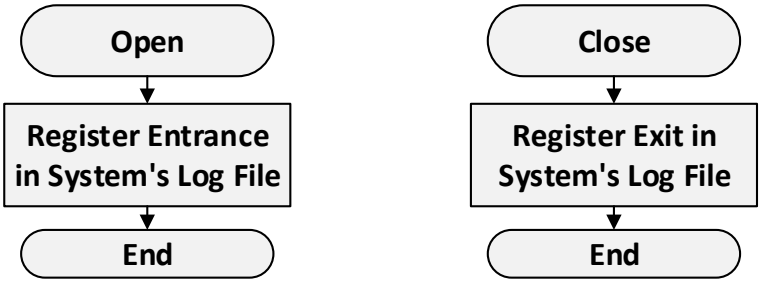
Em resposta à interrupção do *Controlador do ADC* é lançada a execução de um *handler* que o *device driver* deve registar no momento em que faz o pedido da linha de interrupção ao sistema operativo.

A tarefa correspondente ao sinal tem como objetivo inicializar um temporizador que, no fim do tempo definido, sinalize a tarefa no espaço utilizador como pronta para ser executada, de modo a ser realizar uma recolha periódica de dados na memória partilhada.

4.7.2 Abertura/Fecho do *Driver*

As funções *ADC_device_open()* e *ADC_device_close()* são lançadas pelo *kernel* sempre que a aplicação chama as respetivas funcções *open()* e *close()* da biblioteca C. Estas funções permitem à aplicação pedir/libertar o acesso de um certo periférico ao sistema operativo. O sucesso da função *open()* influencia o progresso da aplicação, visto que o seu sucesso indica que o periférico está pronto para ser utilizado.

Neste *driver*, estas funções apenas determinam o sucesso da operação deixando as



(a) função do *kernel* para abrir o ficheiro *device*.

(b) função do *kernel* para fechar o ficheiro *device*.

Figura 4.22: *Flowcharts* das funções *open* (a) e *close* (b) do *device driver*.

inicializações e alocações de memória para quando o *driver* é inserido no *kernel* para evitar que o respetivo tempo do pedido de memória não seja dividido com o tempo de execução da aplicação, como discutido previamente.

Na Figura 4.22 estão representados os *flowcharts* da abertura e fecho do *device driver*.

As funções *open()* e *close()* realizam as ações discutidas anteriormente, bem com registam a entrada e saída do sistema de aquisição no *debug log file* do sistema.

4.7.3 Escrita do *Driver*

A função *ADC_device_write()* do *driver* é lançada pelo *kernel* sempre que aplicação chama a respetiva função *write()* da biblioteca C. O objetivo desta função é escrever no registo de configuração do ADC através do *Controlador do ADC*. Os passos que realizam este processo encontram-se representados na Figura 4.23 na forma de *flowchart*.

4.7.4 *Input/Output Control* do *Driver*

A função *ioctl()* providencia um meio para enviar comandos específicos que visam controlar o periférico de hardware. Por isso, foram definidos alguns comandos para interação com o *Controlador do ADC* e o acesso a algumas estruturas do *kernel*, que são enviados pela aplicação através da função *ioctl()* da biblioteca C. A função lançada pelo *kernel* em resposta à chamada ao sistema por parte da aplicação, designa-se de *ADC_device_ioctl()*, cujos comandos definidos encontram-se

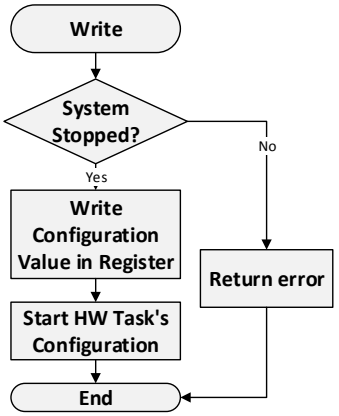


Figura 4.23: Ações realizadas pela função *write()* do *device driver*.

descritos na Figura 4.24.

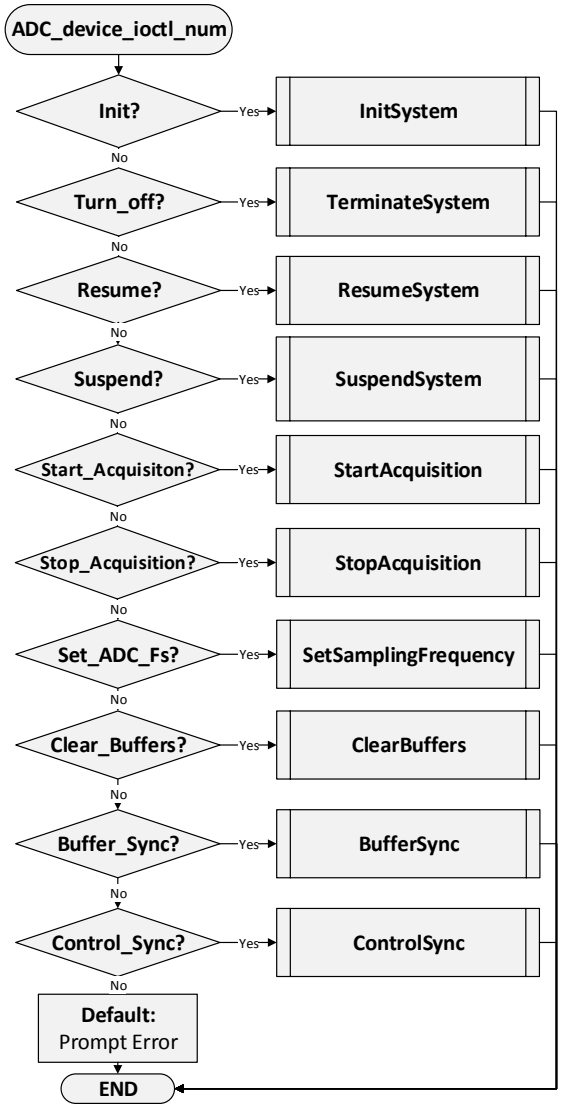
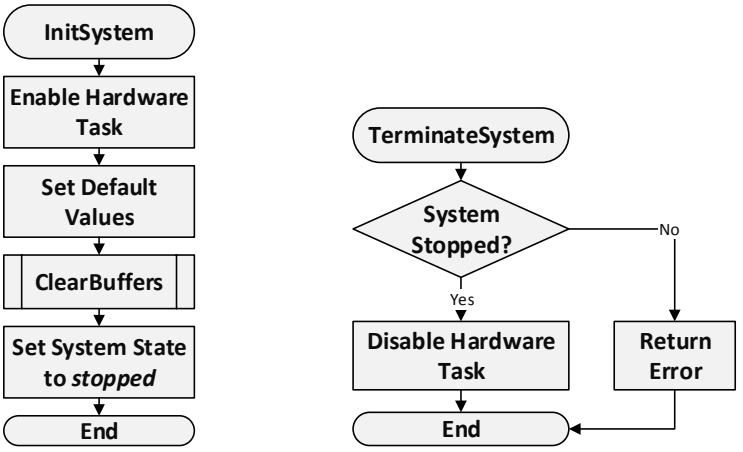


Figura 4.24: Comandos definidos na função *ioctl* do *device driver*.



(a) função do *kernel* para iniciar o sistema.

(b) função do *kernel* para terminar o sistema.

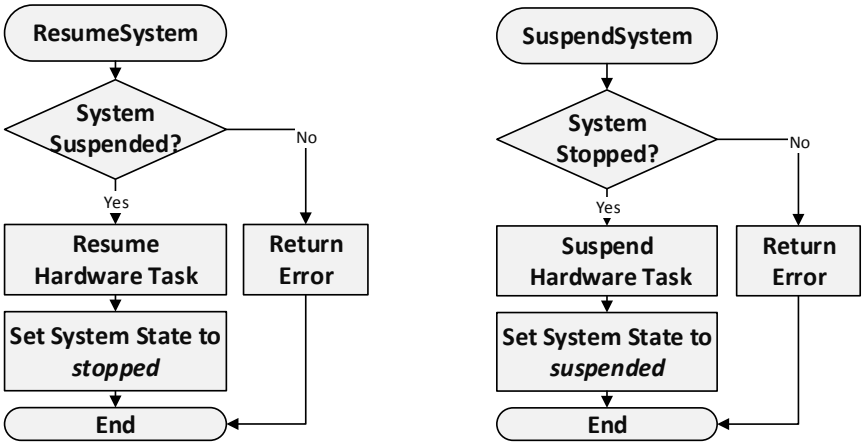
Figura 4.25: *Flowcharts* das funções responsáveis por iniciar (a) e terminar (b) o sistema de aquisição.

Os comandos especificados permitem: inicializar/terminar o sistema de aquição; resumir/suspender o sistema de aquisição; iniciar e parar a aquisição de dados; definir a frequência de amostragem; reiniciar o estado das memórias; e obter os endereços físicos das estruturas de dados criadas em modo *kernel*. Nesta função *ioctl()* é lançada uma função interna responsável por realizar uma determina tarefa de acordo com o comando enviado em modo de utilizador. Além deste comando, é enviado um endereço do espaço de utilizador como argumento, o qual é utilizado para trocar informações com o *driver*.

Estas funções internas, inacessíveis pela aplicação, realizam operações de leitura e escrita sob os registos descritos na Tabela 4.1 permitindo à aplicação o controlo indireto sobre o *Controlador do ADC*.

A função *InitSystem()* permite a inicialização do sistema habilitando o *Controlador do ADC*, definindo os valores por defeito e reiniciando o estado das memórias. Por outro lado, a função *TerminateSystem()* permite terminar o sistema, desabilitando o Controlador do ADC. Na Figura 4.25 estão representadas as tarefas realizadas nestas fases.

As funções *ResumeSystem()* e *SuspendSystem()*, representadas na Figura 4.26, foram modeladas de forma a tirar partido do modo de *shutdown* do ADC. A última função permite suspender a execução do *Controlador do ADC*, colocando o ADC em modo *shutdown*, ao passo que a primeira apenas permite resumir a execução do sistema quando suspenso. Esta funcionalidade pode tornar-se útil ao



(a) função do *kernel* para resumir a execução do sistema.

(b) função do *kernel* para suspender a execução do sistema.

Figura 4.26: *Flowcharts* das funções responsáveis por resumir (a) e suspender (b) a execução do sistema de aquisição.

conjugá-la com a placa de aquisição de sinal podendo originar uma redução no consumo energético.

A função *ClearBuffers()* permite reiniciar os endereços de acesso ao *buffer* de memória partilhado, descartando o seu conteúdo, bem como reiniciar o estado das memórias do periférico de hardware. Esta função é lançada sempre que sistema é inicializado de forma a ser apresentado um "ambiente novo". As ações desencadeadas neste processo estão ilustradas na Figura 4.27.

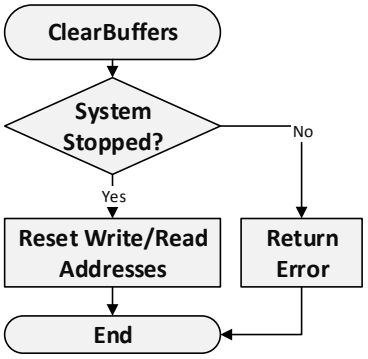


Figura 4.27: Função do *kernel* para reiniciar o estado das memórias do sistema.

A função *SetSamplingFrequency()*, representada na Figura 4.28 na forma de *flowchart*, permite definir a frequência de amostragem do sistema de aquisição, escrevendo no respetivo registo de configuração do *Controlador do ADC*, cujo valor é utilizado para ditar o período do início da conversão. A aplicação especifica este valor através do argumento da função *ioctl()* usado para trocas de dados.

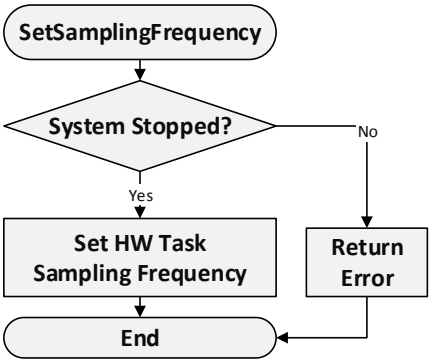
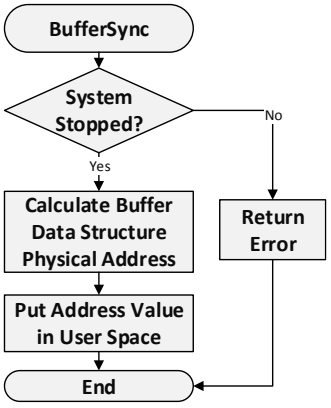
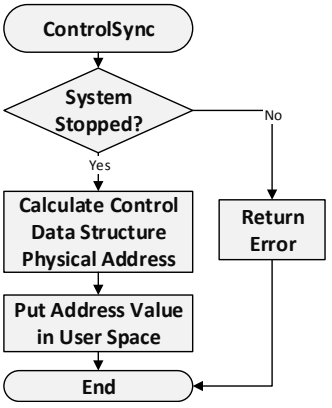


Figura 4.28: Função do *kernel* para definir a frequência de amostragem do sistema de aquisição.



(a) Função do *kernel* que devolve o endereço físico do *buffer* de memória partilhado.

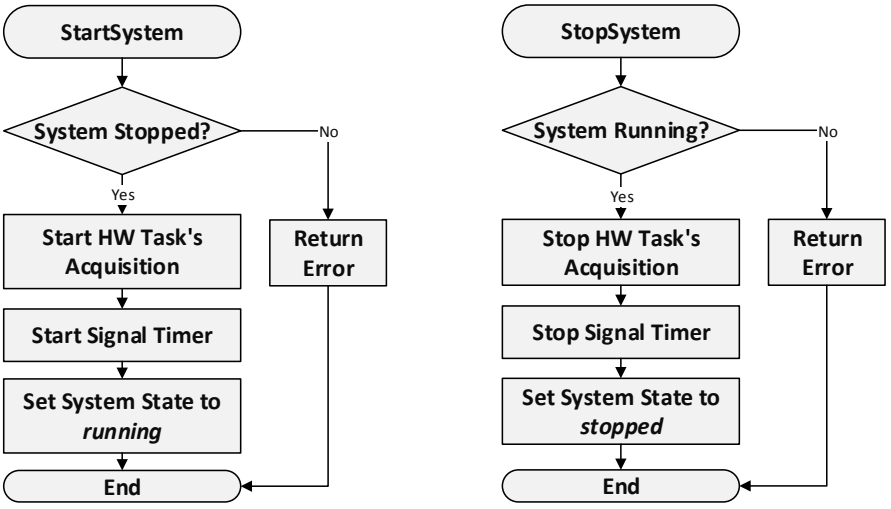


(b) Função do *kernel* que devolve o endereço físico do estrutura de controlo do sistema.

Figura 4.29: *Flowcharts* das funções responsáveis por devolver os endereços relevantes para mecanismo de mapeamento de memória no espaço da aplicação.

As funções *BufferSync()* e *ControlSync()* são utilizadas no processo de partilha de memória executado pela aplicação em modo de utilizador. A primeira função permite à aplicação conhecer o endereço físico do *buffer* de memória através do argumento da função *ioctl()*. A outra função funciona da mesma maneira, só que devolve o endereço físico da estrutura de controlo usada para obter informações sobre o estado do sistema e para controlar o acesso à memória partilhada. Os respetivos *flowcharts* são apresentados na Figura 4.29.

Por fim, as funções *StartAcquisition()* e *StopAcquisition()* têm como objetivo, como o nome indica, iniciar e parar o sistema de aquisição. A iniciação do sistema, representada na Figure 4.30a na forma de *flowchart*, compreende a iniciação do processo de amostragem da *Controlador do ADC*, o começo da contagem do *clock real-time* que notifica a aplicação para consumir amostras e a mudança de estado do sistema.



(a) função do *kernel* para iniciar o processo de amostragem do sistema.

(b) função do *kernel* para terminar o processo de amostragem do sistema.

Figura 4.30: *Flowcharts* das funções responsáveis por iniciar (a) e parar (b) a aquisição do sistema.

No outro lado, a paragem do sistema de aquisição envolve a realização das mesmas tarefas mas, na ordem inversa, como ilustrado no *flowchart* da Figura 4.30b.

4.7.5 Função de Serviço à Interrupção do *Driver*

ADC_Interrupt_Handler() corresponde à função registada no *kernel* durante a inicialização do *driver*, pelo que é lançada em resposta à interrupção gerada pelo *Controlador do ADC*. Esta função é responsável por consumir as amostras presentes nas memórias do periférico de hardware e posteriormente guardá-las num espaço de memória partilhado com a aplicação. O *flowchart* correspondente às ações desencadeadas neste *handler* encontra-se na Figure 4.31.

Antes da leitura das amostras, esta função verifica o estado da memória partilhada, e se esta se encontrar totalmente preenchida, para o sistema de aquisição e notifica a aplicação. Este problema pode surgir se a aplicação não consome as amostras mais rápido que o ritmo a que são armazenadas na memória partilhada. Por outro lado, se a memória partilhada não se encontrar cheia, prossegue com o armazenamento das amostras e termina preparando as estruturas de dados para a próxima iteração.

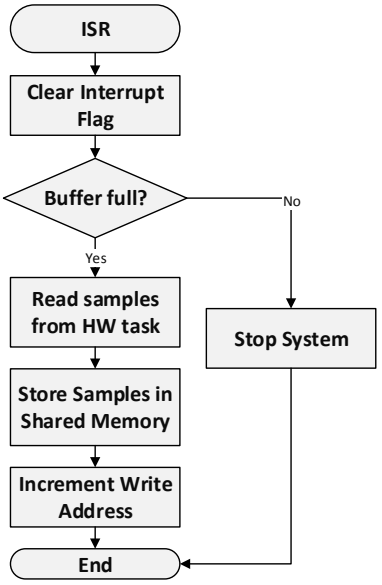


Figura 4.31: *Flowchart* da função que executa a resposta ao evento desencadeado.

4.7.6 Libertação do *Driver*

A função *ADC_device_exit()* do *driver* é lançada pelo *kernel* quando o periférico está prestes a ser removido do sistema. Esta função é a contraparte inversa da função de inicialização, pelo que pode ser lançada quando é chamada a respetiva função da biblioteca C pelo utilizador ou na ocorrência de erro de registo ou alocação de recursos durante a fase de inicialização. Na Figura 4.32 está representado o respetivo *flowchart*.

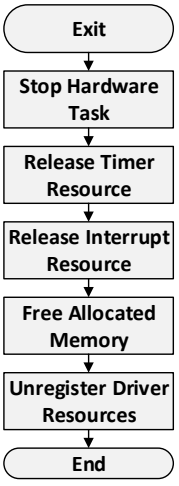


Figura 4.32: Principais ações desencadeadas na libertação do *device driver*.

4.8 *Software API*

A API desenvolvida proporciona uma abstração dos detalhes da implementação, oferecendo um conjunto de funções que são acedidas indiretamente pelo utilizador. Esta interface permite a inicialização, término, parametrização e controlo do sistema de aquisição, como representado na Figura 4.33.

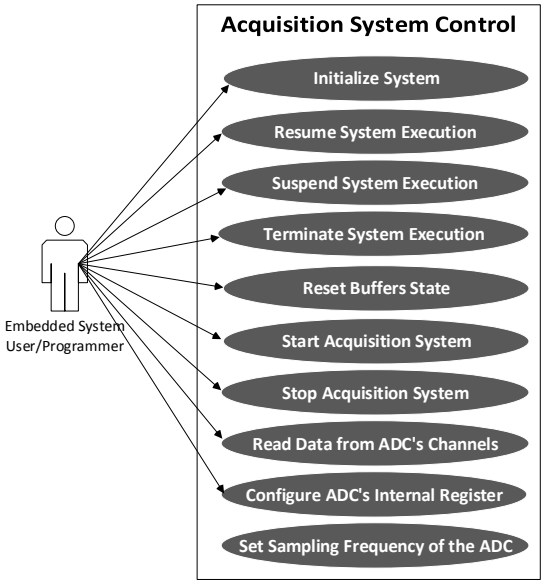


Figura 4.33: Serviços oferecidos pela API de software.

Esta API foi implementada no modo de utilizador, de forma a ser utilizada pela aplicação do sistema, cujo propósito é controlar o sistema de aquisição de acordo com os requisitos do cenário de aplicação.

Conforme a ação do utilizador/programador, a aplicação do sistema executa a respetiva função da SW API de maneira a realizar a ação desejada. Através deste conjunto de serviços, é possível inicializar e terminar o sistema, resumir ou suspender a sua execução, reiniciar o estado das memórias, configurar o registo de configuração do ADC, definir a frequência de amostragem, começar e parar a aquisição.

Na inicialização do sistema, as entidades de execução do sistema são iniciadas com as configurações por defeito. Por outro lado, estas entidades são terminadas aquando da paragem do sistema. Identicamente, no resumo/suspensão do sistema, a execução das entidades que o compõem é retomada ou suspensa, respetivamente, preservando a informação existente no momento.

A configuração do registo do ADC visa configurar o dispositivo de aquisição consoante valor especificado pelo utilizador. O serviço proporcionado relativamente à frequência de amostragem permite definir o período entre cada início do processo de conversão efetuado pelo ADC.

Por fim, no início de aquisição, o sistema inicia o processo de aquisição dos dados resultantes da conversão do ADC disponibilizando-os no espaço da aplicação, ao passo que na paragem de aquisição termina a execução do processo de aquisição.

4.9 Sistema de Aquisição de Dados

Esta secção descreve o modelo desenvolvido para o sistema de aquisição na forma de *flowcharts*, tendo em conta o sistema de controlo e a aplicação do sistema.

4.9.1 Inicialização/Término do Sistema

Durante a fase de inicialização é feita a parametrização da aplicação do sistema responsável por configurar a thread correspondente à aquisição dos dados com prioridade de tempo-real e com uma política de escalonamento `SCHED_FIFO`. Desta a forma, a recolha periódica de amostras realizada pela *thread* que recebe o respetivo sinal desencadeado pelo *device driver* possui uma política de escalonamento *first-in first-out* de forma a ser executada enquanto tiver a prioridade mais elevada ou bloquear. Além disto, as páginas correspondentes ao espaço de memória partilhada são bloqueadas na RAM do sistema para evitar a ocorrência de *page faults* no acesso à memória que induzem *overhead* adicional no sistema, de modo a assegurar que os requisitos temporais são cumpridos.

A aplicação do sistema utiliza a software API para inicializar o sistema, a qual compreende a alocação de todos os recursos de memória necessários para o correto funcionamento do sistema de aquisição, a abertura do *device driver* chamando a função *open()* da biblioteca C, e o lançamento dos comandos, via *ioctl()*, para mapear as estruturas de dados do *kernel* em espaço do utilizador. Esta parte da inicialização está representada na Figura 4.34.

Para além disso, esta aplicação oferece um menu de controlo ao utilizador, para configurar e controlar o sistema de aquisição. Na Figura 4.35 estão representadas todas as ações executadas pela aplicação do sistema.

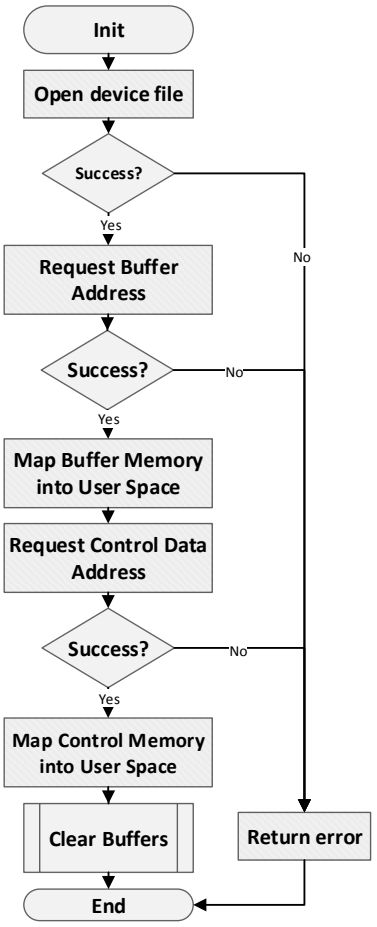


Figura 4.34: *Flowchart* da inicialização da aplicação do sistema.

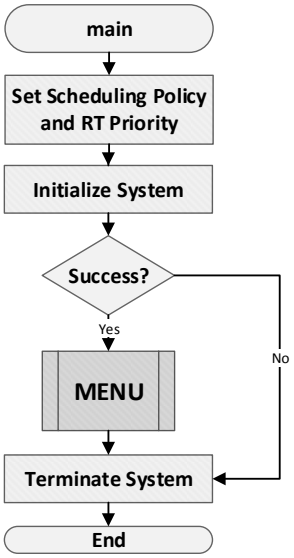


Figura 4.35: *Flowchart* das ações executadas pela aplicação do sistema.

A terminação do sistema é a contraparte da fase de inicialização pelo que usa também a software API para anular todas tarefas realizas na fase inicial, como ilustrado na Figura 4.36.

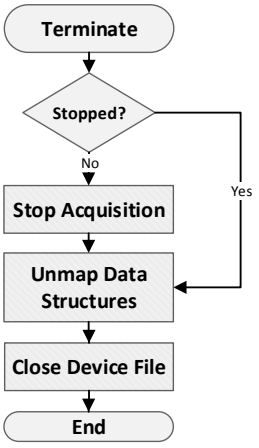


Figura 4.36: *Flowchart* da terminação da aplicação do sistema.

4.9.2 Menu de Controlo

O menu de controlo lançado pela aplicação do sistema, pode ser acedido pelo utilizador através da porta-série (USB-UART) da plataforma na consola terminal. Como tal, o utilizador pode controlar e configurar o sistema de aquisição enviando os comandos descritos na Figura 4.37.

Este menu oferece um conjunto de funcionalidades que permitem a configuração do registo do ADC, a parametrização da frequência de amostragem, o resumo e suspensão do sistema e a iniciação do processo de aquisição de dados do ambiente exterior.

Parametrização da Frequência de Amostragem

A frequência de amostragem indica o período entre a aquisição de cada conjunto de oito amostras convertidas pelo ADC. Este período é definido pelo periférico de hardware, através do respetivo registo de configuração que dita a frequência do sinal de PWM usado para a iniciação de cada conversão do ADC. No *flowchart* da Figura 4.38 estão representados as tarefas realizadas para configurar a frequência de amostragem.

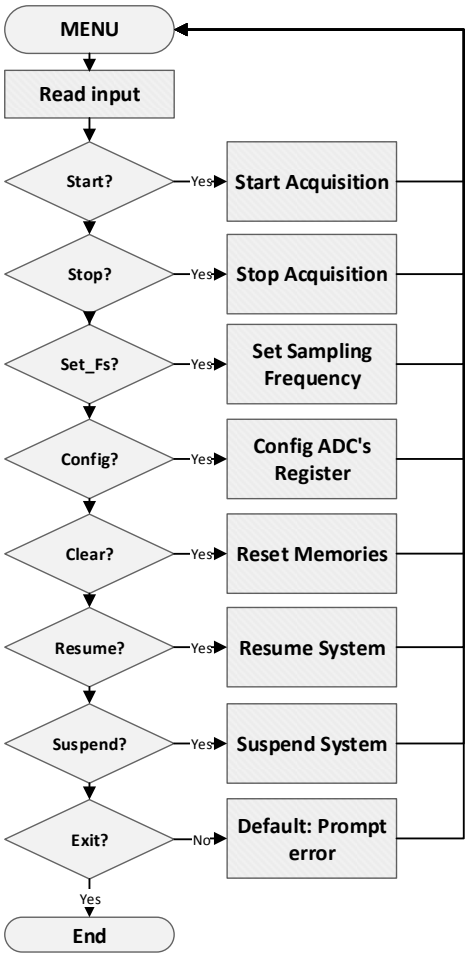


Figura 4.37: *Flowchart* das funcionalidades proporcionadas pelo menu de controlo.

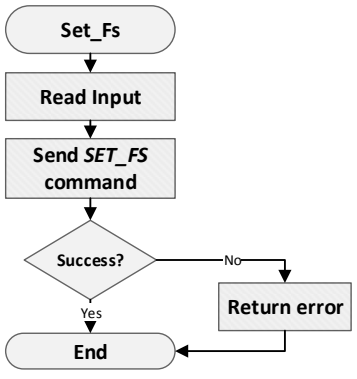


Figura 4.38: *Flowchart* da função responsável por alterar a frequência de amostragem do sistema.

Desta forma, a aplicação do sistema lê o valor do terminal definido pelo utilizador, e envia-o para o *device driver* através da função *ioctl()*. Em espaço do *kernel*, o driver define a frequência de amostragem escrevendo no respetivo registo do *Controlador do ADC*. Posteriormente, quando o processo de amostragem for iniciado, o sistema adquire cada conjunto de amostras no período definido pelo valor do registo da frequência de amostragem.

Configuração do Registo do ADC

O conversor analógico-digital especificado na Secção 4.2 possui um registo de configuração que permite definir a referência do ADC (externa ou interna), o formato dos dados de saída (*offset binary* ou complemento para dois) e o modo de aquisição. Na Figura 4.39 está representado o *flowchart* referente à configuração do registo do ADC.

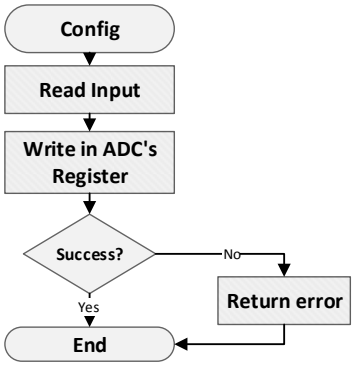
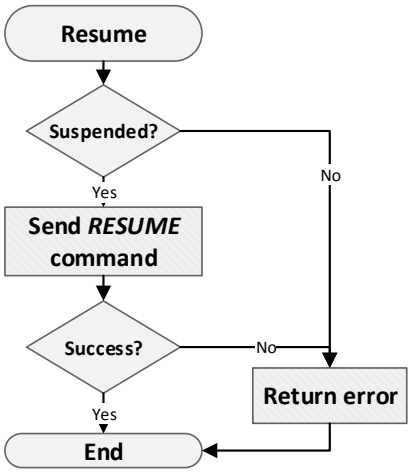


Figura 4.39: *Flowchart* da função responsável por configurar o registo do ADC.

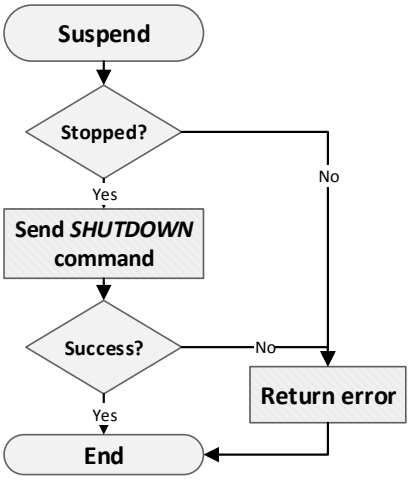
Por defeito, o ADC é configurado para utilizar referência interna, o formato *offset binary* nos dados da saída e para o pino `CONVST` controlar a aquisição. Mais detalhes sobre estes modos de configuração estão presentes no *datasheet* do ADC.

Resumir/Suspender Sistema

A capacidade de resumir/suspender a execução do sistema é uma funcionalidade extra oferecida pelo sistema de aquisição, devido ao facto do ADC possuir um modo *shutdown*. Assim, o sistema de aquisição pode ser suspenso e ser retomado de seguida, restaurando as configurações já efetuadas. Na Figura 4.40 estão representadas as ações desencadeadas nestes processos.



(a) Função da aplicação do sistema para resumir a execução do sistema de aquisição.



(b) Função da aplicação do sistema para suspender a execução do sistema de aquisição.

Figura 4.40: *Flowchart* das funções, no espaço do utilizador, responsáveis por resumir (a) e suspender (b) o sistema de aquisição.

A diferença entre resumir/suspender e inicializar/terminar encontra-se na preservação dos dados no primeiro conjunto de funções.

Iniciação do Sistema de Aquisição

O serviço de iniciação do sistema de aquisição tem como funções iniciar o processo de amostragem do ADC colocando o sistema em execução (*running*). O sistema de aquisição entra em execução enviando o respetivo comando para o *device driver* com o identificador do processo (*pid*), que contém a *thread* responsável pela resposta ao sinal periódico discutido previamente, passado como argumento da função *ioctl()*. Este *pid* é utilizado para instalar o sinal junto do *kernel* Linux, o qual é despoletado por um temporizador. De seguida, a aplicação do sistema lança a *thread* responsável pelo processamento dos dados, a qual possui a sua própria política de escalonamento e prioridade definidas pelo utilizador/programador de acordo com as suas necessidades. O *flowchart* que representa estas ações encontra-se na Figura 4.41.

O processo de amostragem é realizado pelo periférico de hardware. De seguida o *device driver* lê os dados adquiridos e guarda-os numa região de memória partilhada com a aplicação. Quando o sistema de aquisição entra em execução o recurso CPU é dividido entre a *thread* aquisição de dados, a *thread* da aplicação alvo e

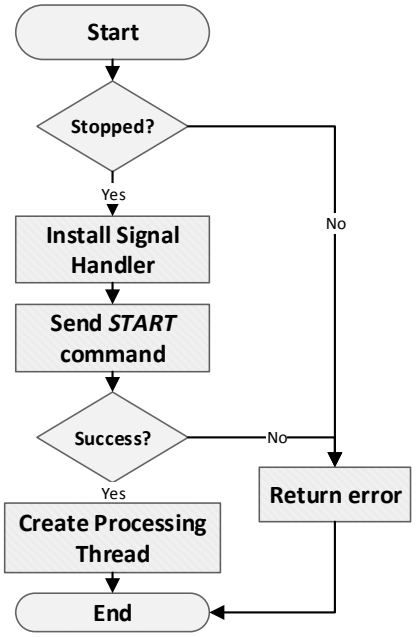


Figura 4.41: *Flowchart* da função responsável por iniciar o sistema de aquisição.

os serviços do sistema operativo, incluindo o *handler* de serviço à interrupção registada pelo *device driver*.

Paragem do Sistema de Aquisição

Ao contrário da sua execução, a paragem do sistema de aquisição envolve o envio de um comando para o *device driver*, que por sua vez cessa o temporizador do sinal, o processo de amostragem do periférico de hardware, altera o estado do sistema (*stopped*) e finaliza terminando a *thread* de processamento de dados. Na Figura 4.42 encontra-se representado o respetivo *flowchart*.

O sistema de aquisição pode ser parado por ação do utilizador ou na ocorrência de uma falha no sistema. Este último caso pode acontecer caso a aplicação não processe os dados o suficientemente rápido.

4.9.3 Partilha e Leitura de Memória

Esta secção descreve todos os passos necessários à implementação de um mecanismo de mapeamento de memória, bem como, o processo de aquisição e partilha e memória entre as três principais entidades do sistema: periférico de hardware, *device driver* e a aplicação do sistema.

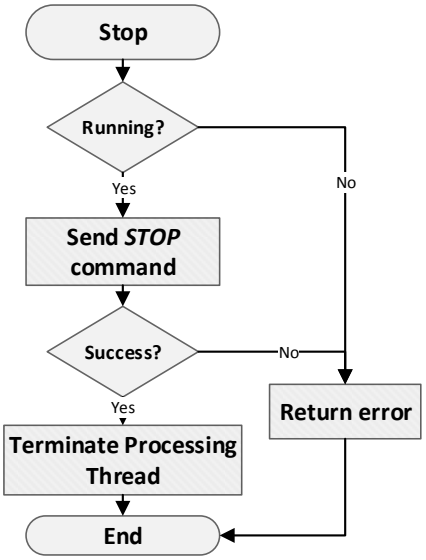


Figura 4.42: *Flowchart* da função responsável por parar o sistema de aquisição.

Criação de um Espaço de Memória Partilhada

A criação de um espaço de memória partilhado entre modo *kernel* e modo de utilizador começa ao nível da aplicação, onde é enviado um comando para o *device driver* para obter o endereço físico da estrutura de dados. Com o endereço físico, a aplicação calcula o número da página que contém a estrutura de dados e o respetivo *offset* dentre dessa mesma página. De seguida, a aplicação efetua um pedido para mapear a respetiva página numa gama de endereços no espaço de utilizador lançando a função *mmap()* da biblioteca C. Visto que este processo é orientado a páginas de memória, o respetivo número de página é passado com argumento nessa chamada ao sistema e o *offset* é guardado para operações futuras. De salientar que este processo é realizado para cada estrutura de dados ou periférico mapeado na memória.

No espaço *kernel*, o *kernel* Linux responde realizando todas as ações necessárias para esse mapeamento e lançando a função *ADC_device_mmap()* do *driver* junto com a estrutura de dados Virtual Memory Area (VMA) criada para representar o tal mapeamento. Esta estrutura é utilizada para gerir espaços de memória virtuais distintos no espaço de utilizador. A inicialização e libertação desta estrutura são completadas pelo *device driver*, através da API do Linux referente à VMA (*vma_open()*, *vma_close()*). De seguida, o *device driver* é responsável por mapear página no espaço virtual da aplicação através da função *ioremap()*, passando o número de página correspondente ao endereço físico da estrutura de dados e área

de memória virtual para a qual a página será mapeada. Daqui em diante, até a memória ser desassociada ou até ocorrer um erro no sistema, a página contendo a estrutura de dados está associada a uma gama de endereços do espaço virtual da aplicação.

O função de mapeamento termina retornando o endereço virtual da página mapeada, ao qual tem de ser somado o *offset* calculado previamente do forma a obter o apontador para a primeira posição da estrutura de dados mapeada.

Este processo é realizado na inicialização do sistema de aquisição, resultando num *buffer* memória que pode ser acedido diretamente pela aplicação e pelo *device driver* através de índices presentes na estrutura de controlo. Estes índices compreendem o índice de escrita, incrementado sempre que uma amostra é escrita no *buffer*, e o índice de leitura, o qual é incrementado quando uma amostra é lida do *buffer*. Este *buffer* possui 1024 posições, cada uma com oito amostras, e comporta-se como um *buffer* circular com a política *first-in first-out*. Para além disto, é preciso salientar que as amostras não podem ser sobrescritas no caso da aplicação não conseguir processá-las rapidamente, o que origina a paragem do sistema.

Aquisição e Partilha de Dados

O processo de aquisição do ADC é realizado pelo periférico em hardware e completado pelo respetivo *device driver*. Já o processo de partilha de dados assenta no mecanismo de mapeamento de memória descrito na secção anterior. Na Figura 4.43 estão representadas as transações de dados entre as entidades do sistema de aquisição, onde é possível visualizar os processos de aquisição e partilha de dados.

Ao nível do hardware, o *Controlador do ADC* é responsável por definir o sinal periódico que dita o início da conversão do ADC, e por adquirir as amostras resultantes através do seu barramento de saída. Os dados adquiridos são armazenados nas respetivas memórias FIFO (uma para cada canal do ADC), e depois lidos pelo *device driver*, via barramento AXI, quando o periférico em hardware despoleta uma interrupção. Esta interrupção é gerada quando as memórias FIFO ultrapassam um certo *threshold* de dados.

No espaço *kernel*, o sistema operativo lança o *handler* de serviço à interrupção, em resposta à interrupção gerada, onde são consumidas amostras das memórias FIFO e colocadas num *buffer* partilhado com a aplicação. Se o *buffer* não se encontra

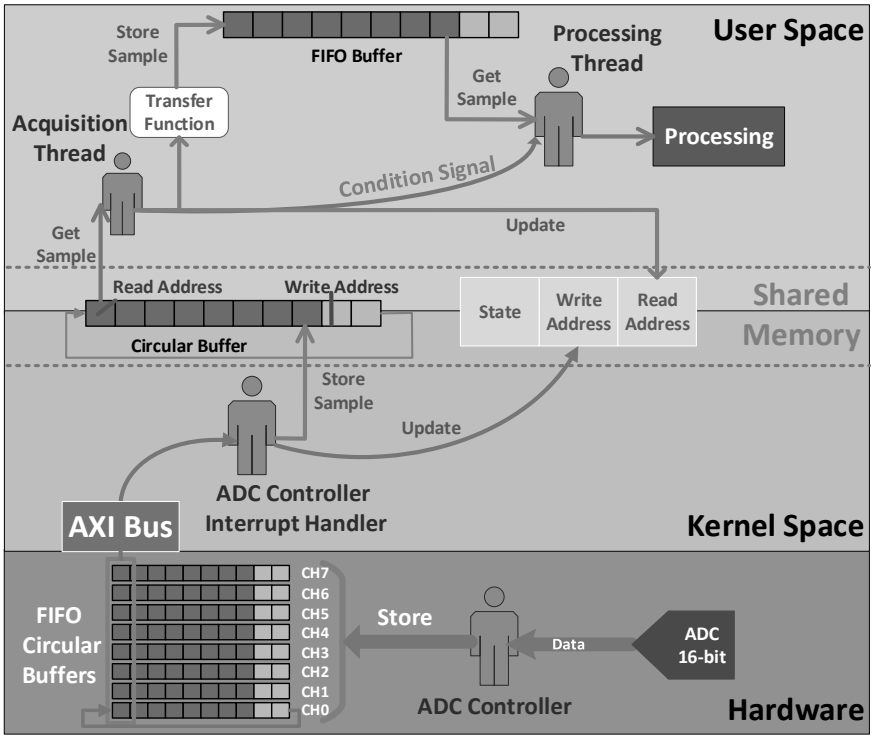


Figura 4.43: Transação de dados entre o *Controlador do ADC*, o *device driver* e a aplicação software.

totalmente preenchido armazena as amostras consumidas e atualiza o índice de escrita. A esta função de serviço à interrupção é atribuída prioridade máxima de modo a evitar a perda de dados devido À capacidade limitada das memórias dos periférico de hardware.

Acerca da *thread* de aquisição de dados, é instalado um sinal junto do *kernel* Linux para efetuar uma recolha periódica dos dados da memória partilhada quando o sistema de aquisição se encontra em execução. Este sinal usa um *real-time timer* que quando expira, o *kernel* envia um sinal `SIGALRM` para o espaço de utilizar, acordando a *thread* de aquisição de dados. Esta *thread* de aquisição de dados lê todas as amostras presentes na memória partilhada, aplica a função de transferência do ADC, guarda os resultados numa FIFO partilhada e sinaliza a *thread* de processamento.

A *thread* de processamento deve ser modelada de acordo com os objetivos e requisitos da aplicação, porém neste caso, esta *thread* apenas consome as amostras da FIFO partilhada e guarda num ficheiro de texto.

Um aspeto importante a considerar, é que este sistema está propenso a erros em certas ocasiões. Um possível cenário existe quando a *thread* de aquisição não

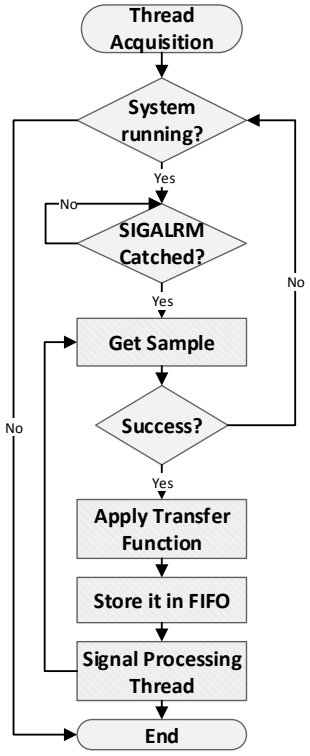


Figura 4.44: *Flowchart* da *thread* que responde ao sinal `SIGALRM`.

consome os dados mais rápido do que o *device driver* está guardá-los, ou quando o *device driver* não consome as amostras do periférico de hardware mais rápido que este as armazena, podem levar ao esgotamento da memória e à consequente perda de dados, comprometendo a fiabilidade do sistema.

4.9.4 *Thread* de Monitorização de Dados

A *thread* de monitorização é lançada pela aplicação do sistema, quando o sistema de aquisição entra em execução. Esta *thread* é responsável pela recolha de dados e pela preparação da interface gráfica onde irão ser visualizados os dados adquiridos, e pode ser interrompida pelo *handler* da interrupção do *device driver* e pela *thread* de aquisição uma vez que possuem requisitos tempo-real.

Esta *thread*, referenciada como *thread* de processamento anteriormente, espera por um sinal da *thread* de aquisição para completar o seu processamento, consumindo amostras da FIFO partilhada, adicionando-as às estruturas de dados correspondentes aos gráficos e desenhá-los quando estes alcançarem um certo número de amostras. Quando o sistema de aquisição é parado, esta *thread* termina libertando todos os seus recursos. Na Figura 4.45 está representado o comportamento desta

thread através de um *flowchart*.

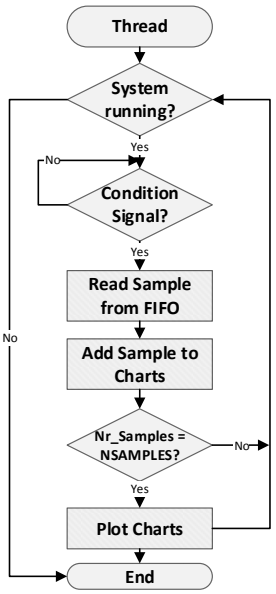


Figura 4.45: *Flowchart* das ações executadas na *thread* de monitorização de dados.

Na Secção 5.3 é possível analisar a aplicação desta *thread* no domínio dos sistemas de energia elétrica.

Capítulo 5

Cenários de Aplicação

O sistema de aquisição proposto, caracterizado no Capítulo 4, encontra-se no domínio de sistemas embebidos com requisitos temporais críticos. Como tal, o determinismo, previsibilidade e baixas latências, exigidos por este tipo de sistemas, são requisitos difíceis de satisfazer, o que leva os *designers* do sistema a encontrarem soluções que mitiguem/eliminem estes problemas de forma a garantir o correto funcionamento do sistema. Deste modo, como discutido anteriormente, a solução passa não só pela customização do *kernel* Linux, mas também pela utilização da tecnologia FPGA, resultando do *design* conjunto de *software* e *hardware* do sistema.

Isto levou à integração do sistema de aquisição desenvolvido em dois cenários de aplicação distintos, de forma a analisar as performances associadas à utilização da tecnologia FPGA da plataforma de desenvolvimento. Inicialmente, o sistema de aquisição em tempo-real foi desenvolvido num ambiente de SW-*only*, onde foram identificados os *bottlenecks* do sistema avaliando as suas performances. No cenário de aplicação seguinte, são avaliadas as performances do sistema de aquisição com uma arquitetura híbrida, na medida que combina a aceleração em *hardware* com as funcionalidades do CPU formando uma solução viável no domínio das aplicações. De seguida, é efetuada uma comparação dos resultados obtidos de cada um destes cenários de aplicação averiguando as suas performances e funcionalidades.

Por fim, sistema híbrido de aquisição em tempo-real baseado em Linux é incorporado no domínio dos sistemas de energia elétrica de forma a verificar e validar as funcionalidades do sistema.

5.1 Sistema de Aquisição em SW-*only*

Como descrito no capítulo anterior, os principais elementos do sistema de aquisição proposto são a placa de aquisição de sinal, a plataforma de desenvolvimento e a aplicação de software.

O primeiro cenário de aplicação consiste na análise e avaliação da performance do sistema de aquisição em tempo-real integrado num ambiente de SW-*only*. Este ambiente envolve a implementação de todas as tarefas realizadas pelo sistema de aquisição de dados puramente em *software*, desde a obtenção dos dados do *hardware* de aquisição de sinal, até à sua disponibilização no espaço da aplicação, tirando proveito apenas da unidade de processamento da plataforma. Na Secção 4.4, foi feita a caracterização deste sistema de aquisição de SW-*only*.

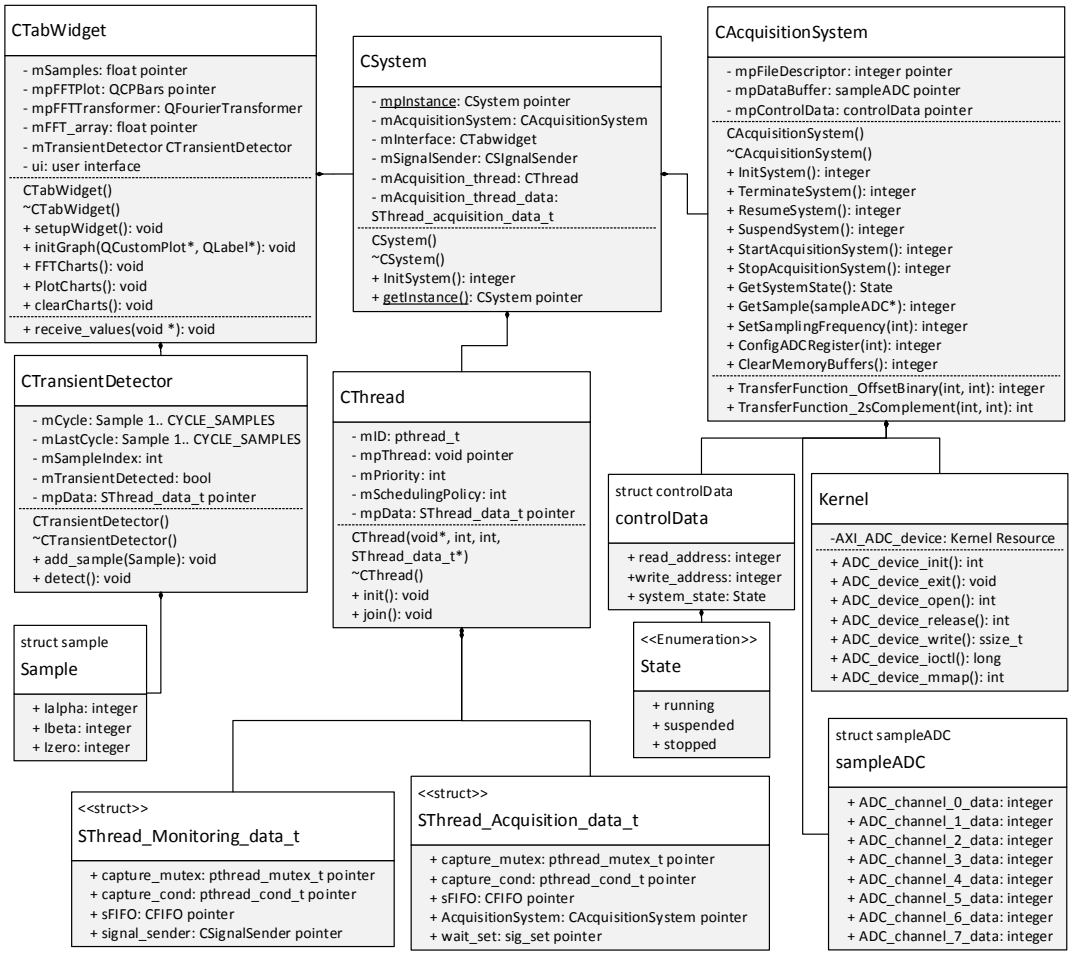


Figura 5.1: Diagrama UML da aplicação software.

No diagrama UML da Figura 5.1 é possível observar as entidades referentes ao sistema de aquisição em conjunto com a aplicação alvo no domínio dos sistemas

de energia elétrica, o qual é caracterizado nas próximas secções. Mediante a integração da aplicação alvo, foram adicionas as respetivas classes ao diagrama UML. Uma destas classes é responsável pela preparação e exibição da interface gráfica, designada de "CTabWidget". Por sua vez, a classe "CTransientDetector", membro da classe superior "CTabWidget", contém as estruturas de dados resultantes da transformada de Clarke, utilizada para a deteção de eventos. Neste cenário de aplicação, caracterizado nas próximas secções, a aplicação foi desenhada com a capacidade deteção de eventos momentâneos de qualidade de energia elétrica através do uso da transformada de Clarke, contudo, devido à falta de recursos não foi concluída a sua implementação.

Uma das técnicas que permite a identificação de tarefas de software críticas no processamento, consiste no *profiling* da aplicação de software. Esta técnica consiste num processo de amostragem estatística de forma a traçar o perfil da aplicação em termos de tempo de CPU consumido por cada tarefa que a constitui, com o objetivo de melhorar a performance do sistema. No entanto, foram utilizados *hardware performance counters* para analisar a performance da aplicação software, medindo os tempos de execução de cada tarefa em *clock cycles*. Estes *hardware counters* foram instanciados no FPGA-*fabric* de modo a guardar as contagens de *clock cycles* de certas tarefas da aplicação.

Assim sendo, foi possível identificar como tarefas críticas do sistema: o controlo do hardware de aquisição de sinal (ADC); a função de transferência do ADC; e a transformada de Clarke.

De salientar que o controlo do ADC está dependente das especificações temporais do próprio dispositivo de aquisição. E ainda, a função de transferência e a transformada de Clarke foram implementas recorrendo à aritmética de inteiros. Esta técnica de otimização consiste em representar valores com vírgula flutuante por valores inteiros nas operações matemáticas envolvidas na realização da função de transferência e da transformada de Clarke, uma vez que os cálculos efetuados com valores de vírgula flutuante podem consumir uma quantidade significativa de tempo do CPU. Assim sendo, esta técnica introduziu uma melhoria na performance no sistema à custa da precisão associada aos dados resultantes de ambas as operações.

5.2 Sistema Híbrido de Aquisição

No segundo cenário de aplicação, as principais tarefas de software identificadas como críticas no processamento foram aceleradas em hardware dedicado através dos recursos da plataforma de desenvolvimento, unificando os domínios de hardware e software. Este processo de aceleração foi realizado de uma forma individual, de modo a aferir o impacto na performance global do sistema de cada tarefa. Os resultados do *offloading* por computação destas tarefas críticas são apresentados posteriormente.

5.3 Sistema de Monitorização

Este cenário de aplicação consiste na monitorização da qualidade de energia elétrica de uma instalação elétrica trifásica utilizando o sistema de aquisição em tempo-real desenvolvido.

Este sistema de aquisição pode ser usado para monitorizar e registar eventos que degradem a qualidade de energia elétrica do sistema elétrico. No Anexo C são descritos, de uma forma sucinta, os problemas de qualidade de energia elétrica que podem ter danos graves nas instalações elétricas.

O sistema de aquisição e a aplicação de monitorização correm sob o mesmo ambiente, na plataforma de desenvolvimento descrita no Capítulo 3. No que diz respeito ao sistema de aquisição, os transdutores, o acondicionamento de sinal e o ADC são os elementos responsáveis pela aquisição dos sinais da instalação elétrica no formato adequado para a plataforma. O controlo do sistema é realizado pela consola terminal (através do recurso USB-UART), onde são enviados comandos específicos a operação a executar. No que concerne à aplicação de monitorização, esta é executada no modo utilizador e tem como função exibir os dados correspondentes aos sinais a serem monitorizados através de uma interface gráfica num ecrã LCD. Assim sendo, o sistema de aquisição juntamente com a aplicação de monitorização funcionam como um sistema de monitorização completo, cuja estrutura e componentes são descritos nas secções que se seguem.

Este sistema de monitorização foi conectado a uma instalação elétrica trifásica com neutro, com o intuito de monitorizar as formas de onda das tensões e correntes. Na Figura 5.2 estão representadas as conexões efetuadas entre o sistema de

monitorização e a instalação elétrica. Assim, o sistema de aquisição tira partido do número total de canais do ADC para monitorizar as tensões simples (V_{an}, V_{bn}, V_{cn}), tensão neutro-terra (V_{nPE}) e as correntes nas fases e neutro (I_a, I_b, I_c, I_n).

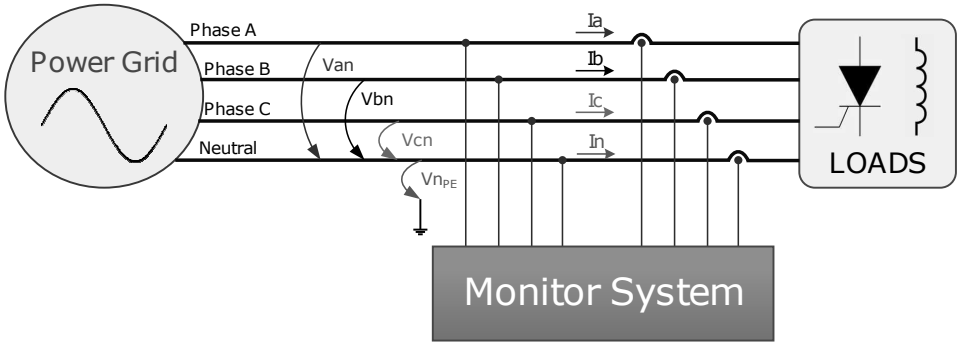


Figura 5.2: Esquemático das conexões entre o sistema de monitorização e uma instalação elétrica.

5.3.1 Constituintes do Sistema de Monitorização

Como ilustrado na Figura 5.3, o sistema de monitorização é constituído por três entidades principais: a placa de aquisição de sinal responsável pela aquisição dos dados de uma instalação elétrica no formato adequado para o sistema computacional; a plataforma de desenvolvimento sobre a qual são executados o sistema de aquisição e a aplicação de monitorização; e um ecrã LCD com entrada VGA.

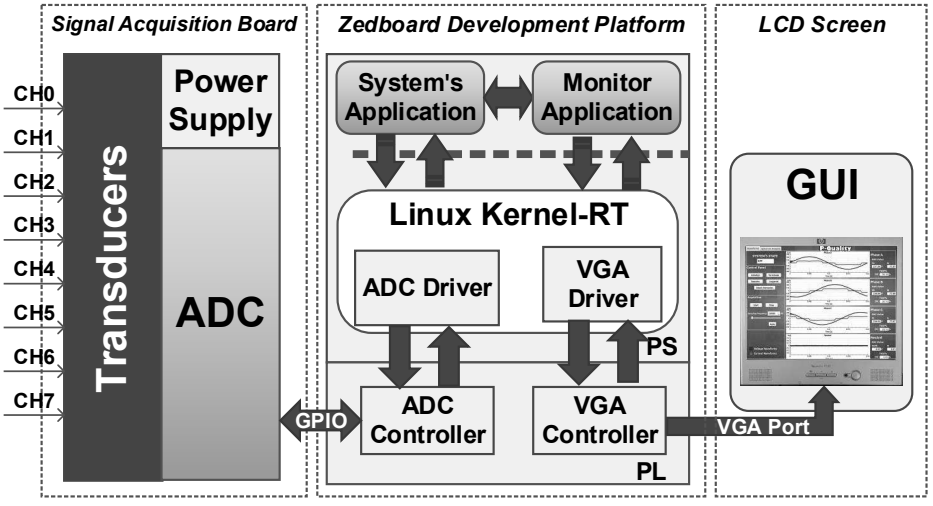


Figura 5.3: Principais constituintes e conexões do sistema de monitorização.

O sistema de monitorização adquire os dados através do ADC, cujo barramento paralelo de saída e sinais de controlo são endereçados através dos portos GPIO da

plataforma de desenvolvimento. Os processos de aquisição e armazenamento de dados e controlo do ADC são realizados pelo controlador em *hardware*, que por sua vez é controlada pelo respetivo *device driver*, como discutido previamente. Para além destes componentes, o sistema possui uma aplicação, executada no espaço de utilizador, que não só permite o controlo do sistema conforme os comandos recebidos no terminal, como também partilha dados com a aplicação de monitorização de forma a serem exibidos no ecrã LCD. Acerca deste LCD, foram utilizados um logicBricks IP *core* em conjunto com os respetivos *device drives* da Xylon [20], os quais são responsáveis pela leitura de dados da memória do sistema (*framebuffer*) e por fazer a interface com o LCD através do conector VGA, exibindo a GUI.

5.3.2 Organização da Memória do Sistema de Monitorização

Visto que o sistema de aquisição e a aplicação de monitorização se encontram no mesmo ambiente computacional, o processo de comunicação foi bem simples. Como tal, ambas as aplicações foram implementadas recorrendo ao paradigma de programação *multithreading* usando a API das *threads* POSIX. A organização de memória do sistema de monitorização, ilustrada na Figura 5.4, é bastante semelhante àquela ilustrada na Figura 4.44.

A controlador do ADC em hardware realiza o processo de leitura do ADC e subsequentemente guarda os dados adquiridos em blocos de memória no FPGA-*fabric* funcionando como um meio de armazenamento secundário. Quando este elemento de armazenamento chega a uma determinada percentagem de utilização, o controlador do ADC despoleta uma interrupção com o objetivo de notificar o sistema operativo para consumir amostras dos blocos de memória. Este processo é efetuado pelo respetivo *device driver*, cujo *handler* registado lê os dados das memórias e coloca-os num *buffer* de memória partilhado entre o espaço *kernel* e o espaço utilizador.

Em espaço de utilizador, a *thread* de aquisição de dados realiza uma recolha periódica dos dados da memória partilhada, aplica a função transferência do ADC e coloca a amostra resultante numa memória FIFO. Esta memória pode ser acedida pela *thread* de aquisição e a *thread* de monitorização. O mecanismo IPC utilizado entre estas duas *threads* é *condition variables*, em que a *thread* de monitorização fica à espera que a *thread* de aquisição envie um *condition signal* depois de efetuar

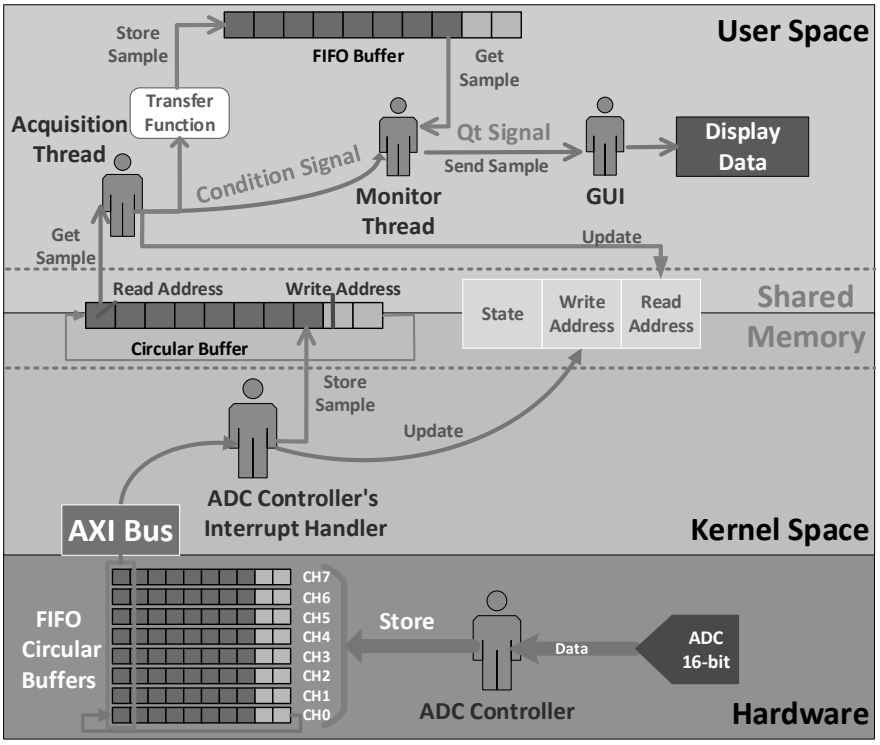


Figura 5.4: Diagrama de interações do sistema de monitorização.

uma leitura. Quando a *thread* de monitorização recebe o sinal consome uma amostra e envia-a para a GUI, usando o conceito de *signal/slot* da Qt *framework*, onde irão ser exibidos os dados na forma de gráficos. Esta interface gráfica só atualiza os seus gráficos quando o número de amostras chega a um valor equivalente a dois ciclos de rede elétrica.

5.3.3 *Graphical User Interface*

Como mencionado anteriormente, a aplicação de monitorização possui uma interface gráfica também integrada na plataforma de desenvolvimento. Contudo, esta GUI tem como único propósito mostrar os dados adquiridos ao utilizador, deixando o controlo do sistema de monitorização ser efetuado pela consola terminal usando o recurso USB-UART da plataforma. Nas Figuras 5.5 e Figure 5.6 está ilustrada a interface gráfica baseada na Qt Creator *framework* da Digia Co. [4].

Como ilustrado na Figura 5.5, é possível visualizar as formas de onda das tensões e correntes de um sistema trifásico com neutro, para fins de monitorização da qualidade de energia elétrica. Também é possível visualizar os valores eficazes de cada tensão e corrente bem como a Total Harmonic Distortion (THD) das

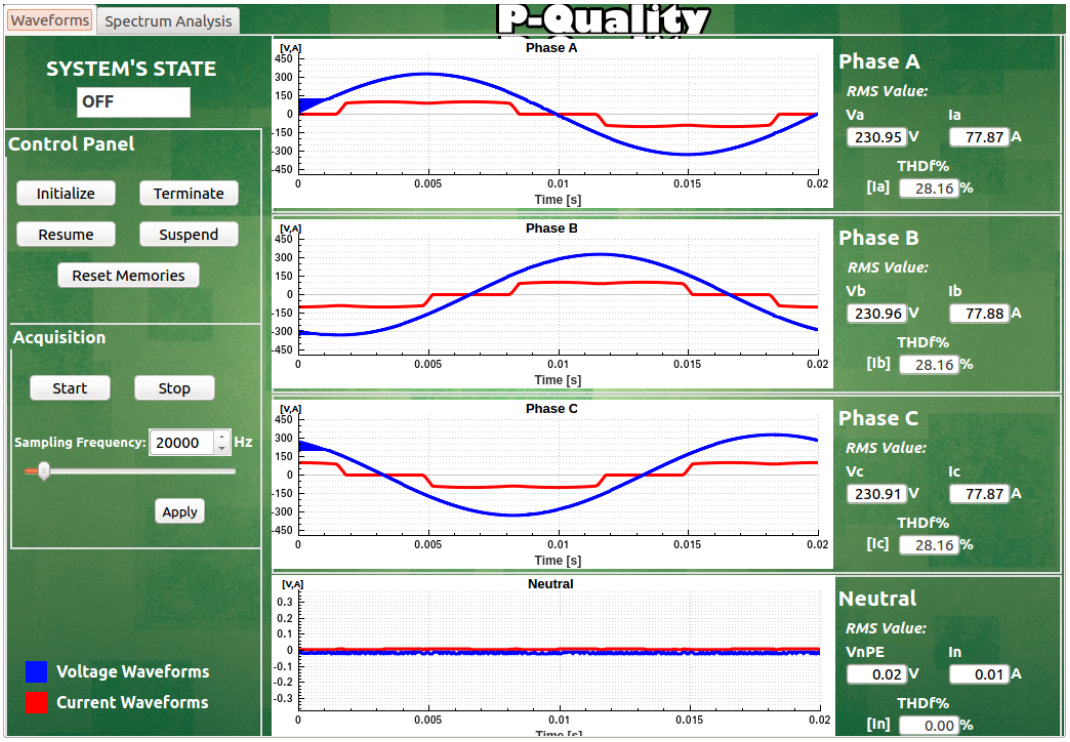


Figura 5.5: *Graphical user interface* - formas de onda.

correntes.

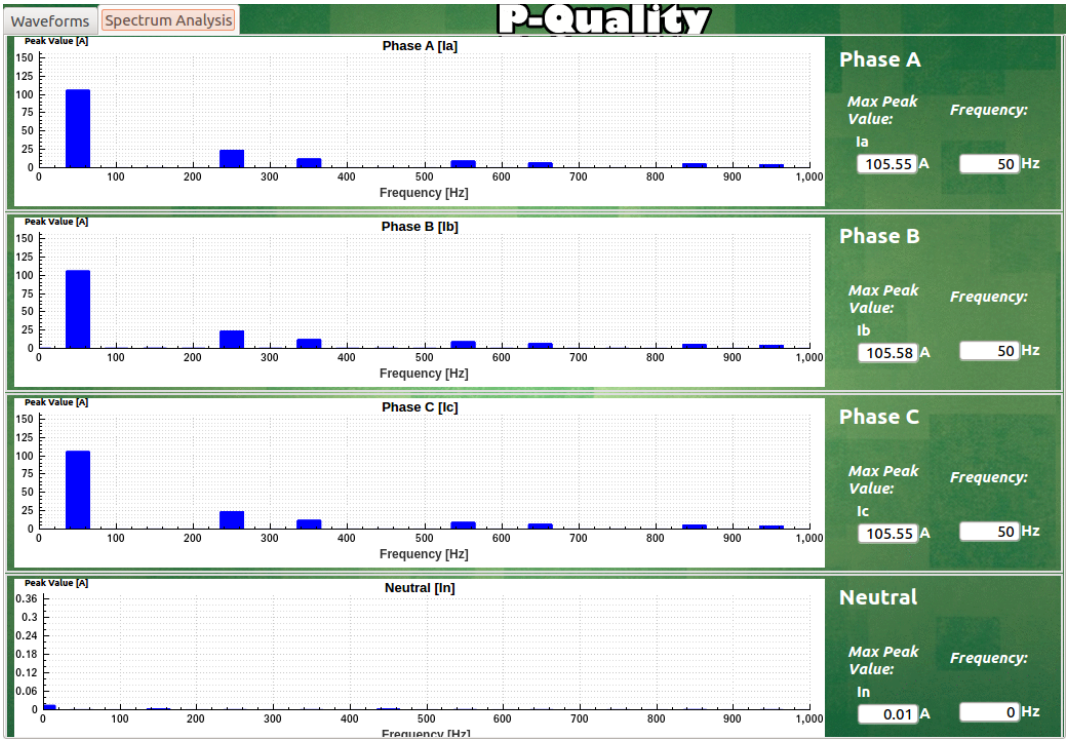


Figura 5.6: *Graphical user interface* - FFT.

No segundo separador da interface, ilustrado na Figura 5.6, são exibidos os gráficos correspondentes à análise do espetro das formas de onda das correntes através de um algoritmo FFT simples. Esta análise permite representar as formas de onda das correntes no domínio da frequência de forma a analisar o seu conteúdo harmónico.

5.4 Recolha de Resultados

Nesta secção são apresentados os resultados que verificam e validam as funcionalidade, serviços performances da tecnologia do sistema híbrido de aquisição em tempo-real desenvolvido.

No que diz respeito ao cenário de aplicação referente ao sistema de aquisição em software, foi analisada a sua performance segundo a métrica de tempos de execução. Esta análise consiste no cálculo do tempo de execução médio de cada parte e o respetivo desvio padrão. Assim sendo, foram realizados vários testes, através de *hardware counters*, a certas tarefas do sistema descritas anteriormente.

Na Figure 5.7 estão representados a média e o desvio padrão (barra de erro) dos tempos de execução das funções selecionadas do sistema de aquisição.

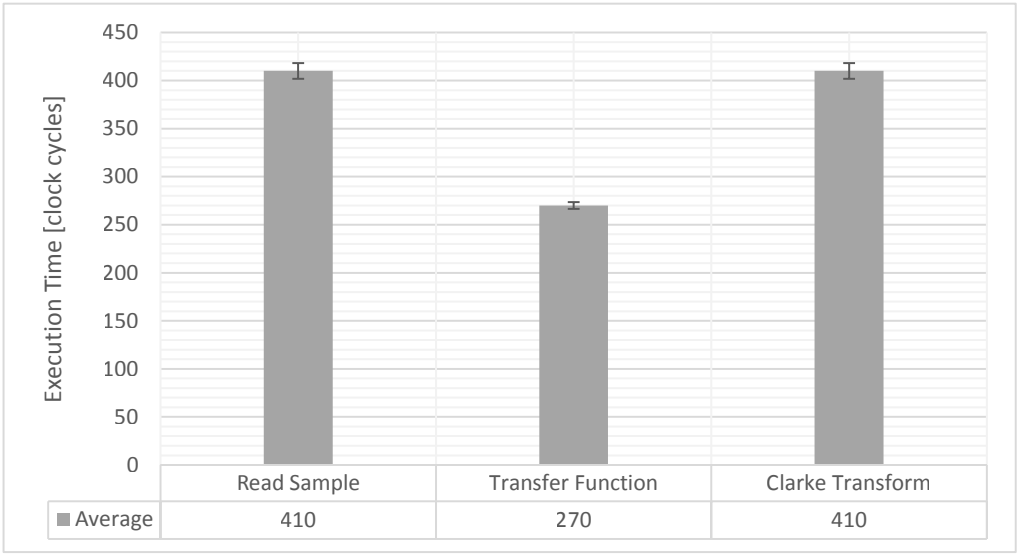


Figura 5.7: Tempos de execução de certas partes do sistema de aquisição SW-*only*.

Como ilustrado na figura, demorou em média 420 *clock cycles* para ler uma amostra de um canal do ADC, pelo que a leitura das amostras de todos os canais do ADC levava um tempo à volta de oito vezes mais. Para realizar todos os cálcu-

los associados à função de transferência do ADC e à transformada de Clarke foi necessário em média 270 *clock cycles* e 410 *clock cycles* para serem completados.

Estes tempos de execução podem ser influenciados pelo estado do processador de várias formas e pela unidade de memória. Em relação aos processadores, as suas unidades de *pipeline* são fontes de variação, visto que o tempo de execução de uma instrução depende em parte do comportamento de outras instruções no *pipeline*, como no caso de duas instruções competirem pelo mesmo recurso executarem mais lentamente. No que diz respeito às unidades de memória, estas podem introduzir variações nos acessos à memória *cache* dependendo se ocorreu um *cache hit* ou *cache miss*.

Desta forma, o sistema de aquisição em software-*only* permitiu efetuar amostragens até cerca de 40 kHz, pelo que a partir desta gama de frequências os recursos de memória partilhada esgotavam-se levando à paragem do processo de aquisição.

Tal como o cenário anterior, no caso do sistema híbrido de aquisição é também efetuada uma análise à performance consoante os respetivos tempos de execução. Estes testes foram realizados através da simulação de *testbenches* criados para cada uma das funções definidas, desprezando o atraso na propagação do sinal na lógica do FPGA-*fabric*.

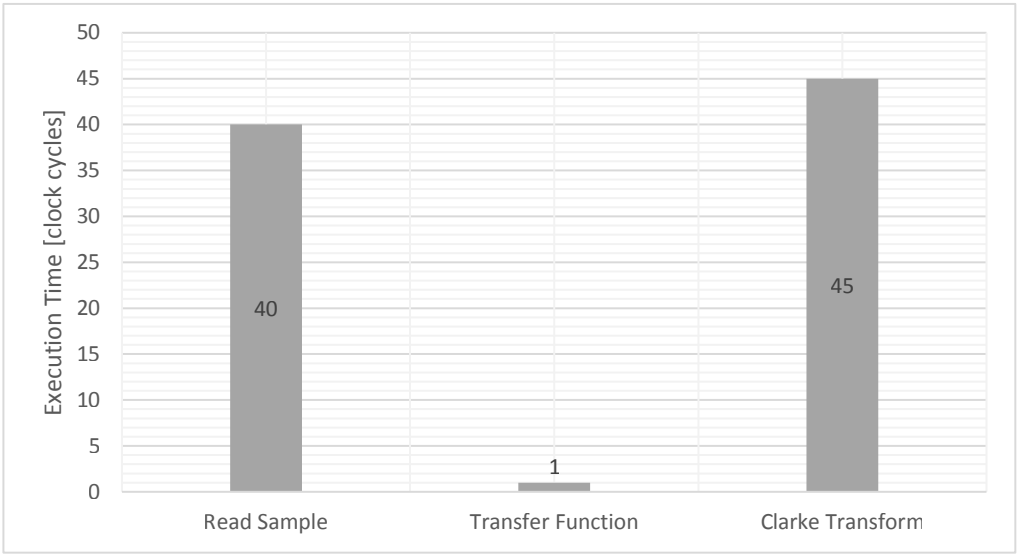


Figura 5.8: Tempos de execução do sistema híbrido de aquisição.

Os resultados apresentados na Figura 5.8 mostram um tempo de execução de 40 *clock cycles* na leitura de uma amostra do ADC, uma execução de um *clock cycle* da função de transferência devido à utilização de blocos de hardware dedicados

para as respetivas operações matemáticas, e aproximadamente 45 *clock cycles* para realizar a transformada Clarke.

Comparando os resultados apresentados nas Figuras 5.7 e Figure 5.8, é verificada uma redução nos tempos de execução das três partes, pelo que originou um melhoramento na performance global do sistema.

Em relação ao sistema híbrido de aquisição, este permitiu realizar a aquisição de sinais até uma frequência na ordem dos 70 kHz, apresentando fiabilidade nos resultados. A partir desta gama de frequências, o sistema não consegue processar as amostras das memórias do controlador em hardware mais rápido que as adquire, levando ao esgotamento destas memórias e consequentemente à perda de informação do mundo real.

No que concerne ao sistema de monitorização descrito anteriormente, a aquisição dos dados resultantes do ADC das componentes de um sistema elétrico trifásico através da placa de aquisição foi emulada em hardware, uma vez que não foi possível concluir a respetiva PCB. Como tal, este periférico de hardware é responsável por emular o comportamento do ADC e por simular os valores convertidos do mundo real. Os resultados da aquisição de dados através deste emulador, referentes às componentes de uma fase da instalação elétrica, podem ser visualizados na Figura 5.9.

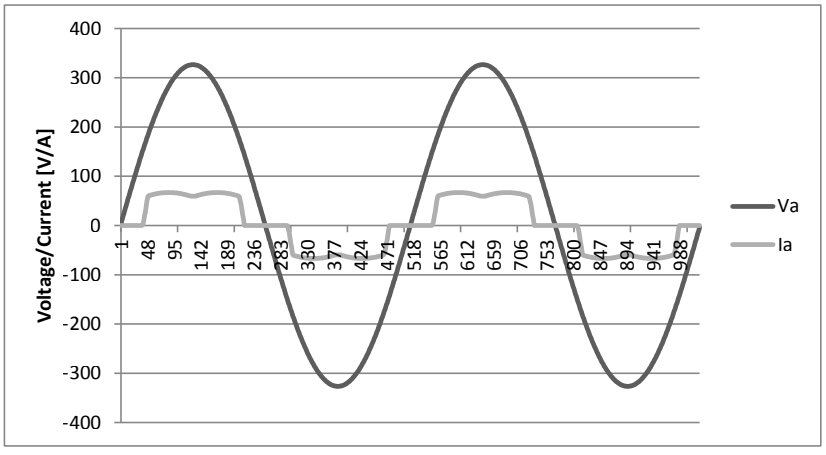


Figura 5.9: Dados emulados da tensão e corrente da fase "A" de um sistema elétrico adquiridos pelo sistema híbrido de aquisição.

A integração da aplicação de monitorização influenciou a gama de frequências de amostragem do sistema de aquisição, devido à exigência computacional associada à exibição de uma interface gráfica, pelo que limitou a frequência de amostragem máxima para a ordem de 35 kHz.

Capítulo 6

Conclusões e Discussão de Resultados

Com o rápido desenvolvimento dos sistemas de monitorização/controlo nos vários setores da indústria moderna, surge a necessidade de sistemas de aquisição em tempo-real mais rápidos, fiáveis, com maiores resoluções que façam a interface entre o mundo real e o mundo digital para um vasto leque de aplicações.

O sistema de aquisição em tempo-real desenvolvido pretende satisfazer estes requisitos de forma a ser utilizado no controlo e monitorização de processos industriais. Ao longo do seu desenvolvimento, foram adquiridas capacidades face à utilização de um sistema operativo como o Linux, ao *patching* do seu *kernel* com a vertente de tempo-real, ao desenvolvimento de *device drivers* e à implementação de mecanismos de comunicação entre o espaço do *kernel* e o espaço de utilizador recorrendo à linguagem de programação C. Para além disso foram obtidos conhecimentos no desenvolvimento de aceleradores em hardware para o *offload* de componentes de software críticas, recorrendo a linguagens de descrição de hardware na tecnologia FPGA.

Este foi um projeto muito desafiante que combinou os domínios de Sistemas Embebidos e de Eletrotecnia e Sistemas de Energia. Alguns dos objetivos inicialmente estipulados não foram cumpridos, nomeadamente no que diz respeito à implementação da PCB de aquisição de sinal. Nas próximas secções é discutido o trabalho desenvolvido bem como as perspetivas para trabalho futuro.

6.1 Trabalho Desenvolvido

O sistema de aquisição em tempo-real desenvolvido é um sistema parametrizável capaz de obter a informação relevante do mundo real e disponibilizá-la ao utilizador/programador, de maneira que os dados recolhidos possam ser analisada e processada de acordo com a aplicação alvo. Este sistema permite a aquisição simultânea de oito sinais do mundo real com uma resolução de 16-bits até uma frequência de amostragem limitada pelo hardware de aquisição de sinal.

De modo a analisar as potencialidades em termos de tempos de execução da tecnologia FPGA, este sistema foi integrado, inicialmente, num ambiente software-*only*, e posteriormente, num cenário onde é efetuada a unificação dos domínios de software e hardware. Os resultados permitiram concluir que houve uma redução significativa nos tempos de execução de algumas funcionalidades.

De seguida o sistema de aquisição foi incorporado no domínio dos sistemas de energia elétrica, foram adquiridos os dados referentes às tensões e correntes elétricas de uma instalação para efeitos da monitorização de qualidade de energia elétrica. Neste cenário de aplicação, a gama de frequências de amostragem é condicionada pela intensidade computacional da aplicação de software integrada, uma vez que o sistema de aquisição e a aplicação alvo disputam o recurso CPU. O sobrecarregamento do CPU por parte da aplicação pode levar o sistema de aquisição a perder informação, devido ao facto da aplicação não conseguir processar os dados. Isto é, se a aplicação não progride no processamento dos dados, os recursos de memória irão eventualmente esgotar originando perda de informação.

Como discutido no Capítulo 5, a aplicação de software com uma interface gráfica, consome muito tempo de CPU, principalmente a preparar a composição visual da interface. Neste caso, também são realizados os cálculos dos valores eficazes e os de THD, bem como um simples algoritmo da FFT. Assim, todo este processamento integrado com o sistema de aquisição limita a frequência de amostragem que pode ser usada para garantir a operacionalidade do sistema. Todavia, para efetuar uma análise precisa da qualidade de energia elétrica de uma instalação elétrica, é apenas necessário analisar as respetivas formas de onda de tensão e corrente elétricas dentro de uma gama de frequências até ao sétimo harmónio (abaixo de 1 kHz), visto que para além destas frequências o conteúdo harmónico é considerado desprezável.

6.2 Trabalho Futuro

Como trabalho futuro, o sistema de aquisição em tempo-real desenvolvido pode ser reutilizado numa aplicação alvo diferente. Relativamente ao *offloading* por computação em FPGA, este sistema encontra-se dependente da interface com barramento AXI presente nas plataformas da família Xilinx 7-series. Para além disto, graças à portabilidade proporcionada por um ambiente de Linux, este sistema permite a integração de plataformas alvo independemente da arquitetura do processador.

De modo a integrar este sistema de aquisição noutros cenários de aplicação, é apenas necessário desenvolver a respetiva placa de aquisição de sinal, com os transdutores e acondicionamento de sinal, e integrar a aplicação alvo na plataforma.

Bibliografia

[1] J. Afonso and J. Martins, "Qualidade da energia eléctrica," *Revista o Eletricista, no. 9*, pp. 66–71, 2004. [Online]. Available: http://repositorium.sdum.uminho.pt/handle/1822/1920

[2] Avnet Inc., "Zedboard." [Online]. Available: http://zedboard.org/

[3] Buildroot, "Buildroot - Making Embedded Linux Easy." [Online]. Available: http://buildroot.uclibc.org/

[4] Digia Co., "Digia Plc." [Online]. Available: http://www.digia.com/

[5] C. Hallinan, *Embedded Linux Primer: A Practical Real-World Approach*, 2nd ed. Prentice Hall, 2006. [Online]. Available: http://www.amazon.com/Embedded-Linux-Primer-Practical-Real-World/dp/0131679848

[6] LEM Co., "LEM - current transducer, voltage transducer, sensor, power measurement." [Online]. Available: http://www.lem.com/

[7] C. Maxfield, *FPGAs: World Class Designs*. Newnes, 2009. [Online]. Available: http://www.fs.fed.us/ne/newtown{_}square/publications/research{_}papers/pdfs/scanned/OCR/ne{_}rp351.pdf

[8] Maxim, "MAXIM MAX11046 ADC Datasheet," pp. 1–28, 2013.

[9] Maxim Co., "Analog, linear, and mixed-signal devices." [Online]. Available: http://www.maximintegrated.com/en.html

[10] National Instruments, "National Instruments: Test, Measurement, and Embedded Systems." [Online]. Available: http://www.ni.com/

[11] P. Oliveira, "Sistema de Aquisição de Sinais em Tempo Real Baseado em Linux," Master's Thesis, University of Minho, 2013.

[12] R. Sass and A. Schmidt, *Embedded Systems Design with Platform FPGAs Principles and Practices*. Morgan Kaufmann, 2010.

[13] K. Schipman and F. Delincé, "The importance of good power quality," -, pp. 1–20, 2010.

[14] V. Silva, T. Malheiro, J. A. Mendes, J. Cabral, and A. Tavares, "Real-time low-cost industrial acquisition system," *IEEE International Conference on Industrial Informatics (INDIN)*, pp. 763–767, 2011.

[15] V. Silva, "Sistema de Aquisição de Dados Tempo Real baseado em Linux," Master's Thesis, University of Minho, 2011.

[16] W. Stallings, *Computer Organization and Architecture*, 9th ed. Prentice Hall, 2012.

[17] G. Started, I. Communication, and M. Linux, *Advanced Linux Programming*, 1st ed. New Riders Publishing, 2001, vol. 49, no. 4. [Online]. Available: http://portal.acm.org/citation.cfm?id=558873

[18] Xilinx Inc., "All Programmable Technologies." [Online]. Available: http://www.xilinx.com/

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

[20] Xylon, "Xylon - logicBRICKS IP Cores for FPGA and Design Services." [Online]. Available: http://www.logicbricks.com/

[21] K. Yaghmour, "Building Embedded Linux Systems," *Chemistry*, p. 416, 2008. [Online]. Available: http://books.google.de/books?id=I-hVqkX6A9cC{&}dq=Yaghmour+Embedded+Linux{&}source=gbs{_}navlinks{_}s

Apêndice A

Customização do *Kernel* Linux com a vertente *Real-Time*

O *kernel* Linux possui uma *patch* que o torna num *kernel fully preemptible*, conferindo um melhoramento no determinismo, previsibilidade e baixas latências no sistema operativo. Estas características possibilitam a execução de aplicações com requisitos temporais exigentes. De seguida, são apresentados os passos que descrevem como aplicar esta *RT-patch* ao kernel 4.0.4 do Linux, com a ajuda da ferramenta Buildroot.

A.1 Como aplicar a *Real-Time Patch* ao Linux

O primeiro passo é obter os ficheiros fonte do kernel Linux, pelo que o Buildroot permite selecionar no seu menu de configuração.

Buildroot's configuration menu for setting kernel version:

```
Buildroot 2014.08 Configuration
  Kernel -->
         Kernel version (Custom version) -->
      (4.0.4) Kernel version
```

O segundo passo é fazer o *download* da *RT-patch*:

`https://www.kernel.org/pub/linux/kernel/projects/rt/4.0/older/patch-4.0.4-rt1.patch.gz`

O próximo passo é extrair os ficheiros descarregados e aplicar a *RT-patch* ao *kernel* Linux executando o seguinte comando na diretoria de raíz do *kernel*:

```
$ patch -p1 < $PATH_DIRECTORY/patch-4.0.4-rt1.patch
```

Após o *patching* do código fonte do *kernel*, devem ser habilitadas as seguintes funcionalidades no menu de configuração do *kernel* Linux executando:

```
$ make menuconfig
$ make linux-menuconfig #when located in the Buildroot root
  directory
```

Devem ser selecionadas as seguintes opções no menu de configuração: "*Fully Preemptible RT-Kernel*", "*High-Resolution Timer*" e "*Timer Frequency*" para 1000 Hz.

Linux kernel's configuration menu for setting kernel features:

```
Linux/arm 4.0.4 Kernel Configuration
  Kernel features -->
   Preemption Model (Fully Preemptible Kernel (RT)) -->
   Timer frequency (1000 Hz) -->
```

Linux kernel's configuration menu for enabling high resolution timer support:

```
Linux/arm 4.0.4 Kernel Configuration
  General setup -->
   Timers subsystem -->
    [*] High Resolution Timer Support
```

Para concluir, o *kernel* tem de ser compilado e instalado normalmente através da execução do comando `make`.

Apêndice B

Criação de um Ambiente de Desenvolvimento Embebido

De forma a executar um sistema embebido baseado em Linux na plataforma alvo, é necessário o conjunto correto de ficheiros de acordo com a sua arquitetura que pode ser gerado com a ferramenta Buildroot.

Esta ferramenta automatiza o processo de criação deste conjunto de ficheiros criando um ambiente de desenvolvimento embebido para a plataforma alvo, incluindo a geração da toolchain para *cross-compiling*, sistema de ficheiros, *device tree*, bootloader e a imagem do *kernel* Linux. Além disto, o Buildroot suporta várias arquiteturas de CPU no que diz respeito à *toolcahin*, e um grande número de aplicações e bibliotecas no espaço de utilizador.

A sequência de passos apresentada de seguida destina-se à geração da toolchain, da imagem do *kernel* Linux e do sistema de ficheiros da plataforma Zedboard Zynq-7000 a partir da diretoria raíz da ferramenta. Neste tipo de arquiteturas CPU+FPGA, a *device tree* deve ser gerada pelas ferramentas da Xilinx.

B.1 Buildroot *Quick Start*

O primeiro passo é a configuração do Buildroot de acordo com a máquina alvo. Os ficheiros de configuração pré-definidos encontram-se na diretoria "*config/*", pelo que a configuração para a plataforma Zedboard é feita executando:

```
$ make zedboard_defconfig
```

Após isto, o Buildroot é configurado para gerar todos os ficheiros necessários para a plataform alvo, porém, é possível alterar esta configuração através de:

```
$ make menuconfig
```

O último passo é a geração de *cross-compilation toolchain*, o sistema de ficheiros, a imagem do *kernel* Linux para a máquina alvo compilando o Buildroot através do comando:

```
$ make
```

Os ficheiros gerados pelo Buildroot encontram-se na diretoria *output/images/*.

Toda esta informação foi extraída do manual do Buildroot. Para mais detalhes é necessário consultar o manual referido [3].

Apêndice C

Problemas de Qualidade de Energia Elétrica

Hoje em dia, os problemas de qualidade de energia elétrica levantam uma séria preocupação nas empresas de produção, distribuição e transporte de energia. Uma preocupação que tem vindo a aumentar devido ao surgimento de novas cargas sensíveis às variações da qualidade de energia elétrica, ao aumento da exigência na eficiência da qualidade de energia elétrica, ao conhecimento destes assuntos por parte do utilizador final e devido às consequências destes problemas nas instalações elétricas. Por isso, têm sido desenvolvidas soluções que permitem ultrapassar estes problemas, contribuindo para uma melhoria na qualidade de energia elétrica e, por consequência, originar um aumento na eficiência dos equipamentos elétricos.

Por definição, uma instalação elétrica com uma qualidade de energia elétrica fiável é alcançada quando:

- Existe uma fonte constante de energia elétrica;
- As frequências das tensões e correntes situam-se dentro da gama admitida pelo fornecedor de energia elétrica;
- As formas de onda das tensões e correntes são aproximadamente sinusoidais em qualquer ponto da instalação elétrica;
- O desfasamento entre a tensão e a corrente de cada fase encontra-se dentro dos valores máximos admitidos.

Por outro lado, uma fraca qualidade de energia elétrica pode levar à deterioração

das cargas da respetiva instalação elétrica. Na maioria dos casos, o problema não se encontra na própria instalação elétrica, contudo o seu correto dimensionamento pode mitigar algumas das suas consequências. Os pontos seguintes representam as principais consequências destes problemas de qualidade de energia elétrica [13]:

- Falhas inesperadas nas fontes de energia elétrica (e.g., disparo de disjuntores, atuação de fusíveis);
- Falha ou funcionamento indevido do equipamento;
- Sobreaquecimento do equipamento elétrico (e.g., condensadores, transformadores, máquina elétricas) levando à redução do seu tempo de vida;
- Avaria nas cargas sensíveis, como PCs e sistemas de controlo nas linhas de produção;
- Interferências nas comunicações eletrónicas;
- Aumento das perdas energéticas do sistema e consequente diminuição da sua eficiência;
- Necessidade de dimensionamento da instalação elétrica de forma a suportar as consequências de problemas elétricos adicionais;
- Abalo da sensação visual provocada por uma luz cuja intensidade oscila com o tempo (*flicker*).

Os maiores problemas que originaram um défice na qualidade de energia elétrica de uma instalação elétrica e os seus causadores podem ser encontrados de seguida:

- Desequilíbrios do sistema elétrico, visíveis no valor da corrente do neutro, devido ao dimensionamento errado do próprio sistema, causando um desequilíbrio na tensão de alimentação das cargas da instalação elétrica. Este desequilíbrio é bastante comum em sistemas trifásicos onde sejam conectadas apenas cargas monofásicas;
- Poluição harmónica causada pela existência de cargas não-lineares, como fontes de alimentação, cicloconversores, inversores de corrente ou tensão e acionadores de frequência variável;
- Interferência eletromagnética (ruído de alta frequência) produzido pelas rápidas comutações dos conversores eletrónicos de potência;

- Interrupções no fornecimento de energia elétrica na ocorrência de curto-circuitos. Por exemplo, se o sistema elétrico possuir um disjuntor com reinício automático, este abre o circuito quando ocorre um curto-circuito, e volta a fechar passados alguns milissegundo, repetindo o processo até o curto-circuito estive extinto.;

- Variações intermitentes de certas cargas causa varições nas tesões de alimentação de outras cargas, podendo originar o *flicker*;

- Ligação de banco de condensadores ou a ocorrência de relâmpagos resulta em eventos transitórios;

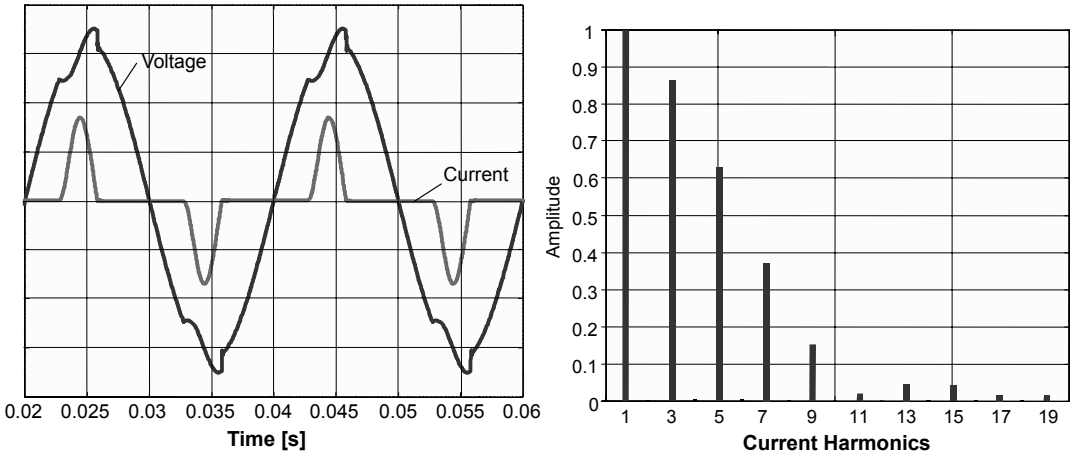
- Sobretensões e subtensões momentâneas podem ser causados por condições de falha na ligação de equipamento elétrico na instalação e pela ocorrência de curto-circuitos com a atuação automática de um disjuntor, respetivamente.

A principal causa destes problemas de qualidade de energia elétrica está relacionada com a existência de cargas não-lineares devido ao seu consumo de correntes com formas de onda distorcias, afetando as cargas mais sensíveis a estas variações. Estas cargas não-lineares podem causar problemas de qualidade de energia elétrica severos, originando a poluição do sistema elétrico com correntes com harmónicos de várias ordem, incluindo inter-harmónicos. Estas correntes distorcidas circulando pelas impedâncias de linha da instalação elétrica causam distorções também nas formas de onda das tensões.

Na Figura C.1a é possível visualizar as formas de onda da tensão e da corrente de um determinado sistema elétrico. A forma de onda da corrente, quando decomposta pela análise de Fourier, ilustrada na Figura C.1b, resulta na soma de sinusoides com frequências múltiplas da frequência fundamental, denominados de harmónico. Além disto, é possível observar componentes não múltiplos da frequência fundamental, também conhecidos como inter-harmónicos, presentes em casos mais graves.

Como discutido anteriormente, os harmónicos presentes na forma de onda da corrente de cargas não-lineares resulta numa distorção na tensão aos terminais da instalação elétrica, alimentando as cargas presentes na instalação com uma tensão distorcida, consumindo assim, também correntes com alto conteúdo harmónico.

As soluções tradicionais para eliminar ou atenuar estes problemas de qualidade de energia elétrica passam pela utilização do seguinte equipamento:



(a) Formas de onda da tensão e da corrente na fonte.

(b) Conteúdo harmónico na forma de onda da corrente.

Figura C.1: Retificador monofásico com filtro capacitivo [1]: (a) formas de onda na fonte e (b) harmónicos da corrente.

- *Uninterruptible Power Supplies* (UPS) para interrupções de longa duração na alimentação;
- *Transient voltage surge suppressors* (TVSS) para variações problemáticas momentâneas na tensão de alimentação;
- Filtros passivos para eliminarem/atenuarem o ruído de alta frequência na instalação elétrica;
- Transformadores de isolamento blindados para proporcionarem isolação galvânica com proteção eletroestácia que evita o aparecimento de transitórias na tensão dos enrolamentos do secundário;
- *Static VAR compensators* para regularem a potência reativa do sistema elétrico.

Os problemas associados com conteúdo harmónico nas correntes que circulam pelo sistema elétrico são uma preocupação crescente para os utilizadores finais, pelo que levou ao desenvolvimento de soluções baseadas em filtros passivos para mitigarem estes problemas. Todavia, o dimensionamento destes componentes pode ser um processo muito complica uma vez que: são dimensionados para filtrarem uma certa gama de frequências; podem causar ressonância com as cargas do sistema elétrico; precisam de ser sobredimensionados devido Às suas limitações face à operação com certo número de cargas, entre outros.

Desta forma, para ultrapassar estes problemas foram desenvolvidas soluções baseadas em filtros ativos de potência como a utilização de: um filtro ativo paralelo que compensa harmónicos na corrente, regula a potência reativa compensando o fator de potência, e equilibra as correntes de um sistema trifásico eliminando a corrente no neutro; um filtro ativo série que garante uma tensão de alimentação na fonte livre de harmónicos e em alguns casos, compensar variações na tensão; e de um *Unified Power Quality Conditioner* (UPQC) combinando ambos os filtros ativos discutidos.