Universidade de Aveiro
2015

Departamento de Eletrónica, Telecomunicações e Informática

Tiago Silva Lopes

Controlador de Tempo Real Baseado em Raspberry Pi

Raspberry Pi based Real Time controller

Universidade de Aveiro
2015

Departamento de Eletrónica, Telecomunicações e Informática

Tiago Silva Lopes

Controlador Tempo-Real Baseado em Raspberry Pi

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Dr. Alexandre Manuel Moutela Nunes da Mota, Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática e do Dr. Paulo Bacelar Reis Pedreiras, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática.

Dedico este trabalho aos meus pais por possibilitarem a minha formação académica.

o júri / the jury

Presidente / President

Prof. Doutor Rui Manuel Escadas Ramos Martins

Professor Auxiliar da Universidade de Aveiro

Vogais / Examiners Committee

Doutor Luís Miguel Pinho de Almeida

Professor Associado da Universidade do Porto – Faculdade de Engenharia

Prof. Doutor Paulo Bacelar Reis Pedreiras

Professor Auxiliar da Universidade de Aveiro (Co-Orientador)

Agradecimentos

Gostaria de começar por agradecer aos meus pais pela possibilidade e incentivo em prosseguir a minha vida académia, pois sem eles não teria chegado a este patamar.

Aos meus orientadores Alexandre Mota e Paulo Pedreiras por me orientarem e por toda a preocupação, tempo e paciência investidos neste trabalho.

Ao Tiago Gonçalves pela ajuda que me deu por diversas vezes.

Aos professores Joaquim Sousa Pinto e Cláudio Teixeira pela oportunidade profissional que me deram e por todo o apoio e incentivo que foram essenciais para o término desta dissertação. Não me posso esquecer aqui também de agradecer ao professor Alexandre Mota por se ter lembrado de mim no aparecimento desta oportunidade.

Aos meus amigos e colegas de trabalho pelo apoio, incentivo e pelos momentos de boa disposição.

A todos os meus amigos, destacando-se o Daniel Valentim, o Frederico Malafaia, e o Diogo Curto por me acompanharem ao longo do curso, pelo apoio e por todas as horas bem passadas no “aquário”.

Para finalizar, queria deixar um agradecimento especial ao Daniel Valentim, por me ter acompanhado ao longo do curso, por todas as viagens que fizemos juntos, por todas as conversas e discussões parvas e absurdas, mas acima de tudo, por todo o apoio que me deu.

A todos um muito obrigado!

Palavras-chave

Controlo digital, Sistemas de tempo-real, *Raspberry Pi*, interfaces AD/DA, Linux

Resumo

O Raspberry Pi é um dispositivo de tamanho reduzido que combina a facilidade no desenvolvimento de aplicações em ambiente Linux com a presença de interfaces digitais, sendo possível aceder e configurar diretamente o *hardware* a partir de aplicações desenvolvidas em linguagem C que executam em *user-space.*

Estas características fazem do Raspberry Pi um potencial dispositivo para utilização em aplicações de controlo de sistemas. Contudo este dispositivo não possui interfaces analógicas, essenciais para interação com sistemas analógicos.

Com o objetivo de tornar o Raspberry Pi mais versátil em aplicações de controlo, foram desenvolvidas no âmbito deste trabalho placas de expansão que adicionam a este dispositivo interfaces analógicas e digitais, que acrescentam também um grau de proteção, protegendo-o de possíveis casos de má utilização.

Keywords Digital Control, Real-time systems, Raspberry Pi, AD/DA interface, Linux

Abstract

Raspberry Pi is a small device that combines the application development facilities of a Linux system, with the presence of digital interfaces accessible from applications developed in C language running in user-space.

These characteristics make Raspberry Pi a potential device for use in control systems applications. However this device does not have analog interfaces, essential for interaction with analog systems.

To enable the use of the Raspberry Pi system on control applications, in the scope of this thesis was developed an extension board with AD and DA capabilities. Additionally, this board also provides digital I/Os, in order to protect the Raspberry Pi hardware.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

AD	Analog to Digital
ADC	Analog to Digital Converter
APB	Advanced Peripheral Bus
API	Application Programming Interface
CAN	Controller Area Network
CI	Circuito Integrado
CPU	Central Processing Unit
CS	Chip Select
CSI	Camera Serial Interface
DA	Digital to Analog
DAC	Digital to Analog Converter
DC	Direct Current
DSI	Display Serial Interface
FIFO	First in First out
GPIO	General Purpose Input/output
HDMI	High-Definition Multimedia Interface
I/O	Input/output
ISR	Interrupt Service Routine
LoSSI	Low Speed Serial Interface
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
PBCLK	Peripheral BUS Clock
PC	Personal Computer
RAM	Random-access Memory
RTOS	Real Time Operating System
SCLK	Serial Clock
SD	Secure Digital
SO	Sistema Operativo
SOC	System on a Chip
SPI	Serial Peripheral Interface
USB	Universal Serial Bus

1 Introdução

1.1 Motivação

O controlo de sistemas está presente em muito daquilo que nos rodeia, desde fornos de cozinha e sistemas de ar-condicionado a robôs industriais e meios de transporte como os aviões. Alguns destes sistemas não exigem muita capacidade de processamento, mas outros como os aviões, impressoras e dispositivos médicos necessitam de um controlo exigente e preciso, pois de outra forma podem falhar e tornar-se ineficientes, podendo mesmo ter consequências graves pondo em risco a integridade física de pessoas e bens.

Os sistemas mais exigentes necessitam de uma grande precisão temporal, o que obriga à utilização de sistemas operativos de tempo real. A utilização deste tipo de sistemas operativos permite garantir uma resposta correta atempadamente, com um tempo de resposta previsível, enquanto nos oferecem as vantagens de utilizar um sistema operativo, como a abstração entre o *hardware* e a aplicação que vai ser executada [1].

Sistemas mais complexos exigem também uma capacidade de processamento que se encontra acima da oferecida pelos microcontroladores, o que leva a que seja necessário passar para o domínio dos PCs (*Personal Computers*). Hoje em dia destaca-se no mercado o *Raspberry Pi* que é um microcomputador que combina as vantagens de utilizar um computador, com o tamanho e a facilidade de acesso ao *hardware* característica dos microcontroladores. Este computador, capaz de executar o sistema operativo Linux, torna-se uma boa opção na altura de escolher um dispositivo para implementar um algoritmo de controlo, pois associado à sua capacidade de processamento e ao seu tamanho, apresenta também um baixo custo e interfaces de comunicação com o exterior.

A aplicação do Raspberry Pi em sistemas de controlo depende ainda da utilização de interfaces analógicas, uma vez que este apenas possui interfaces digitais. Assim no âmbito deste trabalho desenvolveu-se também uma interface I/O analógica tendo por objetivo facilitar a utilização deste dispositivo em aplicações de controlo.

1.2 Objetivos

O Raspberry Pi é um dispositivo que conjuga uma capacidade de processamento bastante acima de um microcontrolador e a facilidade de utilização proporcionada pelo sistema Linux com a proximidade e possibilidade de acesso direto ao *hardware* característica de um microcontrolador, o que o torna um dispositivo susceptivel de ser utilizado em controlo de sistemas. Contudo, o Raspberry Pi não possui interfaces analógicas, o que impede a sua utilização tanto para obter informação de sensores com

saída analógica, como para atuar em sistemas analógicos como é o caso dos motores controlados por tensão. Adicionalmente o número de I/Os digitais é limitado e suscetível a danos em caso de má utilização. Assim, pretende desenvolver-se *hardware* capaz de permitir a interação deste dispositivo com sistemas analógicos e digitais de uma forma simples e robusta. Este *hardware* será composto por um módulo AD (*Analog to Digital*), um módulo DA (*Digital to Analog*) e um módulo de I/Os (Inputs/Outputs) digitais.

Para complementar o *hardware* será também desenvolvida uma biblioteca de *software* que permita a sua utilização, de uma forma simples e rápida, promovendo uma abstração da camada física, facilitando assim a utilização destes módulos sem necessidade de conhecer a sua estrutura. Estas bibliotecas serão desenvolvidas utilizando a linguagem C, tendo em conta que poderão ser utilizadas em sistemas com características de tempo real. A linguagem C é uma linguagem de baixo nível que permite um grande controlo sobre o fluxo de execução do programa, possibilitando um bom controlo temporal.

1.3 Estrutura

Este documento está organizado em oito capítulos:

- Introdução;
- Controlo de tempo-real;
- *Raspberry Pi*;
- *Tempo*-real e tarefas periódicas em Linux;
- Interfaces AD e DA;
- *Software* desenvolvido;
- Testes realizados;
- Conclusões e trabalho futuro.

No primeiro capítulo efetua-se o enquadramento e apresenta-se a motivação, objetivos, estrutura e contribuições desta dissertação.

No segundo capítulo apresenta-se o controlo de tempo real, começando por uma abordagem ao controlo digital e apresentando-se os sistemas de tempo real.

No terceiro capítulo é apresentado o Raspberry Pi, que é o ponto de partida para este trabalho. Inicialmente apresentam-se as suas especificações técnicas e a sua estrutura, seguindo-se uma apresentação mais detalhada do seu processador e periféricos.

O capítulo quatro é dedicado às capacidades do sistema operativo Linux para execução de tarefas periódicas e as suas limitações temporais.

Nos capítulos cinco e seis descreve-se o *hardware* e *software* desenvolvidos, apresentando-se as suas especificações e explicando-se o seu funcionamento. Começando pelo *hardware* são apresentados detalhadamente os circuitos desenvolvidos e as suas especificações. Já no capítulo seis apresentam-se os *device-drivers* GPIO e SPI desenvolvidos e as bibliotecas desenvolvidas para controlo dos módulos AD e DA desenvolvidos.

No capítulo sete são apresentados os testes realizados e são feitas algumas observações aos resultados obtidos.

No oitavo e último capítulo é feita uma apreciação global do trabalho, são apresentadas conclusões e são apresentadas propostas para a continuação deste trabalho.

1.4 Contribuições

As contribuições deste trabalho são:

- O desenvolvimento de placas de extensão para Raspberry Pi com entradas e saídas analógicas, possibilitando a interação do dispositivo com sistemas analógicos.
- Desenvolvimento de *device-drivers* Linux, em linguagem C, para os módulos SPI e GPIO do Raspberry Pi, que permitem a utilização dos seus portos digitais bem como o módulo de comunicação SPI, de uma forma simples e possibilitando uma total abstração do *hardware*.
- Desenvolvimento de APIs para interface às placas de extensão.

O trabalho desenvolvido permite a construção de plataformas de baixo custo para controlo de sistemas e ensino de controlo.

2 Controlo de tempo-real

Neste capítulo são abordados o controlo digital e os sistemas de tempo-real.

2.1 Controlo Digital

A massificação dos microprocessadores levou à diminuição do seu custo, proporcionando uma diminuição do custo dos controladores digitais. Associado a esta diminuição de custo os controladores digitais são menos sensíveis a variações do ambiente e mais flexíveis que os controladores analógicos, uma vez que são reprogramáveis, permitindo assim fáceis modificações ao algoritmo de controlo. Estas razões levaram a que a popularidade deste tipo de controladores subisse, tornando-se assim no principal tipo utilizado nos dias de hoje. [2]

Apesar de os controladores digitais dominarem nos dias de hoje, os sistemas físicos são muitas vezes sistemas analógicos, deparando-nos com sistemas de natureza diferente, que não podem interagir diretamente. Assim é necessário utilizar uma interface que permita esta interação.

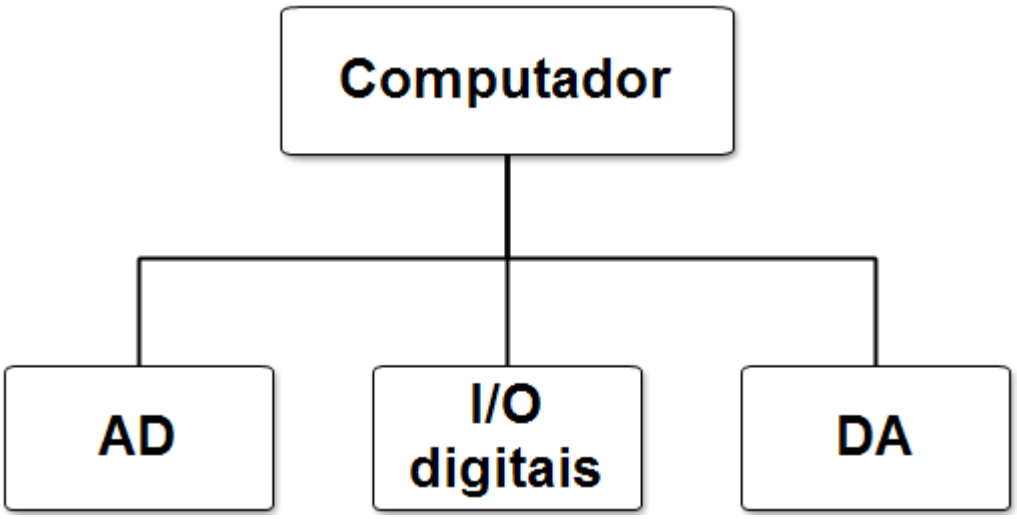

Figura 2.1: Interface AD/DA para computador

A interface que permite a comunicação entre sistemas digitais e sistemas analógicos é uma interface AD/DA constituída por um módulo conversor analógico para digital (AD) e um módulo conversor digital para analógico (DA). O módulo AD permite aquisição do estado do sistema por parte do sistema digital, possibilitando ao controlador determinar o valor de atuação que deve ser aplicado ao sistema. O módulo DA permite a atuação sobre o sistema analógico, possibilitando o controlo do sistema. Esta interface possui ainda um módulo de I/O digitais, que permite a comunicação com sensores e atuadores digitais.

2.1.1 Controlo no domínio discreto

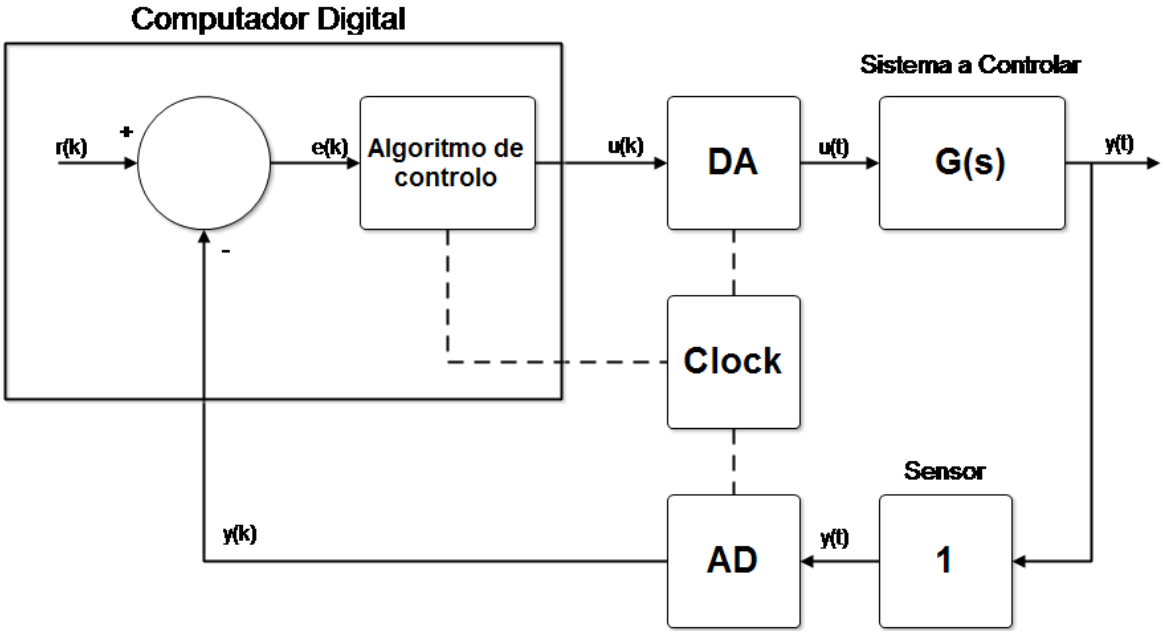

Figura 2.2: Sistema de controlo discreto [2]

A Figura 2.2 apresenta o diagrama de blocos de um sistema de controlo discreto, onde se podem identificar os blocos correspondentes ao computador digital, que executa um algoritmo de controlo, os conversores AD e DA, o sistema a controlar G(s), e um sensor. É ainda apresentado um bloco designado por Clock, responsável pela sincronização entre o algoritmo de controlo e os conversores AD e DA. Esta função de sincronização está tipicamente implícita na periodicidade do algoritmo de controlo e na comunicação entre o computador e os conversores AD e DA, sendo a função do bloco de relógio determinar a periodicidade de execução do algoritmo de controlo. As interfaces AD e DA podem ter interferência no período do algoritmo de controlo, pois adicionam um atraso ao sistema que pode ser variável e não controlável. Na figura estão ainda identificados os sinais de referência r(k), de erro e(k), de controlo u(k) e u(t) e de saída y(t) e y(k). [2]

Com uma periodicidade h (período de amostragem), a tarefa do controlador é executada começando por aceder ao conversor AD para amostrar o valor do sinal de saída, o algoritmo de controlo é executado utilizando o sinal convertido y(k) e o sinal de referência r(k) para gerar o sinal de controlo. Por fim o computador acede ao conversor DA para converter o sinal u(k) no sinal u(t) que é aplicado na entrada do sistema.

Um *software* para controlo de sistemas possui assim três passos principais:

- Ler o conversor AD
- Executar o algoritmo de controlo para calcular o sinal de controlo
- Enviar o sinal de controlo para o conversor DA

Note-se que a tarefa de controlo é executada com as interrupções desativadas. Isto é necessário para garantir que a tarefa não é interrompida por outras interrupções.

Na implementação de sistemas de controlo discreto, um novo valor para o sinal de controlo é calculado em cada iteração do algoritmo de controlo, ou seja com uma periodicidade igual ao período de amostragem. Assim o sinal de controlo permanece constante durante o intervalo de tempo h, estando durante este tempo o sistema a funcionar em malha aberta. [2]

2.1.2 Abordagem de controlo digital

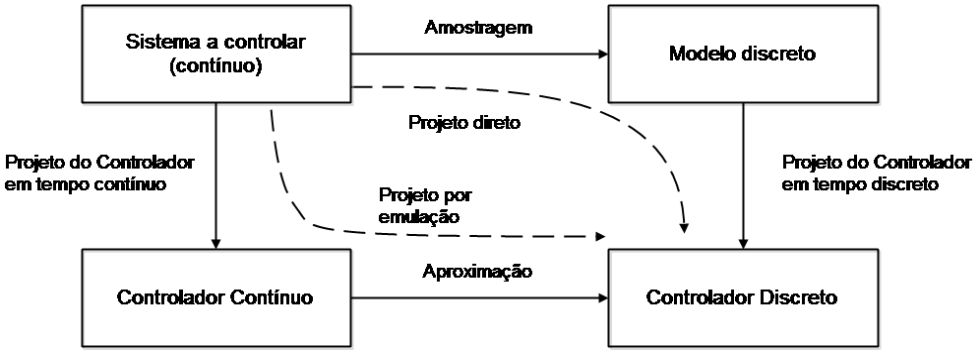

Figura 2.3: Abordagens de controlo Digital [2]

Na implementação de um sistema de controlo digital podem seguir-se duas abordagens:

- Controlo Digital Direto
- Controlo por emulação

O controlo digital direto consiste em projetar o controlador totalmente no domínio discreto, conhecendo o modelo discreto do sistema a controlar. Para isso é necessário determinar o modelo discreto do sistema, recorrendo a técnicas de identificação de sistemas.

No controlo por emulação é projetado um controlador no domínio contínuo cujas equações diferenciais são aproximadas por equações diferença, obtendo-se as equações no domínio discreto. [2]

2.1.3 Período de amostragem

Para abordagens por controlo digital direto, é possível obter um desempenho aceitável através da abordagem sugerida por Åström Wittenmark que consiste em dividir o tempo de subida do sistema a controlar (T_r) por um número de amostras (N_r) entre 4 e 10. Assim

$$h = \frac{T_r}{N_r} \qquad 4 \leq N_r \leq 10$$

Para abordagens por controlo digital por emulação, pode ser utilizada a abordagem sugerida por Franklin, Powell e Workman que sugerem tomar como frequência de amostragem (f_s) um valor pelo menos trinta vezes superior ao valor pretendido para a largura de banda (f_{cl}) do sistema em malha fechada. Assim [2]

$$f_s \geq 30 f_{cl}$$

2.2 Sistemas de tempo-real

Um sistema de tempo-real é um sistema cujo bom comportamento não é definido apenas pela correção lógica da sua resposta, mas também pelo tempo em que esta é gerada. Nestes sistemas existe um limite temporal para a obtenção de uma resposta em relação ao momento do estímulo. Este limite temporal pode ser:

- **Deadline** – é imposto um tempo máximo para a terminação de uma tarefa. O instante em que esse tempo termina denomina-se *deadline.*
- **Janela** – é imposto um limite máximo e um limite mínimo ao instante de terminação da tarefa.
- **Sincronismo** – é imposto um tempo máximo para a diferença entre a geração de dois eventos de saída.
- **Distância** – é imposto um tempo máximo à diferença entre a terminação ou ativação de duas instâncias consecutivas.

A utilidade da resposta gerada pelo sistema a um estímulo é o fator caracterizador da *deadline*. Assim a *deadline* pode ter as seguintes classificações:

- *Soft*
- *Firm*
- *Hard*

Uma *deadline* é caracterizada como *soft* se uma reação tardia ainda mantem utilidade, apesar de causar degradação de desempenho. Se após a *deadline* a resposta do sistema deixa de ter qualquer utilidade, a *deadline* é classificada como *Firme.* A caracterização de *hard* é atribuída às *deadlines* que ao serem ultrapassadas podem levar à falha do sistema, e a possíveis perdas humanas e/ou económicas catastróficas. [3] [4] [5]

2.2.1 Classificação dos sistemas de tempo-real

Os sistemas de tempo-real podem ser classificados de vários pontos de vista e assumir classificações dependentes das características do sistema a controlar, ou dependentes da implementação do sistema computacional. [3]

Dependendo das características do sistema que se pretende controlar os sistemas de tempo-real podem ser:

- *Soft real-time* ou *hard real-time*
- *Fail-safe* ou *fail-operational*

Dependendo do sistema computacional os sistemas de tempo-real podem ser:

- *Guaranteed-response* ou *best-effort*
- *Resource-adequate* ou *resource-inadequate*
- *Event-triggered* ou *time-triggered*

2.2.1.1 *Soft Real-Time* e *Hard Real-Time*

Os sistemas de tempo-real são caracterizados quanto às *deadlines* das suas tarefas como *soft* ou *hard*. Sistemas que necessitam de cumprir pelo menos uma *deadline* do tipo *hard* denominam-se *hard real-time systems,* enquanto sistemas que apenas possuem *deadlines* do tipo *soft* e *firm* denominam-se *soft real-time systems.* [3] [4]

De entre os sistemas do tipo *hard* existem os sistemas nucleares e os sistemas de aviação, em que uma falha pode levar a uma catástrofe nuclear ou à queda de um avião. Já nos sistemas do tipo *soft* existem por exemplo os reprodutores de som, que em caso de falha o resultado pode ser completamente inútil, mas não levam a nenhuma catástrofe.

2.2.1.2 *Fail-Safe* e *Fail-Operational*

Em alguns sistemas de tempo-real do tipo *hard* existem estados de segurança que podem ser atingidos em caso de falha do sistema. Estes sistemas denominam-se *fail-safe.* A existência destes estados de segurança é uma característica intrínseca ao sistema a controlar. Os sistemas *fail-safe* devem ter uma alta capacidade de deteção de erros e devem ser capazes de em caso de falha, convergir rapidamente para um estado de segurança, sendo que durante o tempo necessário a esta transição estes sistemas operam como um sistema *fail-operacional*. Nos sistemas do tipo *hard* que não possuem estados de segurança, em caso de falha do sistema, este deve permanecer em funcionamento tentando evitar uma catástrofe. Estes sistemas denominam-se *fail-operational.* [3]

Um exemplo de um sistema *fail-safe* são os comboios, em que no caso de ocorrer uma falha do sistema, os comboios podem parar não causando nenhuma catástrofe. Já um

sistema *fail-operational* são os aviões, em que o sistema tem de continuar a funcionar da forma mais segura possível, permitindo ao avião continuar a voar, mesmo em caso de falha.

2.2.1.3 *Guaranteed-Response* e *Best-Effort*

Os sistemas *guaranteed-response* são aqueles em que conhecendo inicialmente todas os cenários de falha e de carga do sistema, é possível determinar o comportamento exato do sistema. Os sistemas *best-effort* são sistemas nos quais não é necessária uma rigorosa análise dos cenários de falha e de carga do sistema, sendo o desempenho do sistema determinado na fase de testes. É difícil garantir o bom funcionamento dos sistemas *best-effort* para eventos esporádicos. [3] [4]

2.2.1.4 *Resource-Adequate* e *Resource-Inadequate*

Em sistemas *guaranteed-response* é necessário garantir que nas situações de pico de carga, e falha especificadas durante a fase de desenvolvimento, vão existir recursos suficientes que permitam manter o correto funcionamento do sistema. Assim designam-se estes sistemas como *resource-adequate*. Para os sistemas *best-effort* é utilizado um modelo probabilístico para determinar a quantidade de recursos, economicamente viável, necessária recorrendo-se a mecanismos de partilha de recursos. Estes sistemas são caracterizados como *resource-inadequate*. [3]

2.2.1.5 *Event-Triggered* e *Time-Triggered*

Event-triggered e *time-triggered* são as caracterizações que um sistema pode tomar, de acordo com o mecanismo utilizado para desencadear as ações do sistema computacional, como a execução de tarefas. [3] [4]

Um sistema diz-se *event-triggered* se as suas ações são despoletadas por eventos, sinalizados ao CPU pelo sistema de interrupções. Estes sistemas são tipicamente utilizados na monitorização de condições aperiódicas no estado do sistema, e caracterizam-se por:

- Taxa de utilização do sistema computacional variável
- Utilização de mecanismos de escalonamento dinâmicos
- Situação de pior caso mal definida.

Nos sistemas *time-triggered* o disparo das tarefas é realizado pelo relógio do sistema de forma periódica. Estes sistemas são tipicamente utilizados em aplicações de controlo e caracterizam-se por:

- Taxa de utilização de CPU constante
- Situação de pior caso bem definida

2.2.2 Sistema computacional

Um sistema de tempo-real segue o modelo computacional de tempo-real. Este modelo consiste num programa que pode executar indefinidamente sobre um fluxo de dados com o qual tem de se manter sincronizado, impondo restrições temporais à execução do programa. [4]

Num modelo de tempo real as tarefas podem ter requisitos de três tipos: [5]

- **Temporais** – limites temporais aos instantes de terminação ou geração de eventos de saída.
- **Precedência** – estabelecem uma determinada ordem de execução entre tarefas.
- **Exclusão mutua** – acesso a recursos partilhados.

Um dos requisitos ao controlo temporal é a periodicidade das tarefas, quanto à qual as tarefas podem ser caracterizadas como: [5] [4]

- **Periódicas** – tarefas instanciadas infinitamente com um intervalo de tempo fixo entre ativações consecutivas.
- **Aperiódicas** – tarefas instanciadas infinitamente sem regularidade no intervalo entre ativações consecutivas.
- **Esporádicas** – tarefas aperiódicas em que existe um intervalo mínimo entre ativações consecutivas definido.

2.2.3 Aplicações de tempo-real

O método utilizado no desenvolvimento de aplicações de tempo-real está relacionado com a quantidade e o tipo de tarefas que é necessário executar.

2.2.3.1 Programação direta sobre o CPU

Em casos em que a aplicação de tempo-real envolve apenas um ciclo principal e algumas tarefas assíncronas, a programação é normalmente efetuada diretamente sobre o CPU, sem recurso a Sistemas Operativos ou Executivos. Nestes casos as tarefas assíncronas podem ser encapsuladas em rotinas de interrupção.

Em programação direta sobre o CPU, a execução de tarefas recorrendo interrupções pode ser feita utilizando interrupções periódicas ou interrupções assíncronas. As interrupções periódicas são despoletadas por *timers* e utilizadas para executar tarefas

periódicas. Para executar tarefas disparadas por eventos utilizam-se interrupções assíncronas. [4]

A utilização de interrupções impõe algumas preocupações adicionais, como a necessidade de salvaguardar o contexto do CPU e o facto de o programa principal perder tempo de execução por ser interrompido para execução das rotinas de serviço a interrupção (ISR), podendo no limite ficar bloqueado.

As interrupções podem ainda ser utilizadas com encadeamento. Assim é permitido que a execução de uma ISR seja interrompida por uma interrupção de prioridade mais elevada. Neste caso é melhorado o tempo de resposta das tarefas mais críticas à custa de um aumento da complexidade do sistema. [4]

2.2.3.2 Utilização de Sistemas Operativos ou Executivos

Quando uma aplicação de tempo-real envolve múltiplas tarefas, utilizam-se Sistemas Operativos ou Executivos multitarefa, que facilitam bastante a programação da aplicação por já oferecerem suporte a múltiplas tarefas. Nestes o disparo das tarefas é feito por uma interrupção periódica (*tick*) que é a unidade de tempo do SO.

A utilização de um SO ou um Executivo tem como vantagens: [4]

- Maior nível de abstração
- Menor dependência relativamente ao *hardware*
- Maior facilidade de manutenção do *software*

Um Executivo possui os seguintes serviços básicos: [4]

- Gestão de tarefas
- Gestão de tempo
- Escalonamento de tarefas
- Despacho de tarefas
- Gestão de recursos partilhados

Contudo a utilização de um SO ou Executivo leva também ao aumento do número de instruções a executar, uma vez que é necessário executar as instruções correspondentes a esta camada intermédia de *software*.

Alguns exemplos de Sistemas Operativos de Tempo-Real (RTOS) e Executivos são:

- **LynxOS** – Sistema operativo desenvolvido para aplicações *hard real-time* que possui suporte para processadores multicore. O LynxOs dispões também de uma API POSIX nativa que proporciona uma fácil portabilidade das aplicações desenvolvidas em Linux, e permite que as aplicações para LynxOS sejam testadas em Linux.

- **VxWorks** – Sistema modular para tarefas *hard real-time* que permite adicionar ou remover pacotes sem intervir no núcleo do SO. Este SO disponibiliza suporte a alguns protocolos de comunicação como USB, CAN e Bluetooth.

- **FreeRTOS** – É um executivo de tempo gratuito, projetado para ser simples, fácil de utilizar e pequeno o suficiente para ser executado em microcontroladores. Tipicamente ocupa entre 4KB a 9KB.

- **Xenomai** – É um *software* gratuito, desenvolvido para combater a falta de garantias temporais do Linux. Este *software* consiste num *kernel* de tempo-real que executa paralelamente ao *kernel* do Linux e implementa *skins* para diversos RTOS, o que facilita o porte de aplicações de outros RTOS para Xenomai.

2.2.4 Algoritmos de Escalonamento de tarefas

Os Executivos são responsáveis pela gestão da atribuição do processador às tarefas. Para isso utilizam algoritmos de escalonamento que implementam uma política de escalonamento, determinando qual a ordem de execução das tarefas.

Os algoritmos de escalonamento podem ser classificados como: [5]

- Preemptivo ou não-preemptivo
- Estático ou dinâmico
- *Off-line* ou *On-line*
- Ótimo ou heurístico

2.2.4.1 Algoritmo preemptivo e não-preemptivo

Num algoritmo preemptivo a tarefa em execução pode ser interrompida para que outra tarefa seja executada.

Num algoritmo não-preemptivo a tarefa em execução não pode ser interrompida, executando até que termine.

2.2.4.2 Algoritmo estático e dinâmico

Nos algoritmos estáticos a ordem de execução é baseada em parâmetros definidos aquando da ativação de cada tarefa. Nestes algoritmos as prioridades das tarefas são fixas, sendo definidas aquando da sua ativação e permanecendo inalteradas a partir daí. Um exemplo de um algoritmo de escalonamento estático é o escalonamento estático cíclico.

Em algoritmos dinâmicos, os parâmetros, que definem a ordem de execução das tarefas, podem variar durante a evolução do sistema, levando a que as prioridades relativas variem (prioridades dinâmicas).

2.2.4.3 Escalonamento *on-line* e *off-line*

O escalonamento *off-line* é aquele em que a ordem de execução das tarefas é definida tendo em conta todas as tarefas, e guardado numa tabela, não sofrendo alterações durante a execução da aplicação.

O escalonamento *on-line* é aquele em que as decisões de escalonamento são tomadas sempre que uma tarefa é ativada ou termina a sua execução.

2.2.4.4 Algoritmo ótimo e heurístico

Um algoritmo de escalonamento é ótimo quando minimiza uma função de custo. Do ponto de vista de escalonamento, um algoritmo diz-se ótimo, dentro de uma dada classe, se consegue encontrar um escalonamento praticável sempre que qualquer outro algoritmo da mesma classe é também capaz de encontrar um escalonamento praticável.

Um algoritmo de escalonamento é heurístico quando pode determinar um escalonamento ótimo, mas não existe garantias de o conseguir.

3 Raspberry Pi

O Raspberry Pi é um computador de baixo custo, do tamanho de um cartão de crédito. Este pode ser ligado a um monitor ou televisão através da saída HDMI ou da saída de vídeo analógica. Possui ainda uma saída de áudio e na sua versão mais básica uma ficha USB, que permite ligar um simples teclado e rato USB. [6] Este dispositivo é dotado de um SOC (*System on a chip*) BCM2835 com 256MB de memória RAM na sua versão mais básica, utiliza como unidade de armazenamento um cartão SD e disponibiliza também uma ficha de 26 pinos GPIO e conectores *Display Serial Interface* (DSI) e *Camera Serial* Interface (CSI). O Raspberry Pi é alimentado por uma fonte DC de 5V através de uma ficha micro USB.

O Raspberry Pi é compatível com sistemas operativos baseados em GNU/Linux.

A versão utilizada neste trabalho é o modelo B com o sistema operativo Raspbian.

3.1 Raspberry Pi modelo B

O modelo B do Raspberry Pi conta com 2 portas USB, uma porta *Ethernet*, e 512 MB de memória RAM. A presença de uma porta *Ethernet* permite a ligação à rede, possibilitando operar o Raspberry Pi remotamente.

A Figura 3.2 apresenta o principal conector GPIO (General Purpose Input/Output) disponibilizado pelo Raspberry Pi.

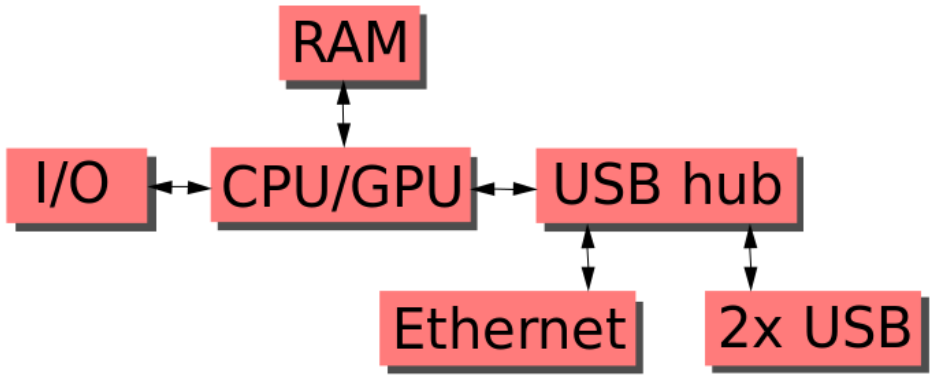

Figura 3.1: Arquitetura do Raspberry Pi B

Figura 3.2: Conector de 26 pinos [7]

3.2 Periféricos do SOC BCM2835

O SOC BCM2835 contém vários periféricos. Alguns destes periféricos são controlados pelo GPU pelo que não é recomendada a sua utilização. Existem no entanto periféricos que podem ser utilizados de uma forma segura. Esses periféricos são os seguintes:

- Timers
- Controlador de Interrupções
- GPIO
- USB
- PCM/I2S
- Controlador de DMA
- I2C
- SPI
- PWM
- UART

Estes periféricos estão mapeados em memória podendo ser acedidos na gama de endereços físicos entre 0x20000000 e 0x20FFFFFF.

3.3 Acesso aos periféricos

A nível físico os periféricos do BCM2835 estão na gama 0x20000000 a 0x20FFFFFF, mas a nível de barramento de endereços, estes encontram-se mapeados em 0x7Exxxxxx. Assim, os periféricos que se encontram associados a um endereço 0x7Ennnnnn aparecem fisicamente em 0x20nnnnnn. [8]

3.3.1 Mapeamento de dispositivos em Linux

O SO utilizado é o Raspbian, que é baseado em Linux. Nesta secção vai ser apresentado o processo de mapeamento de dispositivos para o SO Linux.

Para mapear uma região da memória física no espaço de endereçamento do processo, é necessário primeiro garantir acesso à memória. Em Linux a memória física do sistema está mapeada em /dev/mem. /dev/mem é uma imagem da memória principal do computador, sendo assim qualquer operação sobre este ficheiro traduzida numa operação sobre a memória física do sistema. [9]

Para garantir acesso à memória do sistema, utiliza-se a *syscall* Open disponibilizada pelo SO Linux. Esta *syscall* utilizada sobre /dev/mem devolve um descritor para o ficheiro que pode ser utilizado para subsequentes acessos. [10]

Obtido o acesso à memória física do computador, utiliza-se a função mmap, que é específica do Linux, para mapear a região de memória correspondente ao dispositivo desejado no espaço de endereçamento do processo.

A função mmap serve para mapear ficheiros e dispositivos em memória, devolvendo um ponteiro para a região de memória desejado. Neste caso como se pretende mapear regiões da memória física, utilizando /dev/mem, devem ser utilizados endereços físicos.

Uma vez mapeado o dispositivo no espaço de endereçamento do processo, este passa a ser diretamente endereçável.

3.4 GPIO

O Raspberry Pi possui 54 portos I/O, estando 17 destes portos presentes no conector P1, apresentado na Figura 3.2. Cada GPIO tem pelo menos duas funções alternativas, que são normalmente I/O associados a periféricos. O GPIO possui 41 registos, de 32 bits cada,

tendo um espaço de endereçamento total de 164 bytes iniciando-se no endereço físico 0x20200000.

A função dos portos de I/O pode ser definida utilizando os 6 registos GPFSELx em que a função de cada porto é definida pela combinação de 3 bits, disponibilizando as opções de entrada, saída e 6 funções alternativas.

Bit(s)	Field Name	Description	Type	Reset
31-30	---	Reserved	R	0
29-27	FSEL9	FSEL9 - Function Select 9 000 = GPIO Pin 9 is an input 001 = GPIO Pin 9 is an output 100 = GPIO Pin 9 takes alternate function 0 101 = GPIO Pin 9 takes alternate function 1 110 = GPIO Pin 9 takes alternate function 2 111 = GPIO Pin 9 takes alternate function 3 011 = GPIO Pin 9 takes alternate function 4 010 = GPIO Pin 9 takes alternate function 5	R/W	0
26-24	FSEL8	FSEL8 - Function Select 8	R/W	0
23-21	FSEL7	FSEL7 - Function Select 7	R/W	0
20-18	FSEL6	FSEL6 - Function Select 6	R/W	0
17-15	FSEL5	FSEL5 - Function Select 5	R/W	0
14-12	FSEL4	FSEL4 - Function Select 4	R/W	0
11-9	FSEL3	FSEL3 - Function Select 3	R/W	0
8-6	FSEL2	FSEL2 - Function Select 2	R/W	0
5-3	FSEL1	FSEL1 - Function Select 1	R/W	0
2-0	FSEL0	FSEL0 - Function Select 0	R/W	0

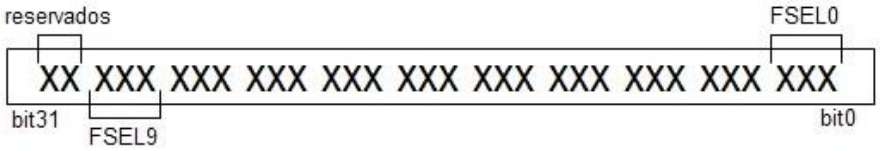

Figura 3.3: Registo GPFSEL0 [11]

Na Figura 3.3 está representada a organização do registo GPFSEL0, um dos registos que permite configurar a função de cada GPIO. Neste registo é possível configurar as funções dos GPIOs 0 a 9. Por exemplo, para definir o GPIO9 como saída, é necessário colocar o campo FSEL9 deste registo com o valor 0x1.

Os portos definidos com função alternativa são controlados pelo respetivo periférico. Pela Figura 3.4 pode-se observar que o GPIO9 pode ter a função de *Master Input Slave Output* (MISO) associado ao periférico SPI0. Para isso é necessário defini-lo com a função alternativa 0, ou seja definir o campo FSEL9 do registo GPIOSEL0 com o valor 0x4.

	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO0	High	SDA0	SA5	<reserved>			
GPIO1	High	SCL0	SA4	<reserved>			
GPIO2	High	SDA1	SA3	<reserved>			
GPIO3	High	SCL1	SA2	<reserved>			
GPIO4	High	GPCLK0	SA1	<reserved>			ARM_TDI
GPIO5	High	GPCLK1	SA0	<reserved>			ARM_TDO
GPIO6	High	GPCLK2	SOE_N / SE	<reserved>			ARM_RTCK
GPIO7	High	SPI0_CE1_N	SWE_N / SRW_N	<reserved>			
GPIO8	High	SPI0_CE0_N	SD0	<reserved>			
GPIO9	Low	SPI0_MISO	SD1	<reserved>			
GPIO10	Low	SPI0_MOSI	SD2	<reserved>			
GPIO11	Low	SPI0_SCLK	SD3	<reserved>			

Figura 3.4: Excerto da tabela de funções alternativas [11]

O nível lógico dos portos definidos como entrada é lido através dos registos GPLEV0 e GPLEV1. Cada bit destes registos corresponde a um GPIO, sendo que para o registo GPLEV0, o bit n corresponde ao GPIOn, e para o registo GPLEV1 o bit n corresponde ao GPIO[n+32].

Para os portos de saída é possível definir o seu nível lógico utilizando os registos GPSETx e GPCLRx para colocar a ´1´e a ´0´ respetivamente. Tal como acontece para os registos GPLEVx, o bit n dos registos GPSET0 e GPCLR0 corresponde ao GPIOn, e o bit n dos registos GPSET1 e GPCLR1 corresponde ao GPIO[n+32].

O facto de serem utilizados dois registos diferentes para definir os diferentes níveis lógicos, permite que a tensão de saída dos portos seja alterada dispensando uma operação de *read-modify-write*, limitando-se a uma escrita no registo GPSETx para colocar o nível lógico '1' e uma escrita no registo GPCLRx para colocar o nível lógico '0'.

Os portos presentes no conector P1 apresentado na Figura 3.2 encontram-se todos mapeados nos registos GPLEV0, GPSET0 e GPCLR0.

3.5 SPI

O periférico SPI do Raspberry Pi pode funcionar em modo SPI comum com comunicação implementada em 3 fios, em modo SPI de 2 fios, sendo o canal de dados bidirecional, ou em modo LoSSI (Low Speed Serial Interface). Este módulo possui ainda uma FIFO de transmissão e uma de receção. O módulo encontra-se representado na Figura 3.5. [12]

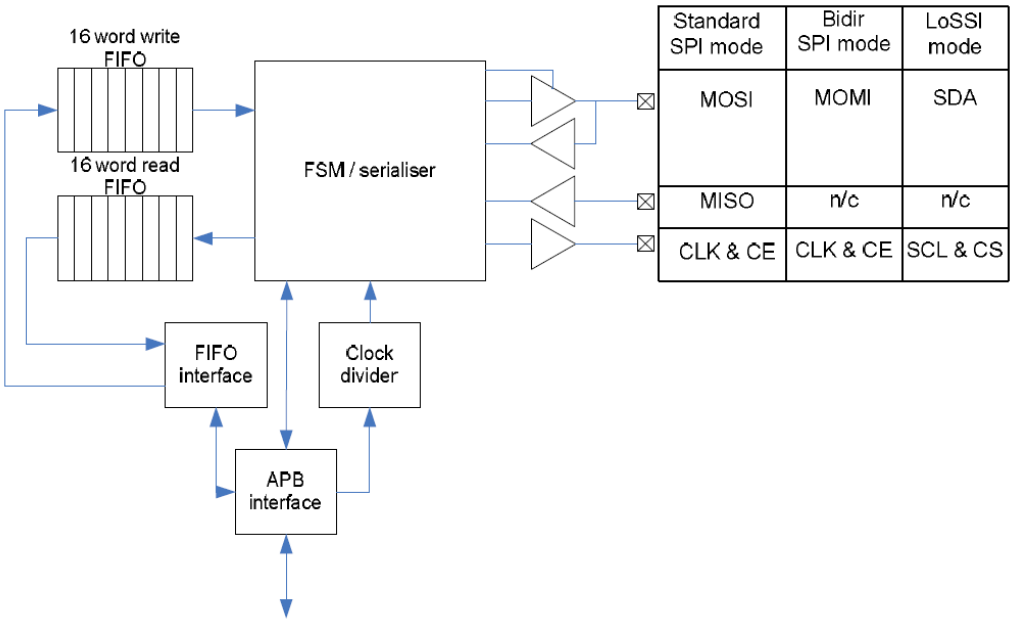

Figura 3.5: Módulo SPI [12]

Este periférico tem associados três *chip select*(CS), um SCLK, um MOSI e um MISO. Dois CS, o SCLK, o MOSI e o MISO estão disponíveis no conector apresentado na Figura 3.2, através da utilização dos respetivos GPIOs com função alternativa, não sendo possível através deste conector utilizar o CS2.

O periférico SPI possui 6 registos, de 32 bits, tendo um espaço de endereçamento total de 24 bytes iniciando-se no endereço físico 0x20204000. [12]

Os registos do SPI são: [12]

- **CS** – contém os principais bits de configuração e de estado do periférico. Neste registo é possível configurar e controlar o funcionamento do periférico SPI, assim como conhecer o seu estado.

- **FIFO** – registo de acesso às FIFOs de receção e transmissão. Operações de escrita neste registo traduzem-se em escritas na FIFO de transmissão, enquanto operações de leitura deste registo são traduzidas em leituras da FIFO de receção.

- **CLK** – este registo permite configurar a frequência de relógio do SPI. Apesar de ser um registo de 32 bits, os 16 bits mais significativos não são utilizados, sendo o valor de configuração CDIV apenas representado pelos 16 bits menos significativos. O valor CDIV é o valor pelo qual deve ser dividida a frequência de relógio do sistema para se obter a frequência desejada.

- **DLEN** – este registo só é utilizado quando o periférico SPI funciona em modo DMA, e indica o número de bytes a transferir.

- **LTOH** – este registo é utilizado quando o SPI funciona em modo LoSSI. Apenas os três bits menos significativos deste registo são utilizados, e representam o tempo que a saída é mantida em número de ciclos de relógio APB.

- **DC** – este registo é utilizado em modo DMA, e permite configurar os níveis das FIFOs para os quais os sinais DREQ e Panic são gerados.

O registo CS tem bits de configuração, controlo e estado. Através deste registo é possível configurar:

- A fase e polaridade o sinal de relógio
- A polaridade de cada linha de CS
- As interrupções a serem geradas
- O funcionamento da linha de CS em transferências por DMA
- O modo de funcionamento do periférico (SPI ou LoSSI)

As opções de controlo disponibilizadas por este registo são:

- Linha de CS a utilizar
- Limpar as FIFOs
- Iniciar transferência

As indicações de estado disponíveis são:

- Transferência completa
- Transferência ativa
- Estado das FIFOs

Estes registos encontram-se representados em 10.3.

4 Tempo-real e tarefas periódicas em Linux

Ao contrário dos SO de tempo-real que são desenhados em função do determinismo temporal, o Linux é um sistema *time-sharing*, otimizado para maximizar a utilização dos recursos de *hardware* (CPU, memória, I/O). Assim, o SO Linux não apresenta nativamente garantias de tempo-real. [13]

Nos sistemas de tempo-real é necessário cumprir *deadlines*, sendo fundamental conhecer o tempo decorrido entre o desencadeamento de um evento e o seu atendimento (latência). Tipicamente nestes sistemas os eventos são a atendidos através de um sistema de interrupções. A rotina de serviço à interrupção (ISR) muda o estado da tarefa que deve atender o evento para *ready* ficando esta pronta a ser colocada em execução pelo escalonador. Nestes casos a latência corresponde ao tempo decorrido entre o surgimento do evento e o instante em que a tarefa, que o deve atender, entra em execução. Em Linux não é possível determinar qual a latência máxima para o atendimento de um evento. [13]

Para melhorar o desempenho temporal do Linux, têm vindo a ser introduzidos melhoramentos ao *kernel*, por forma a limitar a sua latência, e disponibilização de alguns serviços específicos de tempo-real como *timers* de alta resolução, possibilidade de bloqueio de memória e mutexes.

4.1 Aplicações de tempo-real em Linux

O *kernel* Linux suporta várias classes de escalonamento. Por defeito um processo executa com classe de escalonamento do tipo time-sharing, no entanto existem duas classes tempo-real. Os processos a executar com classe tempo-real têm associada uma prioridade estática entre 1 e 99, sendo 1 a prioridade mais baixa e 99 a prioridade mais elevada. Sempre que um processo com classe de tempo-real está pronto a ser executado, qualquer processo de classe não tempo-real é interrompido para que este possa ser executado. [14] [4]

As duas classes de tempo-real são: [14]

- **SCHED_FIFO** – Os processos nesta classe de prioridades respeitam uma política do tipo FIFO, ou seja, quando entram num estado em que estão prontas a ser executadas, entram para o fim da fila da sua prioridade. Se o processo for interrompido por um processo de prioridade mais elevada, permanece no início da fila para que volte a entrar em execução.
- **SCHED_RR** – Esta classe de prioridades respeita os mesmos princípios que a classe SCHED_FIFO, com a diferença de que nesta classe existe um limite para

o tempo de execução de cada tarefa. Se este tempo esgotar, a tarefa entra para o fim da fila da sua prioridade.

A partir da versão 3.14, o SO Linux possibilita também a utilização da classe de escalonamento **SHED_DEADLINE**, que implementa o algoritmo de escalonamento GEDF (Global Earliest Deadline First).

A atribuição de uma classe de prioridades a um processo é feita utilizando a função sched_setscheduler() como apresentado no exemplo seguinte:

```
pid_t pid;                          // ID do processo

struct sched_param proc_sched;      // estrutura que guarda a prioridade do
                                    // processo

pid = getpid();                     // obter o ID do processo

proc_sched.sched_priority=99;       // definir a prioridade desejada (99)

sched_setscheduler(pid, SCHED_FIFO, &proc_sched);   // atribuir a classe de
                                                    // prioridades
```

Figura 4.1: Atribuição da classe de prioridades a um processo.

Para execução de tarefas periódicas em Linux, podem ser utilizados *timers* periódicos disponibilizados pelo sistema. Estes *timers* são configurados para expirarem com um intervalo igual ao período que se pretende para a tarefa. Existem três opções para a ação a tomar quando o *timer* expira: [15] [16]

- Não fazer nada
- Notificar o processo através do envio de um sinal
- Executar uma função numa nova *thread*

Sinais são mecanismos de notificação assíncrona que podem ser utilizados para sinalizar a ocorrência de eventos. Em Linux existem dois tipos de sinais, os sinais comuns, que têm significados específicos e ações predefinidas, e os sinais de tempo-real. Estes últimos são 32 e não possuem nenhum significado específico, sendo todos eles para utilização por parte da aplicação. A função dos sinais pode ser definida utilizando a *syscall* sigaction(). A ação predefinida para os sinais de tempo-real é a terminação do processo que os recebe. [4] [17]

Existem três opções para a ação associada a um sinal: [18]

- Ação predefinida
- Ignorar o sinal
- Executar uma função

Ao longo deste trabalho, as tarefas periódicas consistem em funções que são executadas como resposta ao sinal enviado pelo *timer*. Abaixo é apresentado um exemplo das configurações utilizadas.

```
struct sigaction timer_act;              // estrutura que define a ação associada ao
                                         // sinal

struct sigevent timer_event;             // estrutura que define a ação a tomar
                                         // quando o timer expira

sigemptyset(&timer_act.sa_mask);

timer_act.sa_handler = task;             // task é o nome da função correspondente
                                         // à tarefa periódica a ser executada

timer_act.sa_flags = 0;

sigaction(TIMER_SIG, &timer_act, NULL);     // atribuir a ação desejada ao sinal
                                         // (SIGRTMIN <= TIMER_SIG <= SIGRTMAX)

timer_event.sigev_notify = SIGEV_SIGNAL;

timer_event.sigev_signo = TIMER_SIG;

timer_event.sigev_value.sival_ptr = &timer_id;

timer_create(CLOCK_REALTIME, &timer_event, &timer_id);        // criar o timer
```

Figura 4.2: Configuração do timer para geração de uma tarefa periódica

5 Interfaces AD e DA

As interfaces AD e DA foram desenvolvidas em 3 módulos. Dois destes módulos correspondem às interfaces AD e DA e o terceiro módulo é responsável pela interação entre os dois primeiros módulos e o Raspberry Pi. Este sistema encontra-se representado no diagrama da Figura 5.1.

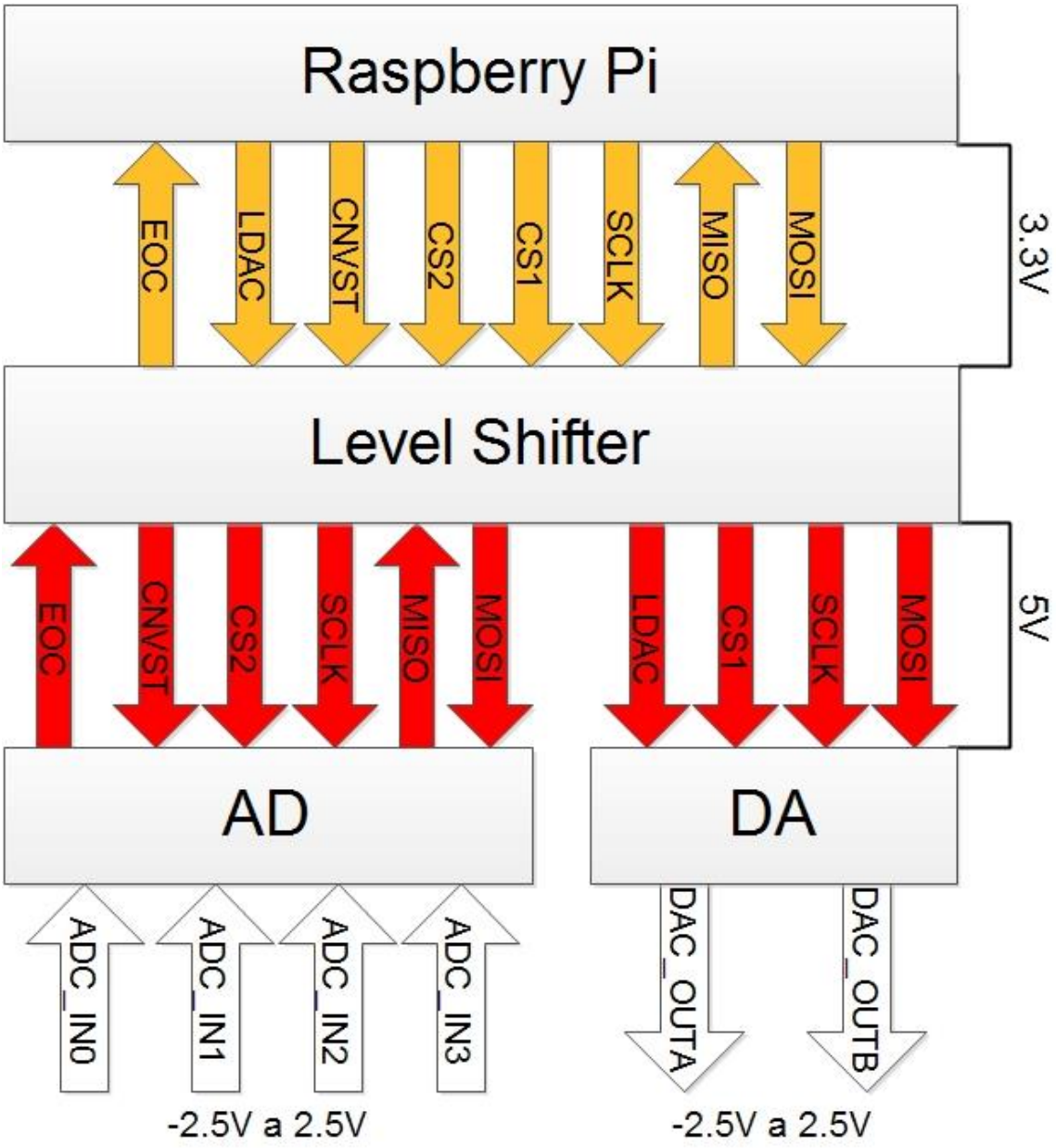

Figura 5.1: Interação entre módulos do Sistema desenvolvido

A comunicação digital entre o Raspberry Pi e os módulos AD e DA utiliza o protocolo SPI e alguns sinais de controlo. Estas duas interfaces digitais não podem ser ligadas diretamente pois enquanto o Raspberry Pi utiliza níveis lógicos de 3.3V os módulos AD e

DA funcionam com níveis lógicos de 5V. Assim desenvolveu-se o terceiro módulo responsável pela conversão de níveis lógicos possibilitando a comunicação entre os restantes dispositivos.

As especificações dos módulos desenvolvidos são:

- 4 canais de entrada analógica de gama [-2.5; 2.5]V
- 2 canais de saída analógica de gama [-2.5; 2.5]V
- AD e DA de resolução de 12 bits.
- Saídas de alimentação para todas as tensões disponíveis em cada módulo.

5.1 Interface AD

A interface AD consiste no conversor analógico para digital MAX11634 com interface digital SPI e quatro canais analógicos. Este AD possui referência interna de 4.096V e 12 bits de resolução, o que leva a uma resolução de 1mV/LSB e a uma excursão de sinal de [0;4096] mV. [19]

Este módulo possui ainda um circuito de acondicionamento de sinal que permite que a gama dos sinais de entrada seja de [-2.5;2.5] V. Este circuito encontra-se representado nas figuras 5.2 e 5.3.

As especificações da interface AD são as seguintes:

- **Alimentação positiva:** de 8V a 35V
- **Alimentação negativa:** de -8V a -35V
- **Interface digital:** SPI 5V
- **Canais analógicos:** 4 canais
- **Gama de entrada analógica:** [-2.5; 2.5] V
- **Saídas de alimentação:** -6V, 5V e 6V

Na Figura 5.2 está representado o circuito de acondicionamento de sinal utilizado neste módulo. Este circuito é composto por um *buffer* de entrada, que garante elevada impedância de entrada, um divisor resistivo de ganho variável seguido de um elevador de nível e como último estágio um *buffer* seguidor de tensão ligado às entradas analógicas do ADC.

Como *buffer* de entrada é utilizado um TL081 com compensação de *offset*, possibilitando compensar o *offset* do circuito.

O segundo estágio composto por um divisor resistivo variável e por um AD623 que possui entrada de referência. Neste estágio a amplitude do sinal de entrada é adaptada à

amplitude suportada pelo AD [0; 4.096] V, através do divisor resistivo de ganho variável, e o nível médio do sinal é elevado para que fique numa gama totalmente positiva, podendo assim ser aplicado ao MAX11634. [19]

Para gerar a tensão de referência utilizada no AD623 utilizou-se um TL431 e um divisor resistivo ajustável, de forma a poder também através desta tensão ajustar o circuito de forma a levar ao seu bom funcionamento. Este circuito encontra-se representado na Figura 5.3.

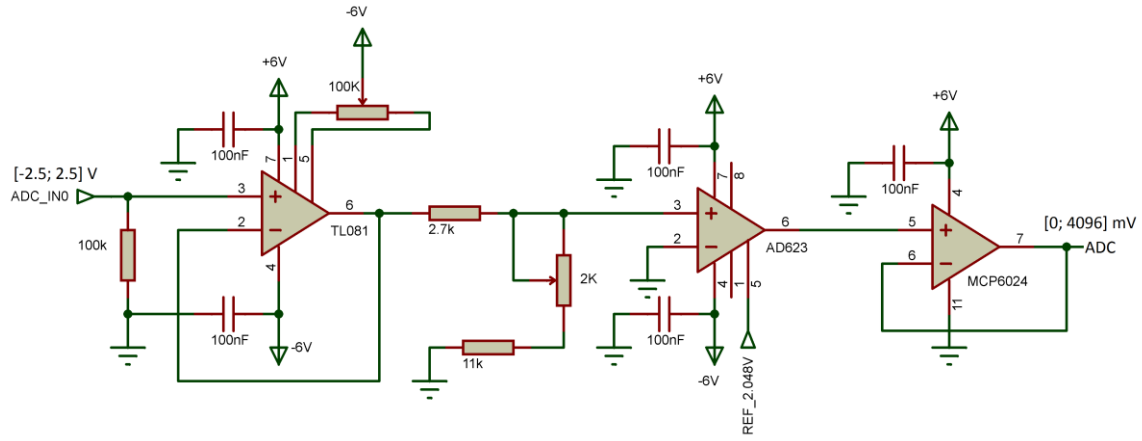

Figura 5.2: Circuito de acondicionamento da interface AD

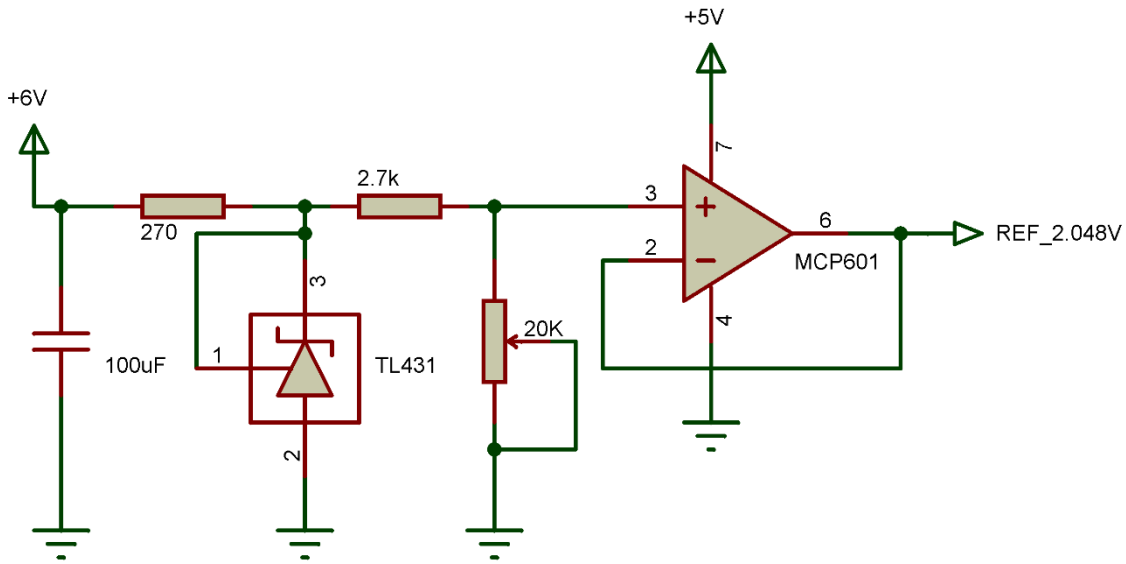

Figura 5.3: Circuito gerador da tensão de referência da interface AD

Experimentalmente observou-se que o AD utilizado apresenta *offset*, sendo que o código gerado por este nunca seria 0 nem 4095, como seria de esperar num funcionamento ideal. Assim o circuito foi calibrado de tal forma que para a gama do sinal de entrada [-2.5;2.5] V a gama do código gerado pelo AD é [20;4093], sendo que para 0V o código gerado é 2056.

O comportamento desta interface traduz-se por uma função afim de domínio [20; 4093] e contradomínio [-2500; 2500], o que se traduz num declive de $\frac{5000}{4073}$. Conhecendo o ponto (20, -2500) obtém-se o valor da ordenada na origem de -2525. Assim, a função que dá a tensão de entrada em milivolts em função do código gerado pelo ADC é

$$y(x) = \frac{5000}{4073}x - 2525$$

O módulo desenvolvido possui quatro conectores de ligação externa. Dois dos conectores são alimentações, sendo um de entrada e outro de saída. Os outros dois conectores são de sinal, sendo um deles de entradas analógicas e o outro a interface digital.

5.2 Interface DA

A interface DA consiste no conversor digital para analógico MCP4822 com interface digital utilizando o protocolo de comunicação SPI e 2 saídas analógicas. Este DA possui referência interna de 2.048 V, 12bits de resolução e um andar de saída com ganho configurável. O ganho do andar de saída pode tomar os valores 1 ou 2. Assim, este DA pode ser configurado para uma excursão do sinal de saída de [0;2.048] V com resolução de 0.5mV/LSB ou para uma excursão de sinal de saída de [0;4.096] V com resolução de 1mV/LSB. [20]

Este módulo possui também um circuito de acondicionamento que permite converter a gama de saída do DA [0;4.096] V para a gama [-2.5; 2.5] V. Este circuito de acondicionamento consiste num AD623 com a entrada de referência utilizada para anular o nível DC do sinal, e com ganho regulável, através de uma resistência em série com um potenciómetro. Este circuito encontra-se na Figura 5.4.

A interface DA tem as seguintes especificações:

- **Alimentação positiva:** de 8V a 35V
- **Alimentação negativa:** de -8V a -35V
- **Interface digital:** SPI 5V
- **Canais analógicos:** 2 canais
- **Gama de saída analógica:** [-2.5; 2.5]V
- **Saídas de alimentação:** -6V, 5V e 6V

Para gerar a tensão de referência utilizou-se o circuito da Figura 5.5, que consiste num TL431 para gerar uma tensão de -2.5V seguido de um potenciómetro que permite variar essa tensão na gama [-2.5,0] V.

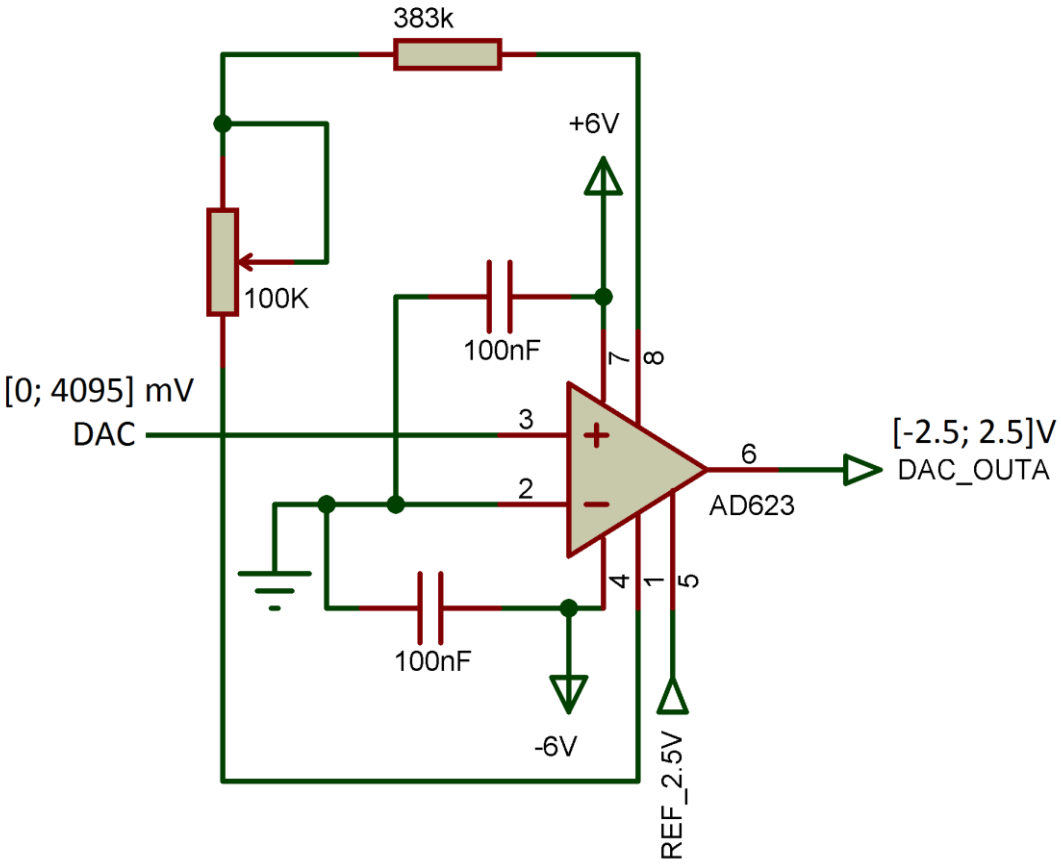

Figura 5.4: Circuito de acondicionamento da interface DA

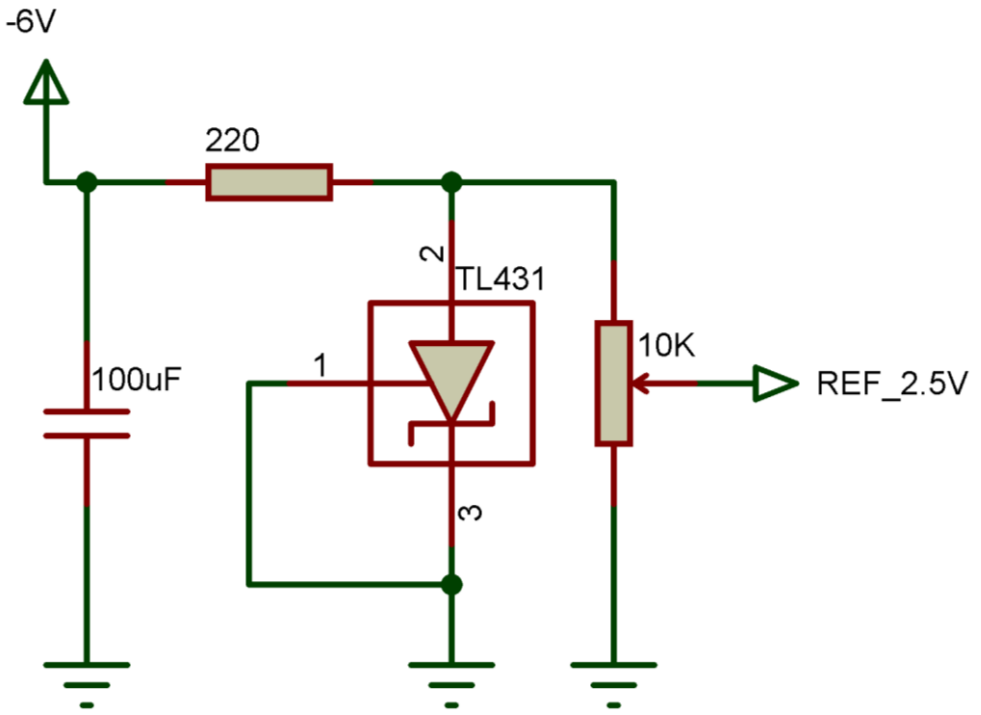

Figura 5.5: Circuito gerador da tensão de referência da interface DA

O circuito da Figura 5.5 foi projetado para gerar uma tensão de referência variável capaz de alimentar dois circuitos integrados (CI) AD623. A resistência da entrada de referência deste CI é de 100KΩ, sendo a resistência de entrada quando estão os dois CIs ligados ao circuito de 50KΩ. O potenciómetro utilizado para variar a tensão de referência, foi escolhido de forma a não interferir no bom funcionamento do CI TL431, optando-se pelo valor de 10KΩ. Para a resistência utilizou-se um valor de 220Ω sendo a corrente que por ela passa de 16mA. A resistência de carga do circuito, composta pelos dois AD623 e pelo potenciómetro, varia entre 8.3KΩ e 10KΩ, variando a corrente consumida pela carga entre 0.25mA e 0.3mA. Assim, a corrente que flui pelo TL431 varia entre 15.7mA e 15.75mA, garantindo o bom funcionamento deste circuito, uma vez que o TL431 funciona para correntes entre 1mA e 100mA. [21] [22]

Experimentalmente observou-se que o DA utilizado não apresenta um comportamento ideal, verificando-se a existência de *offset* de saída e de um ganho intrínseco, levando a que a correspondência entre o código digital e a tensão de saída não seja a esperada. Assim o circuito foi calibrado de tal forma que a saída de ambos os canais seja [-2.5; 2.5] V, para uma gama do sinal digital de [0, 4095] no canal A e [4, 4095] no canal B.

Tal como acontece na interface AD, o comportamento da interface DA traduz-se por uma função afim de domínio [0, 4095] para o canal A e [4, 4095] para o canal B, enquanto o contradomínio é [-2500; 2500] para ambos os canais. Isto traduz-se num declive de $\frac{4095}{5000}$ para o canal A e de $\frac{4091}{5000}$ para o canal B. Conhecendo os pontos (-2500, 0) e (-2500, 4) para os canais A e B respetivamente, obtém-se os valores das ordenadas na origem de 2048 e 2050. Assim as funções que dão o código a enviar para o DAC em função da tensão em milivolts desejada na saída do módulo são

$$y(x) = \frac{4095}{5000}x + 2048\ (canal\ A)$$

$$y(x) = \frac{4091}{5000}x + 2050\ (canal\ B)$$

O módulo desenvolvido possui quatro conectores de ligação externa. Dois dos conectores são alimentações, sendo um de entrada e outro de saída. Os outros dois conectores são de sinal, sendo um deles de saídas analógicas e outro a interface digital.

6 *Software* Desenvolvido

Todo o *software* foi desenvolvido em Linux.

Para comunicação com a interface AD/DA foi desenvolvida uma biblioteca *user-space* que permite facilitar a sua utilização. Esta biblioteca é constituída por 4 módulos:

- *Device-drivers* GPIO
- *Device-drivers* SPI
- Controlo do DAC MCP4822
- Controlo do ADC MAX11634

A organização desta biblioteca encontra-se representada na Figura 6.1

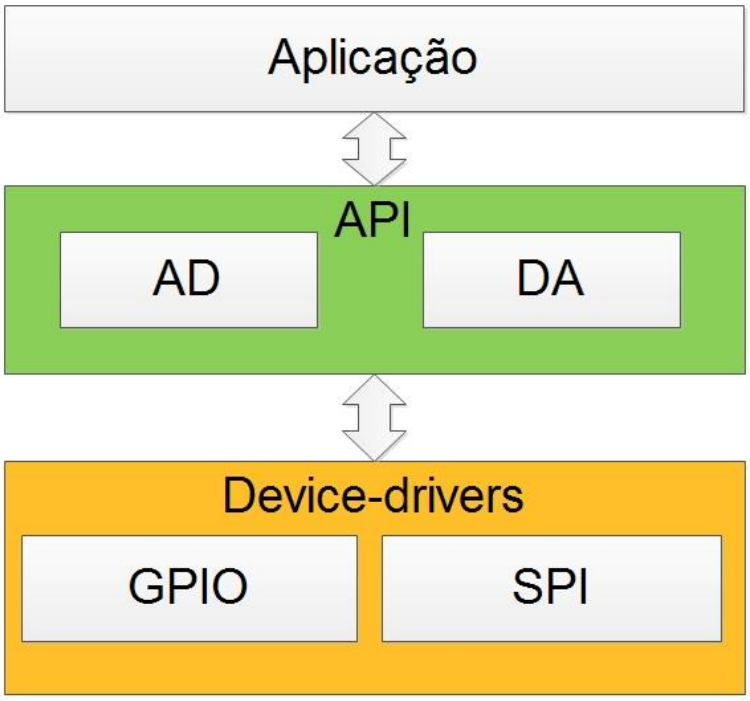

Figura 6.1: Organização da biblioteca desenvolvida

6.1 Device-Drivers GPIO

Este módulo permite utilizar o controlador de GPIO de uma forma simples, proporcionando uma completa abstração da camada física, não sendo necessário conhecer a arquitetura do sistema nem a forma como é feito o acesso aos registos do periférico. A API desenvolvida para este periférico permite a sua completa configuração. Para isso foram desenvolvidos vários métodos, destacando-se os seguintes:

- **INP_GPIO(g)** – definir o GPIOg como porto de entrada.
- **OUT_GPIO(g)** – definir o GPIOg como porto de saída.
- **SET_GPIO_ALT(g,a)** – definir o GPIOg com a função alternativa ‘a’.
- **GPIO_SET(g)** – definir o nível lógico do porto de saída ‘g’ como ‘1’.
- **GPIO_CLR(g)** – definir o nível lógico do porto de saída ‘g’ como ‘0’.
- **GPIO_READ(g)** – ler o nível lógico do porto de entrada ‘g’.
- **gpio_open()** – inicializar o *device-driver*. Deve ser utilizado antes de qualquer operação sobre o periférico.
- **gpio_close()** – liberta o acesso ao periférico. Deve ser utilizado quando o periférico já não é necessário.

O número do GPIO está de acordo com a Figura 3.2.

A utilização do periférico obedece aos seguintes passos:

1. Para ter acesso ao periférico é necessário utilizar a função gpio_open(). Só assim podem ser utilizados os outros métodos disponíveis.
2. Os portos que se pretendem utilizar devem ser configurados como entrada, saída ou configuração alternativa.
3. Os portos configurados como entrada e saída podem agora ser lidos e escritos. Os portos configurados para funções alternativas são agora controlados pelos respetivos periféricos.
4. Para finalizar, quando deixa de ser necessário aceder ao periférico deve ser utilizada a função gpio_close().

NOTA: As funções gpio_open() e gpio_close() não fazem nenhuma configuração, apenas habilitam e inibem o acesso aos registos do periférico. Assim, após a chamada destas funções, todas as configurações do periférico são mantidas, tais como o estado dos portos ou as suas funções.

6.2 Device-Drivers SPI

Tal como o *device-driver* GPIO este módulo permite utilizar o módulo SPI de uma forma simples, proporcionando uma completa abstração da camada física, não sendo

necessário conhecer a arquitetura do sistema nem a forma como é feito o acesso aos registos de controlo do periférico. Os *drivers* desenvolvidos para este periférico permitem a sua completa configuração. Para isso foram desenvolvidos os seguintes métodos:

- **spi_Config(int cs_pol, int freq, int clk_pol, int clk_pha)** – configurar o periférico. Os argumentos desta função encontram-se descritos na Tabela 6.1.
- **spi_SendReceive(char* tx_buff, char* rx_buff, int data_len, int device)** – realizar uma transferência. Os argumentos desta função encontram-se descritos na Tabela 6.2.
- **spi_open()** – inicializar o *device-driver*. Deve ser utilizado antes de qualquer operação sobre o periférico.
- **spi_close()** – libertar o acesso ao periférico. Deve ser utilizado quando o periférico já não é necessário.
- **spi_RXFifoClean()** – limpar a FIFO de receção.

Argumento	Descrição
cs_pol	Polaridade do *chip select*. '1' – CS active high '0' – CS active low
freq	Frequência de relógio desejada. Máximo: 250 MHz Mínimo: 3.815 KHz
clk_pol	Polaridade do sinal de relógio. '1' – Estado de repouso do sinal a '1' ´0´ - Estado de repouso do sinal a '0'
clk_pha	Fase do sinal de relógio. '1' – Primeira transição do SCLK no início do bit de dados ´0´ - Primeira transição do SCLK a meio do bit de dados

Tabela 6.1: Argumentos da função spi_Config()

Argumento	Descrição
tx_buff	Ponteiro para o *buffer* de dados a enviar.
rx_buff	Ponteiro para o *buffer* onde devem ser guardados os dados a receber.
data_len	Número de *bytes* a transferir.
device	Linha de CS a utilizar. 0 – CS0 1 – CS1

Tabela 6.2: Argumentos da função spi_SendReceive()

A utilização do periférico obedece aos seguintes passos:

1. Para ter acesso ao periférico é necessário utilizar a função spi_open(). Só assim podem ser utilizados os outros métodos disponíveis.

2. O periférico deve ser configurado, recorrendo à função spi_Config(), de acordo com a Tabela 6.1.

3. Podem ser realizadas transferências utilizando a função spi_SendReceive(). Pode também ser utilizada a função spi_RXFifoClean() para limpar a FIFO de receção.

4. Para finalizar, quando deixa de ser necessário aceder ao periférico deve ser utilizada a função spi_close().

NOTA: As funções spi_open() e spi_close() utilizam os *device-drivers* GPIO para obter controlo sobre os portos do conector P1. Assim, o utilizador não tem que se preocupar com a configuração dos GPIO.

6.3 Módulo DA

A biblioteca (API) desenvolvida contém um conjunto de ferramentas que permitem utilizar o módulo DA de uma forma simples, proporcionando uma abstração dos *device-drivers* utilizados. Assim é possível configurar totalmente o módulo, e definir o valor analógico de saída.

Esta biblioteca implementa também uma estrutura de dados denominada mcp4822 que possibilita a utilização de vários módulos com configurações diferentes em simultâneo. Esta estrutura caracteriza completamente o módulo DA desenvolvido, contendo as suas configurações. A utilização desta API obriga à utilização desta estrutura, pois os métodos

disponibilizados utilizam uma variável deste tipo para identificar o módulo com o qual estão a comunicar. Os campos desta estrutura estão representados na Tabela 6.3.

A composição desta estrutura é gerida pelos métodos desta API, não sendo da responsabilidade do utilizador. Este apenas deve utilizar esta estrutura de dados para identificar cada um dos módulos que está a utilizar.

O módulo DA pode operar em dois modos distintos. Estes modos determinam a forma como a atualização da tensão de saída é realizada. Estes modos são:

- **Síncrono** – Neste modo, a atualização da tensão de saída dos dois canais analógicos é feita em simultâneo e controlada pelo sinal LDAC. Assim, é possível enviar o novo valor de tensão para ambos os canais, e atualizar ambos em simultâneo.

- **Assíncrono** – Neste modo, assim que o novo valor de tensão é enviado para o módulo, o valor de saída do canal analógico é atualizado. No modo assíncrono não é assim possível atualizar ambos os canais em simultâneo.

Campo	Descrição
gain	Ganho do andar de saída do DAC.
load_type	Modo de funcionamento do módulo DA SYNC ASYNC
ldac	Utilização do sinal LDAC disponibilizado pelo módulo. (não utilizado ou controlado pelo módulo)
cs	Linha de CS a utilizar para este dispositivo.

Tabela 6.3: Estrutura de dados mcp4822

Os métodos disponibilizados por esta API são:

- **dac_init(mcp4822* dvc, int cs, int ldac)** – inicializar a API. Os argumentos desta função encontram-se descritos na Tabela 6.4.

- **dac_config(mcp4822* dvc, unsigned int g, int ltype, int freq)** configurar o módulo DA. Os argumentos desta função encontram-se descritos na Tabela 6.5

- **dac_setVal(mcp4822* dvc, int val, int dac)** – definir o valor desejado na saída analógica do módulo DA. Os argumentos desta função encontram-se descritos na Tabela 6.6.

- **dac_load()** – coloca a tensão na saída analógica do módulo quando este está configurado em modo síncrono.

Argumento	Descrição
dvc	Estrutura de dados (mcp4822) correspondente ao dispositivo a utilizar.
cs	Linha de CS a utilizar para este dispositivo. 0 – CS0 1 – CS1 2 – CS2
ldac	Utilização do sinal LDAC disponibilizado pelo módulo. '1' – sinal não utilizado ´0´ – sinal controlado pelo módulo

Tabela 6.4: Argumentos da função dac_init()

Argumento	Descrição
dvc	Estrutura de dados (mcp4822) correspondente ao dispositivo a utilizar.
g	Ganho do andar de saída do CI MCP4822. Deve ser 2 para o módulo desenvolvido.
ltype	Modo de funcionamento do módulo DA. SYNC ASYNC
freq	Frequência do sinal de relógio da comunicação SPI.

Tabela 6.5: Argumentos da função dac_config()

Argumento	Descrição
dvc	Estrutura de dados (mcp4822) correspondente ao dispositivo a utilizar.
val	Valor de tensão desejado. Máximo: 2500mV Mínimo: -2500mV
dac	Canal de saída do módulo. DAC_A – canal A DAC_B – canal B

Tabela 6.6: Argumentos da função dac_setVal()

Antes de começar a utilizar cada um dos módulos devem ser seguidos os seguintes passos:

1. Definir uma variável do tipo mcp4822 para cada módulo DA a utilizar.
2. Inicializar a API, para cada módulo, utilizando a função dac_init().
3. Configurar cada um dos módulos recorrendo à função dac_config().

Assim que cada módulo esteja configurado, pode ser utilizado normalmente. O valor da tensão de saída de cada canal analógico pode ser definido utilizando a função dac_setVal(). No caso de o módulo estar configurado para funcionar em modo síncrono, é necessário também utilizar a função dac_load() para atualizar o valor de saída de ambos os canais.

O módulo pode ser reconfigurado em qualquer altura.

6.4 Módulo AD

A biblioteca desenvolvida contém um conjunto de ferramentas que permitem utilizar o módulo AD de uma forma simples, proporcionando uma abstração dos *device-drivers* utilizados. Assim é possível configurar totalmente o módulo, e ler o valor analógico de entrada.

Esta biblioteca implementa também uma estrutura de dados denominada max11634 que possibilita a utilização de vários módulos com configurações diferentes em simultâneo. Esta estrutura caracteriza completamente o módulo AD desenvolvido, contendo as suas configurações. A utilização desta API obriga à utilização desta estrutura, pois os métodos

disponibilizados utilizam uma variável deste tipo para identificar o módulo com o qual estão a comunicar. Os campos desta estrutura estão representados na Tabela 6.7.

A composição desta estrutura é gerida pelos métodos desta API, não sendo da responsabilidade do utilizador. Este apenas deve utilizar esta estrutura de dados para identificar cada um dos módulos que está a utilizar.

Campo	Descrição
Conversion_REG	Valor do registo conversion do DAC.
Setup_REG	Valor do registo setup do DAC.
Averaging_REG	Valor do registo averaging do DAC.
Reset_REG	Valor do registo reset do DAC.
Unipolar_REG	Valor do registo unipolar do DAC.
Bipolar_REG	Valor do registo bipolar do DAC.
cs	Linha de CS a utilizar para este dispositivo.
eoc	Utilização do sinal EOC disponibilizado pelo módulo. (não utilizado ou controlado pelo módulo)
cnvst	Utilização do sinal CNVST disponibilizado pelo módulo. (não utilizado ou controlado pelo módulo)

Tabela 6.7: Estrutura de dados max11634

Os métodos disponibilizados por esta API são:

- **adc_init(max11634* dvc, int cs, int eoc, int cnvst, int freq)** – inicializar a API. Os argumentos desta função encontram-se descritos na Tabela 6.8.
- **adc_clkMode(max11634* dvc, unsigned int mode)** configurar o modo do sinal de relógio do módulo DA. Os argumentos desta função encontram-se descritos na Tabela 6.9.
- **adc_scanMode(max11634* dvc, unsigned int mode)** – configurar o modo de amostragem do módulo DA. Os argumentos desta função encontram-se descritos na Tabela 6.10.
- **adc_configAvg(max11634* dvc, unsigned int state, unsigned int nConv,unsigned int scanCount)** – configurar o registo Averaging do CI

MAX11634. Os argumentos desta função encontram-se descritos na Tabela 6.11.

- **adc_refSel(max11634* dvc, unsigned int ref)** – configurar a referência a utilizar. Os argumentos desta função encontram-se descritos na Tabela 6.12.
- **adc_ANSelect(max11634* dvc, unsigned int ch)** – selecionar o canal analógico. Os argumentos desta função encontram-se descritos na Tabela 6.13.
- **adc_startConv(max11634* dvc)** – Iniciar a conversão. Os argumentos desta função encontram-se descritos na Tabela 6.14.
- **adc_getVal(max11634* dvc, int* buff)** – lê o valor convertido. Se a conversão ainda não tiver terminado a função bloqueia até que o novo valor esteja disponível. Os argumentos desta função encontram-se descritos na Tabela 6.15.

Argumento	Descrição
dvc	Estrutura de dados (max11634) correspondente ao dispositivo a utilizar.
cs	Linha de CS a utilizar para este dispositivo. 0 – CS0 1 – CS1 2 – CS2
eoc	Utilização do sinal EOC disponibilizado pelo módulo. ‘1’ – sinal não utilizado ´0´ – sinal controlado pelo módulo
cnvst	Utilização do sinal CNVST disponibilizado pelo módulo. ‘1’ – sinal não utilizado ´0´ – sinal controlado pelo módulo
freq	Frequência do sinal de relógio da comunicação SPI.

Tabela 6.8: Argumentos da função adc_init()

Argumento	Descrição
dvc	Estrutura de dados (max11634) correspondente ao dispositivo a utilizar.
mode	Sinal de relógio para conversão e amostragem. 0 – relógio interno e amostragem temporizada internamente. 1 – relógio interno e amostragem temporizada pelo CNVST. 2 – relógio interno e amostragem temporizada internamente. 3 – relógio externo e amostragem temporizada pelo SCLK.

Tabela 6.9: Argumentos da função adc_clkMode()

Argumento	Descrição
dvc	Estrutura de dados (max11634) correspondente ao dispositivo a utilizar.
mode	Modo de conversão desejado. 0 – converter canais de 0 a N. 1 – converter canais de N a 3. 2 – converter canal N repetidamente. O registo averaging determina o numero de conversões. 3 – converter canal N apenas uma vez.

Tabela 6.10: Argumentos da função adc_scanMode()

Argumento	Descrição
dvc	Estrutura de dados (max11634) correspondente ao dispositivo a utilizar.
state	Ativar/desativar a função de média ‘0’ – desativar ‘1’ – ativar
nConv	Valor a colocar no campo NAVG do registo Averaging [19]
scanCount	Valor a colocar no campo NSCAN do registo Averaging [19]

Tabela 6.11: Argumentos da função adc_init()

Argumento	Descrição
dvc	Estrutura de dados (max11634) correspondente ao dispositivo a utilizar.
ref	Valor a colocar no campo REFSEL do registo Setup [19]

Tabela 6.12: Argumentos da função adc_refSel()

Argumento	Descrição
dvc	Estrutura de dados (max11634) correspondente ao dispositivo a utilizar.
ch	Canal analógico a selecionar.

Tabela 6.13: Argumentos da função adc_ANSelect()

Argumento	Descrição
dvc	Estrutura de dados (max11634) correspondente ao dispositivo a utilizar.

Tabela 6.14: Argumentos da função adc_startConv()

Argumento	Descrição
dvc	Estrutura de dados (max11634) correspondente ao dispositivo a utilizar.
buff	Ponteiro para o *buffer* onde deve ser guardado o valor lido.

Tabela 6.15: Argumentos da função adc_getVal()

A utilização de cada um dos módulos deve seguir os seguintes passos:

1. Definir uma variável do tipo max11634 para cada módulo AD a utilizar.
2. Inicializar a API, para cada módulo, utilizando a função adc_init().
3. Configurar cada um dos módulos com os parâmetros desejados, utilizando as funções adc_clkMode(), adc_scanMode(), adc_configMode(), adc_refSel() e adc_ANSelect().

Assim que cada módulo esteja configurado, pode ser utilizado normalmente. A sua utilização passa por utilizar a função adc_startConv() para iniciar a conversão, e por utilizar a função adc_getVal() para ler o valor convertido.

O módulo pode ser reconfigurado em qualquer altura.

7 Testes Realizados

7.1 Desempenho do Linux na execução de tarefas periódicas

O Sistema Operativo utilizado é Raspbian, Debian GNU/Linux 7.0 (wheezy), com kernel Linux 3.6.11+.

Todos os testes foram realizados com as tarefas a executarem segundo a política de agendamento SCHED_FIFO e prioridade 99.

A capacidade do sistema operativo para executar tarefas periódicas foi avaliada com a realização de quatro testes, utilizando duas tarefas. Estes testes foram divididos em dois grupos de dois testes cada um, sendo que cada teste correspondente à execução de uma tarefa.

No grupo 1 os testes foram realizados, enquanto se procedia à transferência de um ficheiro entre o Raspberry Pi e um computador remoto. Já no grupo de testes 2, os testes foram realizados sem que fosse realizada a transferência.

As tarefas utilizadas nos testes encontram-se descritas nos fluxogramas da Figura 7.1 e da Figura 7.2.

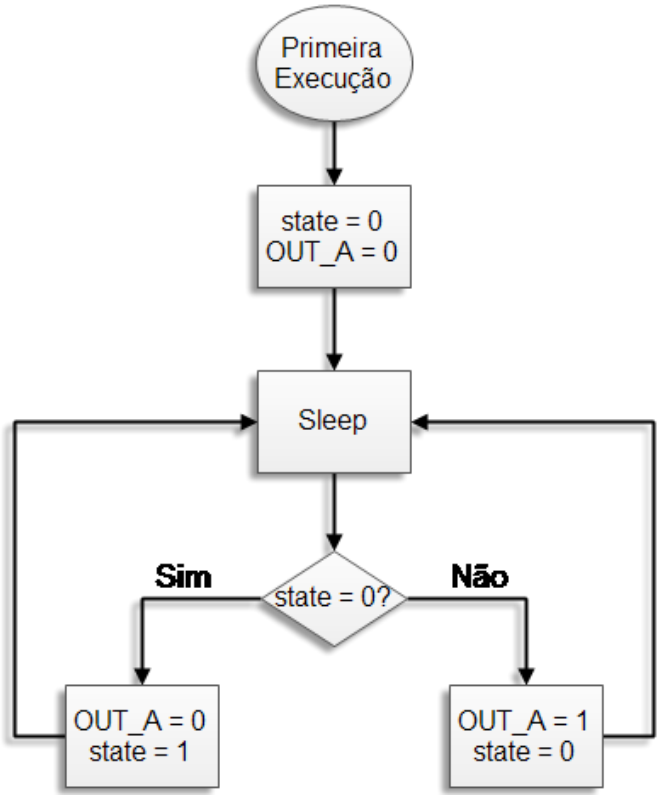

Figura 7.1: Fluxograma da tarefa utilizada nos testes 1 e 3.

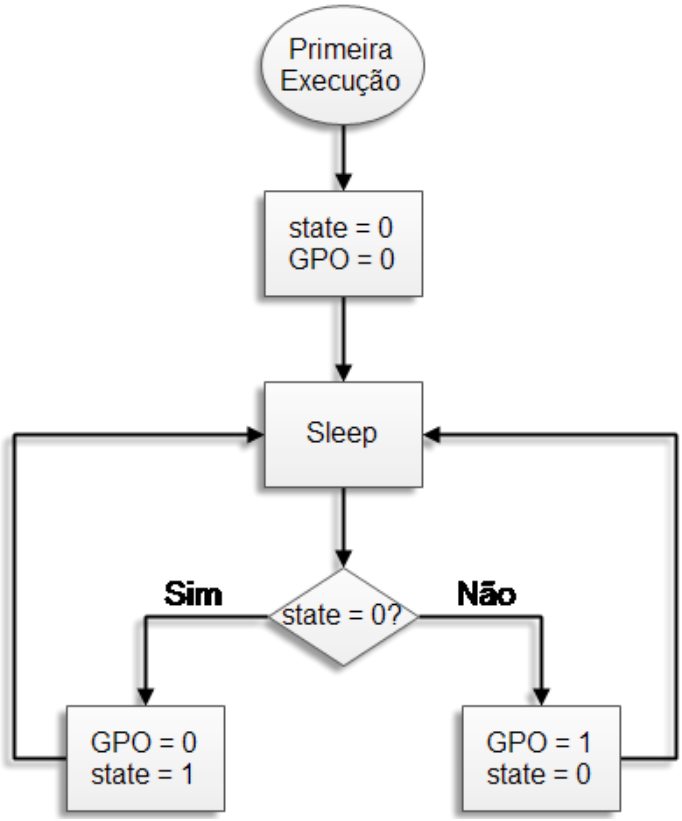

Figura 7.2: Fluxograma da tarefa utilizada nos testes 2 e 4.

Estas tarefas são muito semelhantes, sendo que ambas geram um sinal de relógio com período de 2ms, diferenciando-se apenas pela forma como o sinal é gerado. Enquanto uma das tarefas utiliza o módulo DA para gerar esse sinal (OUT_A), a outra utiliza um dos portos de I/O (GPO).

A execução periódica das tarefas é conseguida utilizando um *interval timer* do Linux que ao expirar lança um sinal ao qual é associada a execução da tarefa pretendida. Assim configurando o *timer* com o período desejado consegue-se a execução de tarefas periódicas.

Para medir o tempo entre execuções consecutivas da tarefa periódica, ou seja, entre 2 transições consecutivas do sinal gerado é utilizado um microcontrolador PIC32MX230F064B.

O microcontrolador foi configurado para funcionar com o oscilador interno de 8MHz ligado à PLL, sendo a frequência do *clock* de 32MHz. O *peripheral bus clock* (PBCLK) foi configurado para 8MHz. Foi utilizado um *timer* para medir o tempo, configurado para uma frequência de 1MHz, sendo a resolução de 1µs.

Os testes foram realizados apenas uma vez, sendo que para cada teste foram realizadas 2500 medições.

7.1.1 Grupo de testes 1

Neste grupo 1 os teste foram realizados em simultâneo com a transferência de um ficheiro entre o *Raspberry Pi* e um computador remoto.

No teste 1 o valor analógico de uma das saídas do módulo DA é alternado entre 0 e 2500mV de cada vez que a tarefa periódica é executada. Assim é gerado o sinal de relógio com período de 2ms e amplitude 2500mV.

No teste 2 é utilizado um porto digital do *Raspberry Pi* para gerar o sinal de relógio, sendo o seu nível lógico invertido a cada iteração da tarefa periódica.

O ficheiro utilizado tem de tamanho 36601KB e foi transferido por SFTP.

7.1.2 Grupo de testes 2

No segundo grupo de testes repetiram-se os testes do grupo 1, sendo que desta vez não foi efetuada a transferência de um ficheiro em paralelo, não havendo assim uma influência significativa de outros processos na utilização do processador.

7.1.3 Resultados obtidos

Os resultados obtidos encontram-se na Tabela 7.1.

Testes		***Jitter* médio (µs)**	***Jitter* máximo (µs)**
Grupo 1	**Teste 1**	607,7	19954
	Teste 2	735,4	20742
Grupo 2	**Teste 1**	4,1	276
	Teste 2	3,2	189

Tabela 7.1: *Jitter* obtido para os testes realizados

Pela análise dos resultados obtidos, verifica-se que para um mesmo grupo de testes os valores de *jitter* são muito próximos para ambos os testes, podendo assim inferir-se que a utilização da interface analógica ou digital não influencia o desempenho na execução de tarefas periódicas.

A influência da execução de tarefas em paralelo é perfeitamente visível nos resultados obtidos. Os valores do *jitter* médio e máximo para o grupo de testes 1 são muito

superiores quando é efetuada a transferência de um ficheiro em simultâneo, chegando a ser 73% e 2000% do período da tarefa respetivamente, enquanto para o grupo de testes 2 estes valores são de aproximadamente 0.4% e 27% respetivamente.

Estes resultados demonstram que a interface desenvolvida não prejudica o desempenho temporal do Raspberry Pi. No entanto, é notório que apesar de as tarefas de teste serem executadas com classe de prioridade de tempo-real, a sua execução é influenciada pela transferência do ficheiro a decorrer em simultâneo. Assim se observa a falta de garantias temporais do Linux.

7.2 Atraso do sistema desenvolvido

O atraso do sistema desenvolvido mediu-se através da captura de um sinal pelo módulo AD e sua reprodução em tempo real pelo módulo DA como representado na Figura 7.3.

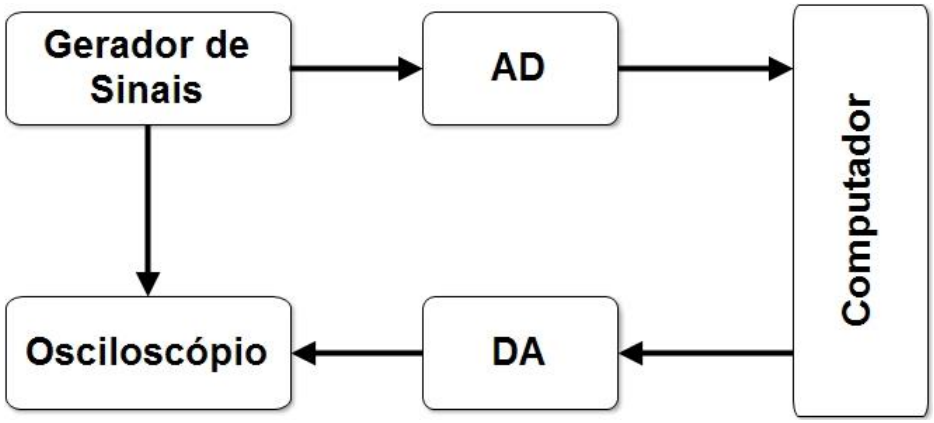

Figura 7.3: Montagem utilizada para testar o atraso do sistema

O atraso entre a obtenção do sinal pelo módulo AD e a sua reprodução pelo módulo DA foi medido utilizando uma tarefa periódica apresentada na Figura 7.4, medindo o tempo que cada iteração desta tarefa demorava a executar. Este teste foi executado apenas uma vez, tendo sido obtidos 2500 valores para o atraso. Os resultados obtidos encontram-se na Tabela 7.2.

	Média	Desvio padrão	Máximo
Atraso (µs)	28	2.1	78

Tabela 7.2: Atraso entre a captura de um sinal e a sua reprodução

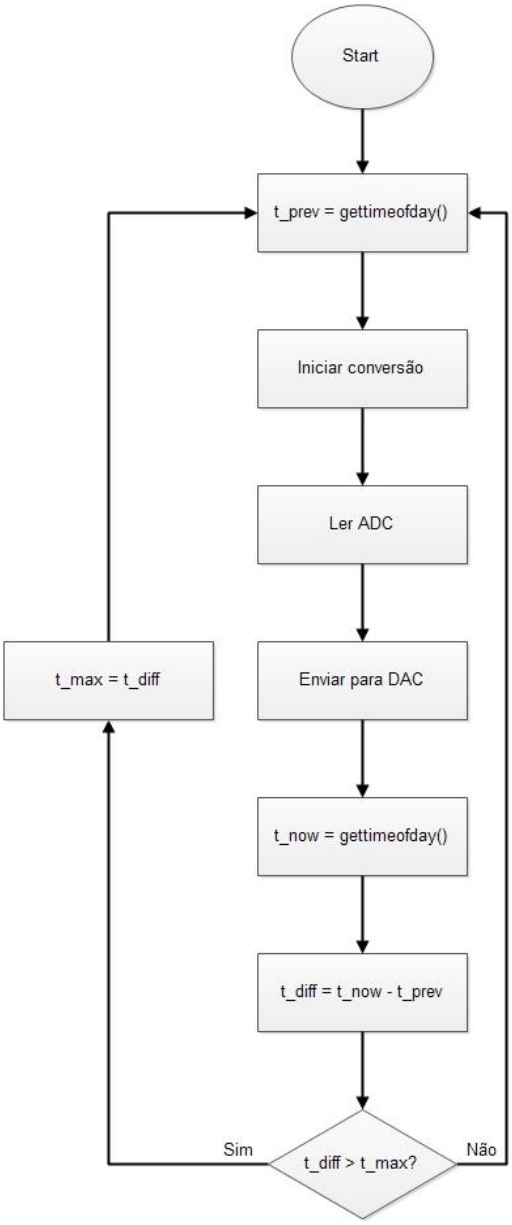

Figura 7.4: Fluxograma da tarefa utilizada para medir o atraso do sistema

Foi também realizado um teste com o objetivo de medir o atraso introduzido por cada um dos módulos desenvolvidos.

Para medir o atraso no módulo AD, foi medido o tempo passado entre a ordem de conversão enviada ao módulo AD, e o momento em que a função que devolve o valor convertido retorna. O valor médio obtido foi de 14.6 µs, e o máximo de 64 µs.

Para medir o atraso nó módulo DA, foi utilizado um GPO como referência. Utilizou-se uma tarefa periódica para geral um sinal de relógio, em que o sinal era gerado num porto digital definido como saída e também no módulo DA. Assim com um osciloscópio mediu-

se o atraso nas transições do sinal gerado pelo módulo DA em relação ao sinal gerado pelo porto digital. O valor do atraso medido foi de 24 µs.

7.3 Teste com simulador de processos contínuos

O simulador de processos contínuos é um dispositivo analógico que simula sistemas físicos. O simulador utilizado neste trabalho, simula sistemas de segunda ordem e possui 3 potenciómetros que permitem variar os coeficientes do sistema a simular.

Neste trabalho, o valor dos coeficientes não é uma preocupação, pelo que os potenciómetros são colocados em posições aleatórias no início dos testes e serão mantidos nessas posições ao longo de todos os testes.

Para testar o simulador de processos contínuos gerou-se um sinal em MATLAB, apresentado como sinal de referência na Figura 7.5, que foi guardado num ficheiro. Esse sinal pode assim ser carregado no *software* desenvolvido para testar o simulador. Na Figura 7.5 pode ver-se o sinal aplicado ao simulador de processos contínuos e a sua resposta.

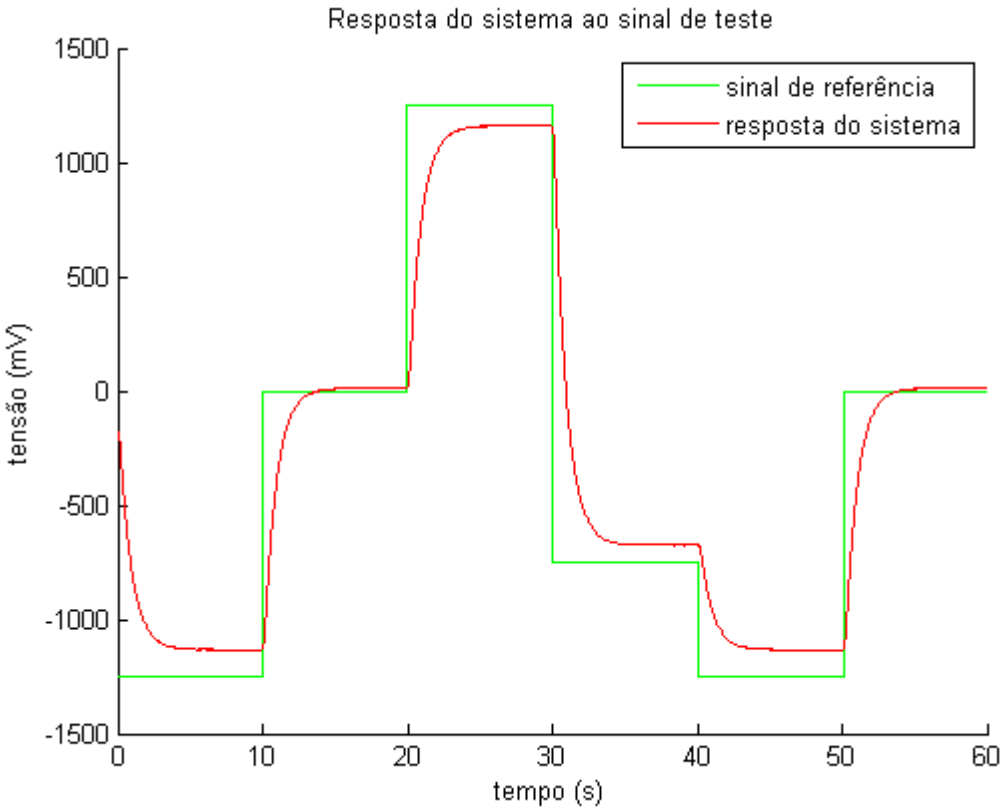

Figura 7.5: Sinal de teste e resposta do simulador

Para testar o desempenho do sistema desenvolvido na execução de algoritmos de controlo, foram implementados dois controladores, um simples controlador proporcional

de ganho unitário e um controlador proporcional integral (PI), para testar o desempenho temporal do sistema ao executar um algoritmo de controlo mais complexo. Em ambos os casos foi utilizado um intervalo de amostragem (h) de 10ms.

Para cada um dos controladores foi medido o período de execução da tarefa de controlo e o tempo de execução do algoritmo de controlo. Ambos os testes foram executados apenas uma vez, sendo que cada uma das variáveis medida 6000 vezes, o que corresponde aproximadamente a uma duração de 60 segundos. Os dados encontram-se na Tabela 7.3 e Tabela 7.4.

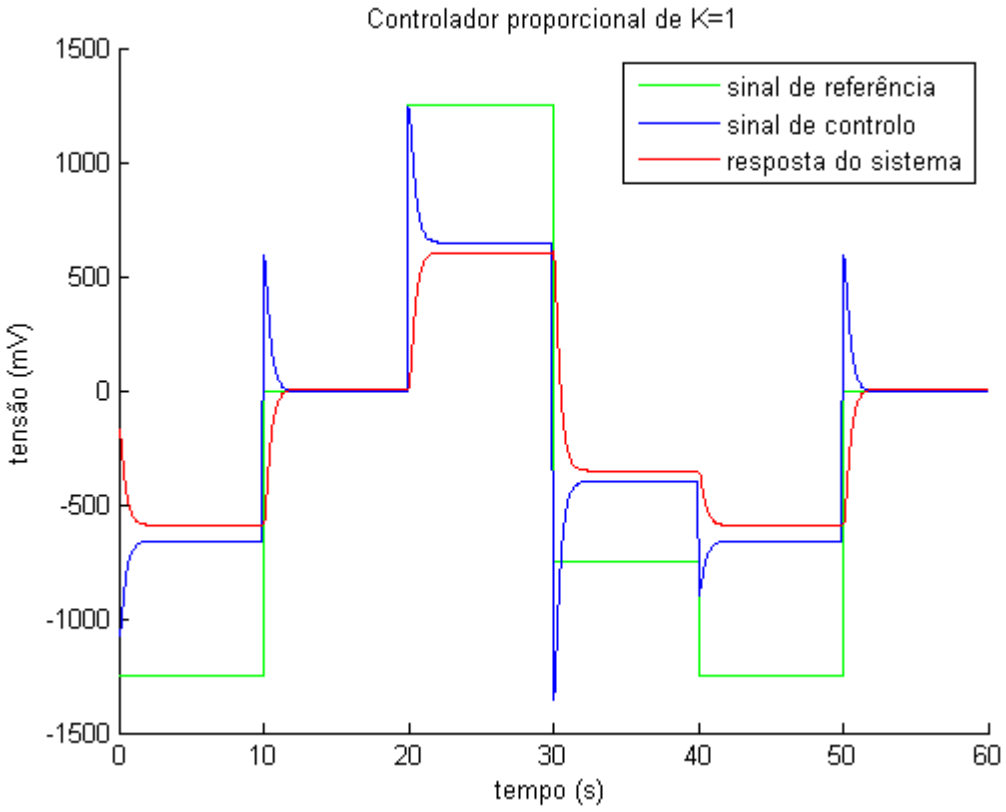

Figura 7.6: Controlador proporcional de ganho unitário

	Média	Desvio padrão	Máximo
Periodo (ms)	10	0.015	10.6
Tempo de execução (µs)	38	5.1	270

Tabela 7.3: Período e tempo de execução do controlador proporcional

Para o controlador PI foram utilizados os seguintes parâmetros:

- **K:** 1
- **Ti:** 0.45

Estes valores foram encontrados experimentalmente de forma a obter um comportamento do sistema, sem oscilação. Não foram utilizadas nenhumas técnicas de *tunning*.

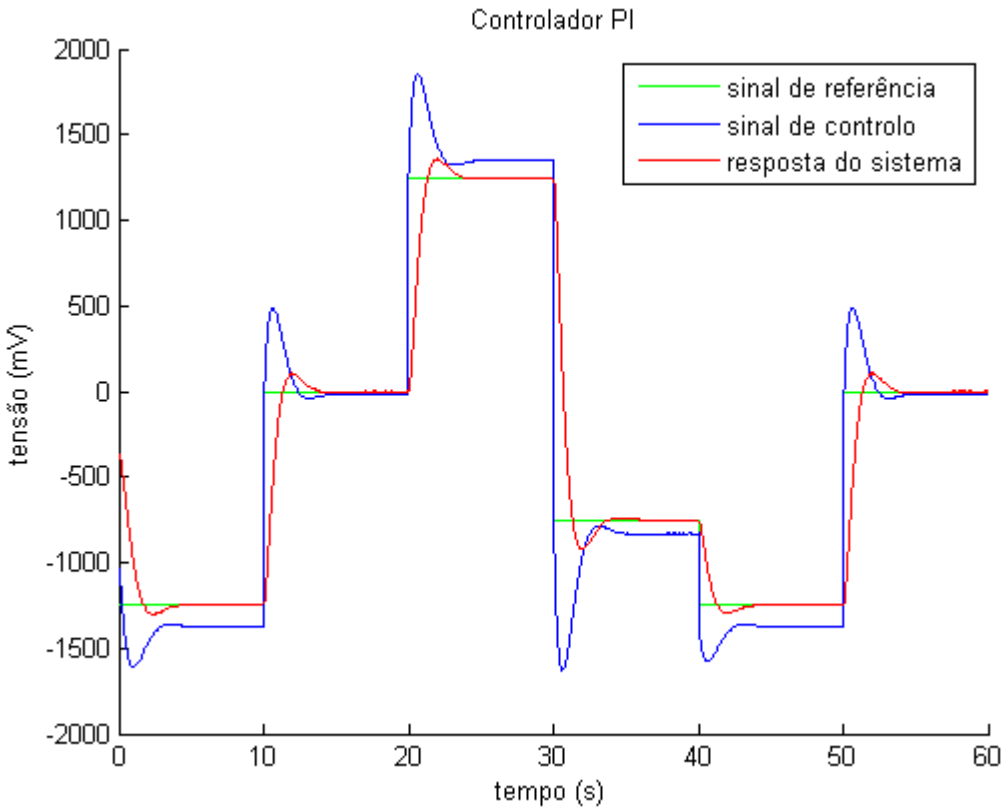

Figura 7.7: Controlador PI

	Média	Desvio padrão	Máximo
Periodo (ms)	10	0.012	10.2
Tempo de execução (µs)	45.6	6.8	390

Tabela 7.4: Período e tempo de execução do controlador proporcional integral

Analisando os valores obtidos para o período, verifica-se que a periodicidade da tarefa de controlo está muito próxima da desejada. Em conjunto com os valores do tempo de execução, pode afirmar-se que é possível baixar o intervalo de amostragem para 1ms

sem que o tempo de execução da tarefa ultrapasse o seu período de ativação. Esta afirmação deixa de ser válida para casos em que existem outras aplicações a fazer uso intensivo do processador, pois como vimos na secção 7.1.3, o *jitter* médio pode ultrapassar os 700µs e o *jitter* máximo pode ultrapassar os 20ms.

Para algoritmos de controlo mais complexos, o tempo de execução da tarefa será também mais elevado, elevando assim o valor mínimo para o intervalo de amostragem.

8 Conclusões e trabalho futuro

Como era objetivo deste trabalho, foi implementada uma interface analógica para o Raspberry Pi, juntamente com o desenvolvimento de APIs para interação com a mesma. A interface desenvolvida é modular, tendo sido decomposta em 3 partes:

- Módulo AD
- Módulo DA
- Módulo de interface (para ligar circuitos digitais com níveis lógicos de tensões diferentes)

Esta abordagem permite uma maior versatilidade do trabalho desenvolvido, pois estes módulos são completamente independentes, podendo ser utilizados nos mais variados projetos.

As APIs e os *device-drivers* desenvolvidos foram também pensados para permitir esta versatilidade, possibilitando a utilização de vários módulos iguais em simultâneo.

Verificou-se também que os módulos desenvolvidos apresentam características temporais satisfatórias, permitindo a sua utilização em aplicações de controlo de sistemas. No entanto os testes realizados deixaram claro o mau comportamento do Linux em tarefas de tempo-real.

Nem todos os objetivos deste trabalho foram cumpridos, ficando por implementar um módulo de I/O digital. Apesar de este módulo não ter sido implementado, foi desenvolvida uma API, baseada no CI MCP23S17, que não foi testada.

As API desenvolvidas beneficiariam de algumas modificação ao nível do suporte de múltiplos módulos em simultâneo, uma vez que no seu estado atual a API apenas controla de forma automática um dos módulos, deixando o controlo dos outros ao encargo do utilizador.

Como trabalho a desenvolver fica também a adaptação do *software* desenvolvido para Xenomai, com o objetivo de melhorar as características temporais.

9 Bibliografia

[1] W. Cedeño e P. Laplante, “An Overview of Real-Time Operating Systems,” *Journal of the Association for Laboratory Automation,* 2007.

[2] A. M. Mota, Introdução ao Controlo por Computador, Aveiro, 2013.

[3] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, Springer, 2011.

[4] P. B. R. Pedreiras, “Sistemas de Tempo-Real,” 2013.

[5] G. C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Pisa: Springer, 2011.

[6] RASPBERRY PI FOUNDATION, “WHAT IS A RASPBERRY PI?,” RASPBERRY PI FOUNDATION, [Online]. Available: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/. [Acedido em 18 Abril 2015].

[7] Tufty, “RPi Low-level peripherals,” eLinux, 28 Janeiro 2012. [Online]. Available: http://elinux.org/RPi_Low-level_peripherals. [Acedido em Janeiro 2014].

[8] Broadcom Corporation, “BCM2835 ARM Peripherals,” Cambridge, 2012, pp. 4-7.

[9] M. Haardt, “MEM(4),” 2 Abril 1993. [Online]. Available: http://man7.org/linux/man-pages/man4/mem.4.html. [Acedido em 27 Outubro 2015].

[10] D. Eckhardt, M. Haardt, I. Jackson, G. Banks e M. Kerrisk, “OPEN(2),” 1992. [Online]. Available: http://man7.org/linux/man-pages/man2/open.2.html. [Acedido em 27 Outubro 2015].

[11] Broadcom Corporation, “BCM2835 ARM Peripherals,” Cambridge, 2012, pp. 89-108.

[12] Broadcom Corporation, “BCM2835 ARM Peripherals,” Cambridge, 2012, pp. 148-159.

[13] K. Yaghmour, J. Masters, G. Ben-Yossef e P. Gerum, *Building Embedded Linux Systems,* O'Reilly, 2008.

[14] M. Kerrisk, P. Zijlstra e 2. J. Lelli, “sched.7,” 2014. [Online]. Available: http://man7.org/linux/man-pages/man7/sched.7.html. [Acedido em 20 Outubro 2015].

[15] M. Kerrisk e P. Baudis, “sigevent.7,” 2009. [Online]. Available: http://man7.org/linux/man-pages/man7/sigevent.7.html. [Acedido em Outubro 2015].

[16] Linux Foundation, “ timer_create.2,” 2009. [Online]. Available: http://man7.org/linux/man-pages/man2/timer_create.2.html. [Acedido em Outubro 2015].

[17] T. Koenig e M. Kerrisk, “ signal.7,” 2008. [Online]. Available: http://man7.org/linux/man-pages/man7/signal.7.html. [Acedido em Outubro 2015].

[18] M. Kerrisk, “sigaction.2,” 2005. [Online]. Available: http://man7.org/linux/man-pages/man2/sigaction.2.html. [Acedido em Outubro 2015].

[19] Maxim Integrated Products, *12-Bit, 300ksps ADCs with Differential,* Sunnyvale, 2011.

[20] Microchip Technology Inc., “MCP4821/4822,” 2005.

[21] Motorola, Inc., *TL431, A, B Series,* Denver.

[22] Analog Devices, Inc., *AD623,* Norwood, 2008.

[23] RASPBERRY PI FOUNDATION, “MODEL B,” RASPBERRY PI FOUNDATION, [Online]. Available: https://www.raspberrypi.org/products/model-b/. [Acedido em 18 Abril 2015].

[24] A. Burkepile, “Raspberry Pi Airplay Tutorial,” RAYWENDERLICH, 8 Novembro 2013. [Online]. Available: http://www.raywenderlich.com/44918/raspberry-pi-airplay-tutorial. [Acedido em 18 Abril 2015].

[25] Catedu, [Online]. Available: http://educativa.catedu.es/44700165/aula/archivos/repositorio/500/518/html/Unidad_02/imagenes/73.jpg. [Acedido em 12 Maio 2014].

[26] P. Jain, “RTOS - Real Time Operating System,” [Online]. Available: http://www.engineersgarage.com/articles/rtos-real-time-operating-system. [Acedido em 2014].

10 Anexos

10.1 Esquemático da Interface AD

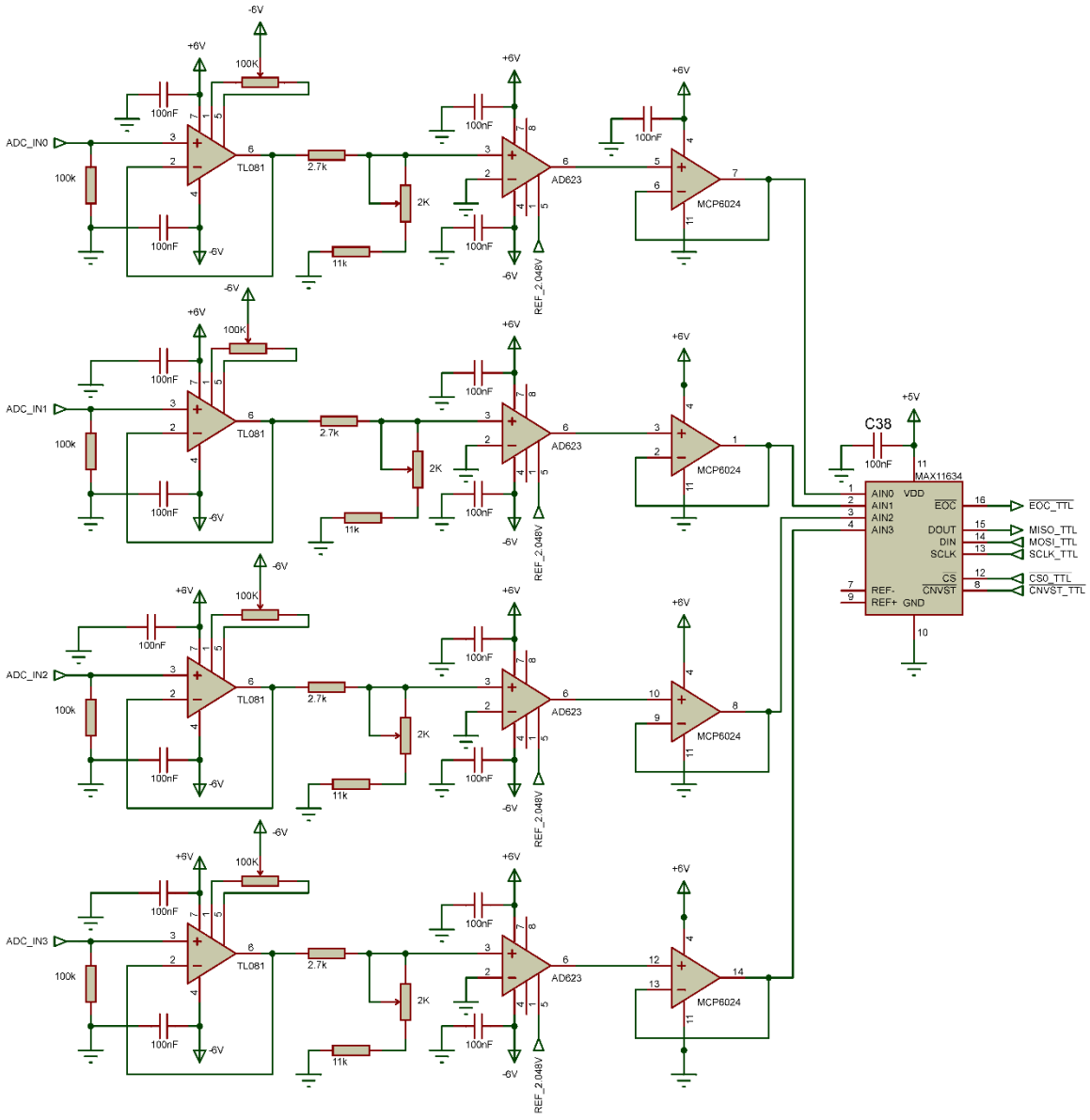

Figura 10.1: Circuito da Interface AD

10.2 Esquemático de Interface DA

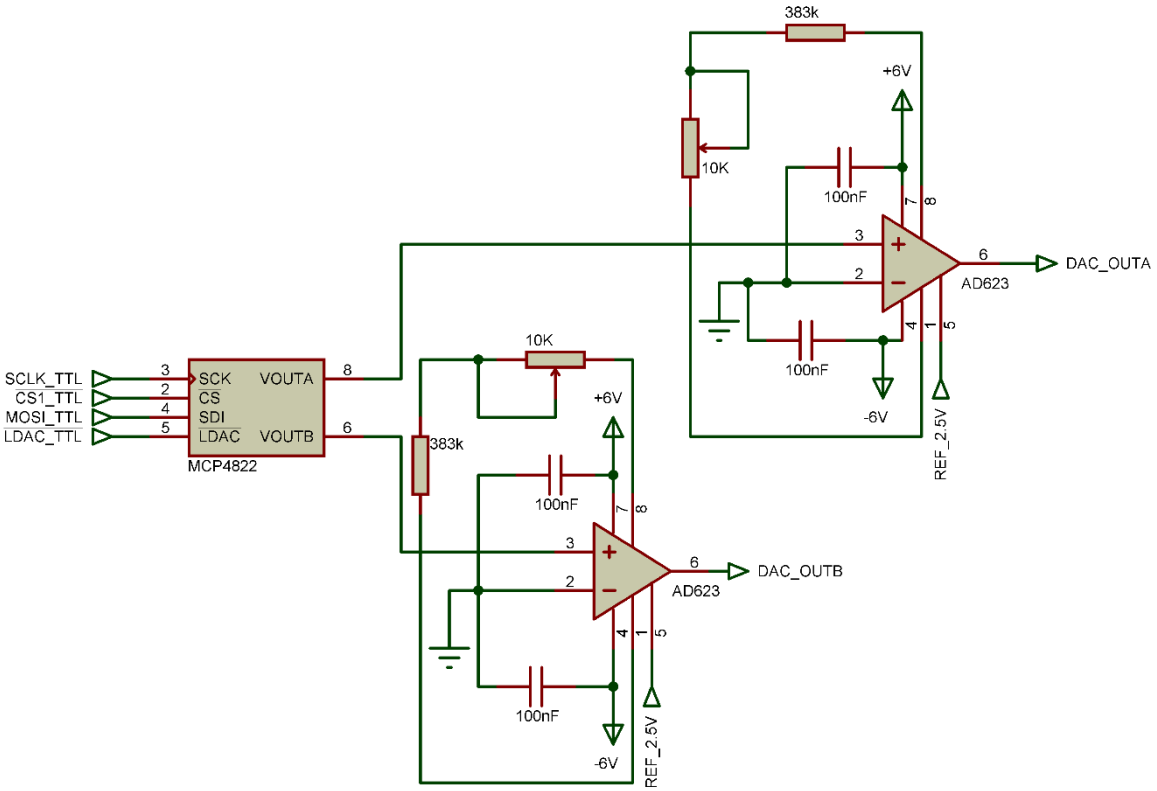

Figura 10.2: Circuito da Interface AD

10.3 Registos do módulo SPI

Bit(s)	Field Name	Description	Type	Reset
31:26		***Reserved*** *- Write as 0, read as don't care*		
25	LEN_LONG	Enable Long data word in Lossi mode if DMA_LEN is set 0= writing to the FIFO will write a single byte 1= wrirng to the FIFO will write a 32 bit word	RW	0x0
24	DMA_LEN	Enable DMA mode in Lossi mode	RW	0x0
23	CSPOL2	Chip Select 2 Polarity 0= Chip select is active low. 1= Chip select is active high.	RW	0x0
22	CSPOL1	Chip Select 1 Polarity 0= Chip select is active low. 1= Chip select is active high.	RW	0x0
21	CSPOL0	Chip Select 0 Polarity 0= Chip select is active low. 1= Chip select is active high.	RW	0x0
20	RXF	RXF - RX FIFO Full 0 = RXFIFO is not full. 1 = RX FIFO is full. No further serial data will be sent/ received until data is read from FIFO.	RO	0x0
19	RXR	RXR RX FIFO needs Reading (full) 0 = RX FIFO is less than full (or not active TA = 0). 1 = RX FIFO is or more full. Cleared by reading sufficient data from the RX FIFO or setting TA to 0.	RO	0x0
18	TXD	TXD TX FIFO can accept Data 0 = TX FIFO is full and so cannot accept more data. 1 = TX FIFO has space for at least 1 byte.	RO	0x1
17	RXD	RXD RX FIFO contains Data 0 = RX FIFO is empty. 1 = RX FIFO contains at least 1 byte.	RO	0x0
16	DONE	Done transfer Done 0 = Transfer is in progress (or not active TA = 0). 1 = Transfer is complete. Cleared by writing more data to the TX FIFO or setting TA to 0.	RO	0x0
15	TE_EN	Unused	RW	0x0
14	LMONO	Unused	RW	0x0
13	LEN	LEN LoSSI enable The serial interface is configured as a LoSSI master. 0 = The serial interface will behave as an SPI master. 1 = The serial interface will behave as a LoSSI master.	RW	0x0
12	REN	REN Read Enable read enable if you are using bidirectional mode. If this bit is set, the SPI peripheral will be able to send data to this device. 0 = We intend to write to the SPI peripheral. 1 = We intend to read from the SPI peripheral.	RW	0x1

11	ADCS	ADCS Automatically Deassert Chip Select 0 = Don t automatically deassert chip select at the end of a DMA transfer chip select is manually controlled by software. 1 = Automatically deassert chip select at the end of a DMA transfer (as determined by SPIDLEN)	RW	0x0
10	INTR	INTR Interrupt on RXR 0 = Don t generate interrupts on RX FIFO condition. 1 = Generate interrupt while RXR = 1.	RW	0x0
9	INTD	INTD Interrupt on Done 0 = Don t generate interrupt on transfer complete. 1 = Generate interrupt when DONE = 1.	RW	0x0
8	DMAEN	DMAEN DMA Enable 0 = No DMA requests will be issued. 1 = Enable DMA operation. Peripheral generates data requests. These will be taken in four-byte words until the SPIDLEN has been reached.	RW	0x0
7	TA	Transfer Active 0 = Transfer not active./CS lines are all high (assuming CSPOL = 0). RXR and DONE are 0. Writes to SPIFIFO write data into bits -0 of SPICS allowing DMA data blocks to set mode before sending data. 1 = Transfer active. /CS lines are set according to CS bits and CSPOL. Writes to SPIFIFO write data to TX FIFO.TA is cleared by a dma_frame_end pulse from the DMA controller.	RW	0x0
6	CSPOL	Chip Select Polarity 0 = Chip select lines are active low 1 = Chip select lines are active high	RW	0x0
5:4	CLEAR	CLEAR FIFO Clear 00 = No action. x1 = Clear TX FIFO. One shot operation. 1x = Clear RX FIFO. One shot operation. If CLEAR and TA are both set in the same operation, the FIFOs are cleared before the new frame is started. Read back as 0.	RW	0x0
3	CPOL	Clock Polarity 0 = Rest state of clock = low. 1 = Rest state of clock = high.	RW	0x0
2	CPHA	Clock Phase 0 = First SCLK transition at middle of data bit. 1 = First SCLK transition at beginning of data bit.	RW	0x0
1:0	CS	Chip Select 00 = Chip select 0 01 = Chip select 1 10 = Chip select 2 11 = Reserved	RW	0x0

Figura 10.3 - Registo CS

Bit(s)	Field Name	Description	Type	Reset
31:0	DATA	DMA Mode (DMAEN set) If TA is clear, the first 32-bit write to this register will control SPIDLEN and SPICS. Subsequent reads and writes will be taken as four-byte data words to be read/written to the FIFOs Poll/Interrupt Mode (DMAEN clear, TA set) Writes to the register write bytes to TX FIFO. Reads from register read bytes from the RX FIFO	RW	0x0

Figura 10.4 - Registo FIFO

Bit(s)	Field Name	Description	Type	Reset
31:16		***Reserved*** *- Write as 0, read as don't care*		
15:0	CDIV	Clock Divider SCLK = Core Clock / CDIV If CDIV is set to 0, the divisor is 65536. The divisor must be a power of 2. Odd numbers rounded down. The maximum SPI clock rate is of the APB clock.	RW	0x0

Figura 10.5 - Registo CLK

Bit(s)	Field Name	Description	Type	Reset
31:16		***Reserved*** *- Write as 0, read as don't care*		
15:0	LEN	Data Length The number of bytes to transfer. This field is only valid for DMA mode (DMAEN set) and controls how many bytes to transmit (and therefore receive).	RW	0x0

Figura 10.6 - Registo DLEN

Bit(s)	Field Name	Description	Type	Reset
31:4		***Reserved*** *- Write as 0, read as don't care*		
3:0	TOH	This sets the Output Hold delay in APB clocks. A value of 0 causes a 1 clock delay.	RW	0x1

Figura 10.7 - Registo LTOH

Bit(s)	Field Name	Description	Type	Reset
31:24	RPANIC	DMA Read Panic Threshold. Generate the Panic signal to the RX DMA engine whenever the RX FIFO level is greater than this amount.	RW	0x30
23:16	RDREQ	DMA Read Request Threshold. Generate A DREQ to the RX DMA engine whenever the RX FIFO level is greater than this amount, (RX DREQ is also generated if the transfer has finished but the RXFIFO isn t empty).	RW	0x20
15:8	TPANIC	DMA Write Panic Threshold. Generate the Panic signal to the TX DMA engine whenever the TX FIFO level is less than or equal to this amount.	RW	0x10
7:0	TDREQ	DMA Write Request Threshold. Generate a DREQ signal to the TX DMA engine whenever the TX FIFO level is less than or equal to this amount.	RW	0x20

Figura 10.8 - Registo DC