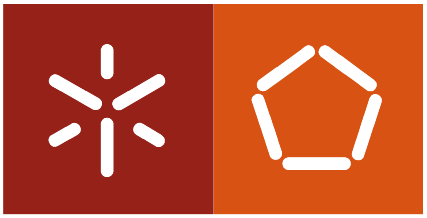
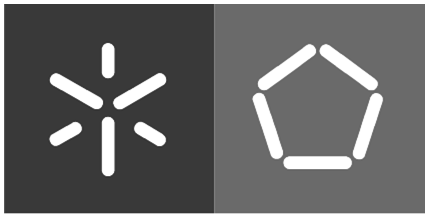


**Universidade do Minho**
Escola de Engenharia
Departamento de Electrónica Industrial

António Herculano de Jesus Moreira

# Sistema embebido no controlo e monitorização de dados de uma célula de trabalho industrial para aparafusamento automático de PCBs

Setembro 2009



**Universidade do Minho**
Escola de Engenharia
Departamento de Electrónica Industrial

António Herculano de Jesus Moreira

# Sistema embebido no controlo e monitorização de dados de uma célula de trabalho industrial para aparafusamento automático de PCBs

Dissertação submetida à Universidade do Minho para obtenção do grau de Mestre em Electrónica Industrial e Computadores

Dissertação realizada sob a orientação científica do
**Professor Jaime Francisco Cruz Fonseca**
**Professor associado do Departamento de Electrónica Industrial da Universidade do Minho**

Setembro 2009

"Algo só é impossível até que alguém duvide e acabe por provar o contrário."
[Albert Einstein]

# AGRADECIMENTOS

Ao longo deste projecto, várias pessoas e entidades contribuíram tornando possível o seu desenvolvimento, às quais quero agradecer.

Quero agradecer aos meus familiares, o meu pai António Moreira e à minha mãe Margarida Miranda, por todo o esforço e apoio que me deram durante todos estes anos como estudante.

Ao Departamento de Electrónica Industrial (DEI) da Escola de Engenharia da Universidade do Minho pela formação e condições que me foram dadas.

Ao meu orientador, Doutor Jaime Fonseca, por todo o apoio e orientação dada ao longo do projecto, permitindo que este projecto se torna-se numa aplicação industrial.

Aos meus colegas e amigos, pelas ajudas dadas, questões construtivas e companheirismo em todos os momentos.

A empresa Silgal que disponibilizou o espaço e condições para o desenvolvimento do protótipo.

# RESUMO

Hoje em dia, frequentemente é requerido aos sistemas industriais, que controlem um determinado processo industrial e monitorizem os dados relevantes do mesmo, através de um ambiente visual amigável. Para tal, é normalmente usado um PLC (Controlador Lógico Programável) no controlo do processo de forma a garantir os tempos de execução e um PC (computador pessoal) para monitorização dos dados.

A implementação deste tipo de solução apresenta as seguintes dificuldades para o programador: necessita de conhecer e implementar o protocolo de comunicação entre o PLC e o PC; necessita trabalhar em dois ambientes de programação distintos e de aprender duas linguagens de programação, uma de mais baixo nível para o PLC e uma de alto nível para o PC.

Por outro lado, o espaço disponível para alguns sistemas de controlo é reduzido, tornando este tipo de solução muito difícil de implementar.

Para minimizar as dificuldades apresentadas pelo sistema anterior, foi usada uma placa com sistema embebido, baseada no SO (Sistema Operativo) Windows CE. No desenvolvimento da aplicação de software quer para o controlo quer para a monitorização foi utilizado o Microsoft Visual Studio e uma única linguagem de alto nível o C++.

Esta solução, permitiu resolver as dificuldades relacionadas com o tempo de aprendizagem de diferentes linguagens e ambientes de programação assim como a necessidade de conhecer e implementar o protocolo de comunicação entre os dispositivos PLC e PC.

Para a realização deste projecto efectuou-se uma pesquisa de mercado, analisando-se os diferentes sistemas embebidos existentes, os SO e o hardware presente em cada placa. Foram ainda estudadas e avaliadas as possibilidades de ligação a diferentes GUI's (Graphical User Interface) uma vez que na aplicação final proposta era necessário monitorizar alguma informação relevante do processo. Este sistema foi inicialmente aplicado numa máquina protótipo com três eixos independentes e depois numa célula de trabalho industrial para aparafusamento de PCB's (Placa de Circuito Impresso).

# ABSTRACT

Nowadays, in Industrial controls systems, frequently is required that it controls some industrial process and monitoring the relevant data about the process, using a friendly visual environment. For that, is normally used a PLC (Programmable Logical Controller) for control the process and assure the execution times and a PC for monitoring the data.

The implementation of this solution present the following difficulties: the programmer need know the communication protocol between the two development platforms, PLC and the PC; the programmer need learn two different programming languages - the low level PLCs language and a high level PC language.

For other side in some cases the space that is reserved to control systems is reduced, making this kind of solution very difficult to implement.

The embedded system used is based in the Windows CE OS (Operating System) and the software application developed is based in a high level programming language (C++), using the Microsoft Visual Studio environment.

This solution overcomes the difficulties present above reducing the developed time because the programmer only use one hardware platform and only needed to learn one programming language, C++.

To realize this project a market survey is carried out, analyzing the different embedded boards available, analyzing the OS (Operating System) and the hardware on each board. A visual environment was required, since the application tested needed to show information's, various GUI (Graphical User Interface) were studied and tested, pointing out the best and easiest GUI to use. The hardware needed to develop is tested on an industrial application.

In the end the system was tested on an initial prototype and then in an industrial screw machine for PCB's (Printed Circuit Board), registering the results and analyzing them.

# ÍNDICE

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ACRÓNIMOS E SIGLAS

| | | |
|---|---|---|
| PLC | → | Controladores Lógicos Programáveis |
| PC | → | *Personal Computer* |
| HMI | → | *Human Machine Interface* |
| SoC | → | *System on Chip* |
| DSP | → | *Digital Signal Processor* |
| FPGA | → | *Field-Programmable Gate Array* |
| $I^2C$ | → | *Inter-Integrated Circuit* |
| SPI | → | *Serial Peripheral Interface Bus* |
| LCD | → | *Liquid Ccrystal Display* |
| V | → | *Volt* |
| Hz | → | *Hertz* |
| IL | → | *Intermediate Language* |
| CLR | → | *Common Language Runtime* |
| JIT | → | *Just-In-Time* |
| GB | → | *Garbage Collector* |
| MFC | → | *Microsoft Foundation Classes* |
| E/S | → | Entrada/Saída |
| MISO | → | *Master Input, Slave Output* |
| MOSI | → | *Master Output, Slave Input* |
| SS | → | *Slave Select* |
| SCL | → | *Serial Clock* |
| SDA | → | *Serial Data* |
| LED | → | Diodo Emissor de Luz |
| ISR | → | *Interrupt Service Routine* |
| IST | → | *Interrupt Service Thread* |
| PWM | → | *Pulse Width Modulation* |
| PCB | → | *Printed Circuit Board* |
| DC | → | *Direct Current* |
| OO | → | Orientado a Objectos |

# CAPÍTULO 1

# 1 INTRODUÇÃO

Tradicionalmente os sistemas de controlo e monitorização de dados usam PLC's (Controladores Lógicos Programáveis) para o controlo do processo e PC's e/ou outros dispositivos HMI (*Human Machine Interface*) na monitorização de dados. Esta implementação torna o sistema mais trabalhoso e demorado de implementar devido ao facto de ser necessário aprender diferentes linguagens de programação e plataformas de hardware [1], mesmo assim estes sistemas estão a ser cada vez mais utilizados em tarefas de controlo e monitorização devido ao aumento de performance dos PCs [2].

Hoje em dia, essa solução continua a ser válida, contudo cada vez menos é economicamente viável, devido essencialmente ao elevado tempo de desenvolvimento necessário quando se trabalha com diferentes linguagens de programação e diferentes plataformas.

Para que novos produtos possam ser economicamente competitivos, o uso de sistemas embebidos com capacidades de tempo-real começam a ser cada vez mais utilizados para reduzir o tempo de desenvolvimento. Com o aumento da adopção do Windows CE em ambientes industriais, devido às suas capacidades de tempo-real e interface amigável surge uma nova oportunidade de redução do tempo de desenvolvimento e consequentemente dos custos dos sistemas industriais neles baseados [3].

## 1.1 OBJECTIVOS DO TRABALHO

As placas baseadas em sistemas embebidos são, hoje em dia, um mercado em grande expansão o que leva a que este tipo de sistemas seja adoptado cada vez com mais frequência por um elevado número de empresas como solução para diversas aplicações.

A aplicação aqui descrita tem como base uma célula de trabalho industrial, Figura 1, que tem como objectivo o aparafusamento automático de placas de circuito impresso a uma base ("chassi") própria. Este tipo de tarefa é normalmente efectuado por um robô manipulador cartesiano com uma aparafusadora eléctrica acoplada ou em alguns casos manualmente com uma aparafusadora eléctrica ou pneumática.


Figura 1 Sistema de aparafusamento automático de PCBs

Contudo, para que o sistema seja economicamente mais competitivo, o aparafusamento será efectuado por uma aparafusadora eléctrica, posicionada por três eixos independentes, accionados por motores de passo com controladores independentes, que forma no seu conjunto um robô cartesiano simplificado.

Esta solução torna-se mais económica, apesar de ser necessário um maior tempo de desenvolvimento da máquina devido sobretudo ao projecto das partes mecânicas de suporte e disposição dos eixos e também à programação/configuração individualizada dos controladores dos respectivos eixos a partir de ambiente de programação proprietário.

Nesta tese, são apresentados todos os passos do desenvolvimento de uma célula de aparafusamento automático industrial em que o seu controlo e monitorização foram implementados numa placa com o Windows CE embebido.

Os principais requisitos para este sistema foram:

- Controlo independente dos três eixos.
- Tempo do ciclo de aparafusamento não deve ser superior a 90 segundos.
- O sistema deve garantir a alimentação correcta da aparafusadora.
- Detectar aparafusamentos incorrectos e guardar registos sobre estes eventos.
- Possibilidade de alterar o programa de aparafusamento rapidamente.
- Guardar registos sobre todos os aparafusamentos efectuados.
- Indicar o estado actual de cada parafuso e tempo de ciclo.
- Garantir a interrupção do processo de aparafusamento o mais rapidamente possível sempre que alguma das seguintes situações acorra:

- Porta de acesso aberta ou botão de emergência pressionado.
- Problemas na alimentação do parafuso, na centragem da base (chassi) ou erros nos controladores.

## 1.2 ORGANIZAÇÃO DA TESE

Os temas abordados nesta tese estão organizados da seguinte forma:

No segundo capítulo é inicialmente efectuada uma análise dos sistemas embebidos comerciais mais conhecidos. Em seguida, é efectuado um resumo dos sistemas operativos para sistemas embebidos mais usuais. Na parte final do capítulo é feita uma breve referência aos sistemas de aparafusamento automáticos comerciais.

No terceiro capítulo é descrito em pormenor a plataforma de software e o hardware do sistema embebido aplicado no sistema de aparafusamento automático desenvolvido.

No quarto capítulo é descrito o desenvolvimento do sistema de aparafusamento automático, explicitando-se o hardware seleccionado e descrevendo-se as características mecânicas do sistema de aparafusamento.

No quinto capítulo é efectuada uma descrição dos módulos de software implementados, dos algoritmos de controlo e do ambiente gráfico para monitorização desenvolvidos.

O sexto capítulo termina com as conclusões da análise dos resultados obtidos após a implementação do sistema de aparafusamento e perspectivas futuras do trabalho descrito nesta tese.

# CAPÍTULO 2

# 2 ESTADO DA ARTE

Actualmente o desenvolvimento de sistemas embebidos tem sido normalmente efectuado pelas empresas que tem necessidade de solucionar problemas específicos, como por exemplo a indústria das telecomunicações, os fabricantes de GPS e MP3, entre muitos outros em que os seus produtos requerem bom desempenho, portabilidade, flexibilidade e autonomia [4].

Os sistemas embebidos nascem desta demanda, aparecendo então os designados SoC (*System on Chip*) que possuem todas as características necessárias de um computador vulgar, num pequeno integrado capaz de ser o mais eficiente possível para a aplicação destino.

É através do progresso alcançado com o desenvolvimento dos SoC que foi possível a criação de sistemas embebidos com funcionalidades diversas, direccionados para aplicações específicas e em que antigamente seria necessário utilizar sistemas que ocupavam mais espaço físico, o que os tornava pouco práticos.

Neste capítulo, serão abordados alguns dos mais conhecidos sistemas embebidos existentes no mercado, com as características necessárias ao desenvolvimento da aplicação proposta.

## 2.1 SISTEMAS EMBEBIDOS

### 2.1.1 TORADEX ORCHID

A Toradex é uma empresa sediada na Suíça que desenvolve e comercializa sistemas embebidos standards, possuindo ainda uma gama de produtos que podem ser integrados

Figura 2 Sistema embebido da Toradex (Colibri e Orchid) [5]

com estes sistemas de modo a satisfazer as exigências do mercado.

A Figura 2 mostra um dos sistemas embebidos comercializado por esta empresa. Este sistema é constituído por duas placas, uma que comporta o processador e as memórias e uma outra de expansão que permite obter acesso aos diversos periféricos do processador, tais como: USB, RS232, Ethernet, VGA, $I^2C$, SPI, GPIO, entre outros.

Este sistema embebido é baseado no sistema operativo Windows CE 5.0. O sistema pode ser alimentado desde 7V até 24V e tem facilidades de ligação de um LCD directamente à placa de expansão. O módulo da placa do processador possui um SoC PXA270 a 520Mhz, contudo esta placa pode ser substituída por outros módulos, dependendo da especificidade da aplicação.

Este sistema tem como vantagem a flexibilidade de ligação com os mais diversos dispositivos externos, é modular no que se refere ao processador e possuí um bom suporte pela Toradex, com muitas bibliotecas de funções, exemplos e programas de configuração.

### 2.1.2 AVR32 NGW100

A empresa Atmel possui o sistema embebido NGW100, Figura 3, baseado no SoC AT32AP7000 deste fabricante, possuindo características semelhantes ao processador PXA270, contudo o SoC AT32AP7000 está limitado a uma frequência de 150Mhz.


Figura 3 Sistema embebido da AVR32 NGW100 da Atmel [6]

Foi desenvolvido com o objectivo de fornecer um sistema embebido com um custo reduzido, especialmente vocacionado para tarefas de rede.

As principais características do sistema são as duas portas de Ethernet, uma porta série, $I^2C$, SPI, ligação para LCD, baixo consumo energético, o sistema operativo preferencial é o Linux mas é possível utilizar outros.

A principal vantagem deste sistema é realmente o seu baixo custo quando comparado com outros semelhantes, as duas portas de Ethernet e a flexibilidade do sistema operativo Linux que possui uma vasta comunidade de suporte.

### 2.1.3 BEAGLEBOARD

A BeagleBoard, Figura 4, é um sistema embebido desenvolvido por uma comunidade de software livre em parceria com a Texas Instruments. Usa o SoC OMAP3530 da Texas Instruments, considerado como um dos melhores, visto ser uma conjugação entre um processador ARM Cortex-A8 a 600 MHz com um DSP (*Digital Signal Processor*) com um reduzido consumo de energia.

Figura 4 Sistema embebido BeagleBoard [7]

Foi desenvolvida, com a finalidade de fornecer à comunidade de software livre um sistema embebido com um custo aceitável, de forma a dar oportunidade a todos os programadores de trabalharem num sistema deste tipo que tem a performance de um PC.

Como principais características podem ser destacadas as seguintes: possibilidade de ser alimentado através de uma porta USB, um processador gráfico com suporte para OpenGL, sistema operativo Linux, tamanho bastante reduzido (7.6cm x 7.6cm), uma saída de S-Video e uma saída DVI compatível com a maior parte dos monitores actuais.

Contudo, possui duas desvantagens significativas: não possui nenhuma porta de Ethernet; as tensões de entrada e saída disponíveis são todas a 1.8V.

### 2.1.4 OVERO GUMSTIX

A Overo Gumstix, Figura 5, é um sistema embebido baseado no SoC OMAP3530 da Texas Instruments.

Figura 5 Sistema embebido Overo da Gumstix [8]

Contudo, ao invés da placa BeagleBoard, esta não possui qualquer ligação para periféricos, é necessária uma segunda placa de modo a permitir expor as mais diversas ligações, Figura 6.

Figura 6 Placa de expansão Tobi para o sistema Overo [9]

Este sistema embebido é caracterizado pelo seu tamanho (1.7cm x 5.8cm) muito reduzido, o que possibilita a sua inclusão noutros produtos. O SoC usado possui uma frequência de 600Mhz e ao contrário da BeagleBoard este sistema embebido possuí uma porta de Ethernet através de uma placa de expansão.

A grande vantagem deste sistema é o seu tamanho reduzido e os vários módulos de expansão existentes para diversos propósitos, tornando o sistema flexível e versátil, no entanto, o seu tamanho também se torna uma desvantagem, dependendo da aplicação pretendida, visto que uma placa de expansão é sempre necessária.

### 2.1.5 ARMADEUS APF27

O sistema embebido Armadeus APF27, Figura 7, é baseado no processador Freescale i.MX27 a 400MHz.

Figura 7 Sistema embebido Armadeus APF27 [10]

Este sistema é muito semelhante aos apresentados anteriormente, necessita de uma placa de expansão para se conseguir aceder aos seus periféricos, Figura 8, possui seis portas série, dois módulos de $I^2C$, três módulos de SPI, USB, Ethernet, conexão para LCD, entre outros. A empresa disponibiliza o sistema operativo Linux já configurado e pronto a utilizar para a placa APF27.

Figura 8 Placa de expansão para o sistema Armadeus [11]

A grande vantagem deste sistema embebido é a incorporação de uma FPGA (*Field-Programmable Gate Array*) que permite a implementação de uma aplicação específica (processamento de voz, processamento de sinal, sistema de visão), maximizando a performance total do sistema.

## 2.2 SISTEMAS OPERATIVOS REAL-TIME

### 2.2.1 WINDOWS CE

O sistema operativo Windows CE, Figura 9, é propriedade da empresa Microsoft, sendo utilizado em muitos dos sistemas embebidos. Este sistema operativo tem características distintas relativamente à versão para computadores x86 pois é capaz de correr em sistemas com 1MB de memória e em processadores da família ARM, MIPS e Intel x86.

Vários sistemas embebidos usam o Windows CE, muito devido ao facto deste ser um sistema operativo em tempo-real com 256 níveis de prioridade para *threads* e com um vasto suporte de periféricos. Como tem características de tempo-real, os tempos de atendimento a uma interrupção são bastante reduzidos, normalmente rondam os 10us (microsegundos) [12], podendo este valor variar em função do sistema usado.

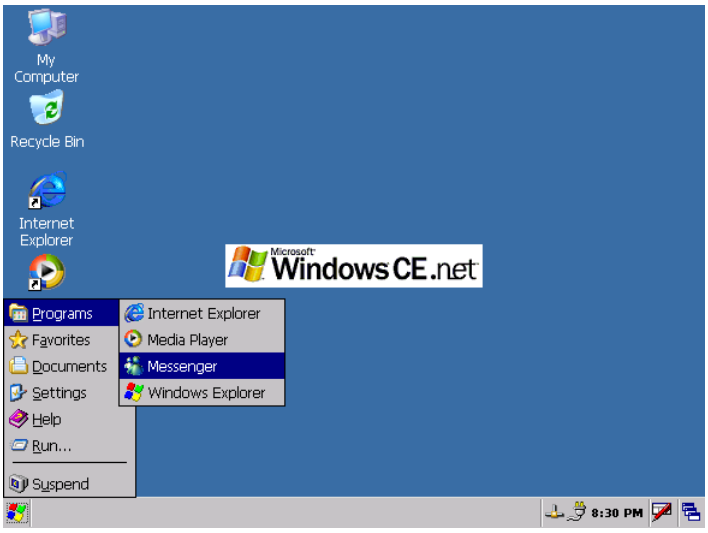



Figura 9 Sistema operativo Windows CE da Microsoft [13]

Em relação às ferramentas disponíveis para o Windows CE temos o Visual Studio da Microsoft. É uma ferramenta muito poderosa, capaz de criar aplicações para sistemas embebidos em C#, C++, VB. Existem também *plugins* capazes de integrar *frameworks* de outras empresas no Visual Studio, como por exemplo a *framework* Qt da Nokia, o que o torna este ambiente ainda mais flexível e completo. Existe ainda o Microsoft eMbedded Visual C++ 4.0, uma ferramenta gratuita mas um pouco desactualizada, contudo, capaz de desenvolver aplicações em C++ para o Windows CE.

### 2.2.2 VxWorks e Montavista

Os sistemas operativos *VxWorks* e *Montavista*, são sistemas baseados em Linux, contudo, modificados para que estes se tornem sistemas operativos de tempo-real, com um kernel multitarefa, um tempo de atendimento das interrupções externas muito reduzido, métodos de sincronismo, entre outras características.

Apesar de ambos os sistemas serem baseados em Linux, o sistema operativo *VxWorks* é proprietário da *Wind River Systems* e é normalmente usado em sistemas críticos que podem ser tão diversos, como por exemplo os sistemas do *Boeing 787*, da Sonda espacial *Deep Impact*, os controladores dos robôs industriais *KUKA*, o sistema espacial *Mars Reconnaissance Orbiter*, Figura 10, entre muitos outros que também incluem sistemas não críticos [14].

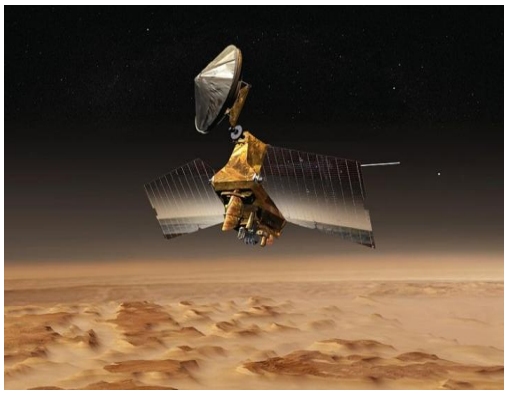


Figura 10 Nave espacial *Mars Reconnaissance Orbiter* [14]

O sistema operativo *Montavista*, é semelhante ao *VxWorks*, contudo, é normalmente usado em sistemas mais compactos e que não sejam sistemas críticos, tais como televisões, telemóveis, *routers*, sistemas de controlo de tráfego, câmaras de vídeo, entre outros pequenos dispositivos [15].

Qualquer um destes sistemas operativos é de tempo-real, no entanto, a diferença entre eles está centrada na segurança e na robustez do sistema, é esta diferença que permite que o *VxWorks* seja escolhido para sistemas críticos, ao invés do *Montavista*.

Esta escolha reflecte, igualmente os custos de cada um deles, sendo o *Montavista* um sistema operativo mais acessível. O desenvolvimento de aplicações sobre qualquer um destes sistemas operativos requer o uso de ferramentas específicas disponibilizadas pelos respectivos fabricantes. Para além destes dois sistemas (*VxWorks* e *Montavista*) existem muitos outros, alguns livres e outros pagos, no entanto, o grande problema é a flexibilidade de cada um deles em termos de *drivers* para periféricos e alguma programação específica que requer tempo de aprendizagem.

## 2.3 CONCLUSÕES

Neste capítulo foram apresentadas as principais características de cada sistema embebido e de alguns sistemas operativos.

A Tabela 1 indica as principais características dos sistemas embebidos analisados.

Tabela 1 Análise e comparação dos diversos sistemas embebidos descritos

| | Processador | $I^2C$ / SPI / RS232 / Ethernet | LCD/VGA | Alimentação | Sistema Operativo |
|---|---|---|---|---|---|
| **Toradex** | PXA270 520MHz | X / X / X / X | X / X | 7V – 24V | Windows CE |
| **AVR32** | AT32AP7000 150MHz | X / X / X / X | X / X | 9V – 15V | Linux |
| **BeagleBoard** | OMAP3530 600MHz | X / X / X / - | - / - | 5V | Linux |
| **Overo** | OMAP3530 600MHz | X / X / X / X | X / X | 5V | Linux |
| **Armadeus** | i.MX27 400MHz | X / X / X / X | X / - | 5V – 16V | Linux |

Desta pequena análise é possível concluir em relação aos sistemas embebidos que todos eles possuem características comuns, no entanto, certos sistemas embebidos possuem características, como a saída VGA, que são determinantes para a sua escolha final, além da saída VGA uma das características mais influentes é, sem dúvida, o sistema operativo.

Visto ser necessário um rápido desenvolvimento do sistema de aparafusamento automático, o sistema operativo escolhido foi o Windows CE.

A escolha do sistema operativo Windows CE implica que os sistemas embebidos AVR NGW100, BeagleBoard, Overo Gumstix e Armadeus sejam excluídos. Deste modo escolheu-se o sistema embebido da Toradex, de notar, que não é o único no mercado capaz de correr o Windows CE, mas é um dos sistemas mais baratos com este sistema operativo capaz de cumprir os objectivos propostos.

Assim o desenvolvimento do sistema automático de aparafusamento basear-se-á no sistema embebido da Toradex, com o sistema operativo Windows CE.

# CAPÍTULO 3

# 3 O SISTEMA EMBEBIDO DA TORADEX

No capítulo anterior foi efectuada uma análise sobre um grupo de sistemas embebidos capazes de cumprir com os objectivos traçados, chegando-se à conclusão que o sistema embebido a usar seria o sistema da Toradex com o sistema operativo Windows CE. Em relação aos sistemas automáticos de aparafusamento, observou-se que os modelos existentes não se adequam aos objectivos propostos.

Neste capítulo serão estudadas, de uma forma mais profunda, as características e capacidades do sistema embebido escolhido procedendo-se à análise da *framework* a usar para o ambiente gráfico, usando vários exemplos de programas teste para o demonstrar as suas funcionalidades e no final analisar-se-á os tempos de interrupção do sistema embebido.

## 3.1 SISTEMA DE CONTROLO

Nos sistemas tradicionais de controlo e monitorização de células industrias, Figura 11, é frequentemente usado um PLC (Controlador Lógico Programável) e um computador capaz de mostrar os dados relevantes sobre o processo.

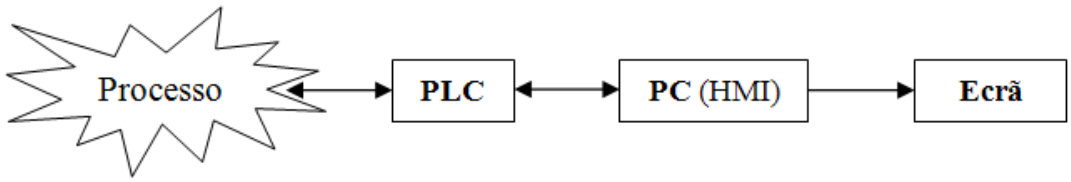



Figura 11 Implementação tradicional do controlo e monitorização de processos industriais

Esta solução possibilita o controlo e a monitorização da célula de trabalho, no entanto, requer conhecimentos de dois sistemas diferentes, o PLC e o PC.

Apesar de esta abordagem continuar a ser válida, começa a deixar de ser economicamente vantajosa, se for avaliado o custo do sistema PLC mais o PC relativamente ao custo de um sistema embebido standard.

Analisando o tempo de desenvolvimento e os conhecimentos necessários para a implementação do sistema PLC mais PC, é possível verificar que é necessário aprender duas linguagens de programação (uma de baixo e outra de alto nível) e implementar o protocolo de comunicação entre os dois sistemas. Desta forma, pode-se facilmente concluir que o custo do desenvolvimento de uma aplicação nestas duas plataformas

(PLC+PC) será mais elevado que o custo de desenvolvimento da mesma aplicação no sistema embebido, uma vez que neste último é apenas necessário utilizar uma linguagem de programação de alto nível e não se implementa qualquer protocolo de comunicação uma vez que a plataforma de hardware é única.

Com esta abordagem o sistema de controlo e monitorização fica mais simplificado, Figura 12.

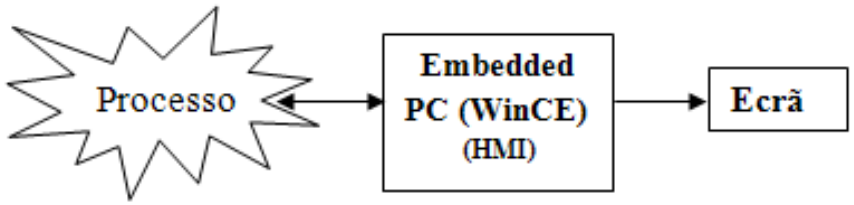



Figura 12 Implementação do controlo e monitorização de processos industriais através de um sistema embebido

## 3.2 REQUISITOS TEMPORAIS

Os requisitos temporais de uma célula de trabalho industrial estão muitas vezes associados com os tempos de segurança, ou seja, os tempos máximos para que um sistema industrial consiga produzir uma reacção nas suas saídas a um sinal de emergência nas suas entradas.

Este tempo de reacção é sem dúvida muito importante na segurança dos sistemas industriais, contudo, não é o único tempo a ter em conta. Segundo os objectivos do projecto onde é necessário monitorizar alguns dados sobre o estado actual do processo e por vezes, interagir em simultâneo com o utilizador, não é desejável que o tempo de resposta do sistema a essas interacções seja muito elevado.

Sendo assim, qual deverá ser o tempo máximo de reacção do sistema embebido para cumprir com os objectivos?

Tabela 2 Requisitos temporais em diversos sistemas automáticos [16]

| 5ms | 2.5ms | 1ms | 0.5ms | 200us | 100us | 50us | 20us |
|---|---|---|---|---|---|---|---|
| | | | Mobile Phones | | | | |
| Consumer Electronics | | | | | | | |
| | | | Telecom/Datacom | | | | |
| | | | | | | Automotive | |
| | | Industrial Automation | | | | | |

Segundo a Tabela 2 o tempo de reacção de um sistema industrial, neste caso um sistema de aparafusamento automático, deverá estar entre os 100us (microsegundos) e o 1ms (milisegundo). Mas este tempo é dependente do tipo de processo, ou seja, significa que a tabela anterior deve servir apenas como referência.

É necessário ter em atenção igualmente que, a maior parte das células de trabalho indústrias, cerca de 95%, funcionam de uma forma sequencial, ocorrendo muito raramente interrupções simultâneas [12].

Desta forma, só existe um limite temporal rígido para um sinal, o sinal de emergência, que neste caso será de 1ms, todos os outros sinais de entrada associados a sensores podem ser processados sem que lhes sejam associados um limite temporal rígido.

## 3.3 MÓDULO COLIBRI

O módulo Colibri da Toradex, Figura 13, é uma placa SO-DIMM de 200 pinos que possuí um processador PXA270 a 520MHz, 64Mb de SDRAM e 32Mb de flash.

Figura 13 Módulo Colibri PXA270 da Toradex [17]

Este módulo possui o processador XScale PXA270 capaz de correr o sistema operativo Windows CE, com interface para ligar periféricos como SPI, $I^2C$, UART, IrDA, GPIO (*General Proposal I/O*), Ethernet, SDCard, USB, PWM, LCD, Audio In/Out, entre outros.

Apesar de possuir vários meios de ligação para periféricos, estes não estão acessíveis da melhor forma, pois está placa foi concebida para trabalhar em conjunto com uma placa principal, esta sim com ligações que permitem ligar periféricos ao módulo Colibri.

## 3.4 PLACA ORCHID

Como referido anteriormente, a placa Orchid, Figura 14, tem como finalidade expor as ligações para conectar os periféricos, permitindo assim um fácil manuseamento do módulo Colibri.

Figura 14 Placa de expansão Orchid da Toradex [18]

Esta placa possui várias características que a tornam muito flexível e com uma grande capacidade de integração nos mais diversos dispositivos, devido ao grande conjunto de ligações.

Possui conectores para diversos protocolos, como por exemplo o USB, RS232, Ethernet, entre outros que se encontram nos pinos de expansão da placa. Além disso possui um leitor de SDCard, CompactFlash, infra-vermelhos, áudio in/out, VGA, GPIO e ligação para um LCD de 4.3 polegadas, Figura 15, com *touchscreen.*

Figura 15 LCD de 4.3 polegadas da Toradex, com ligação directa à placa Orchid [19]

Um ponto forte deste sistema embebido, é o facto de poder ser alimentado desde os 7V até aos 24V, o que significa que pode ser ligado directamente aos 24V da célula de trabalho, outro ponto forte é a ligação VGA existente na placa Orchid que permite ligar a maior parte dos monitores existentes, visto ser um dos requisitos da aplicação a utilização de um monitor.

Com todas estas características este sistema embebido é uma excelente plataforma para o estudo e desenvolvimento de aplicações usando sistemas embebidos comerciais.

## 3.5 PROGRAMAÇÃO

O sistema embebido apresentado é baseado em Windows CE o que faz com que o sistema consiga executar código em diversas linguagens de programação, entre elas C, C++ e linguagens .Net (C#, VB). Com a possibilidade de executar aplicações nas diversas linguagens apresentadas, é necessário realçar que a aplicação final requer uma ambiente gráfico amigável e controlo do sistema de aparafusamento, assim a linguagem C, apesar de ser a mais rápida esta não é a mais fácil de usar para a criação de ambientes visuais, excluindo-se então esta linguagem na implementação deste módulo de software.

O Visual Studio 2008 foi o ambiente de desenvolvimento escolhido sendo também o aconselhado pela Toradex para o desenvolvimento de aplicações nas linguagens acima referidas uma vez que possui facilidades para exportar as aplicações para os sistemas embebidos através do *ActiveSync* da Microsoft.

Possuindo as ferramentas necessárias ao desenvolvimento da aplicação para este projecto, a questão reside, qual a linguagem de programação a escolher?

### 3.5.1 CÓDIGO *MANAGED* E *UNMANAGED*

Para que seja possível responder com clareza a esta questão é necessário compreender a diferença entre código *managed* e *unmanaged.*

Código *managed* está associado a todas a linguagens de programação, neste caso, que usam a plataforma .Net, ou seja, C# e VB.Net. Este tipo de programação não produz uma aplicação com código máquina capaz de ser executado directamente no processador, pelo contrário, produz uma IL (*Intermediate Language*) que depois é executada num CLR (*Common Language Runtime*) e é este que é executado pelo processador. O que as linguagens do tipo *managed* de uma forma simplificada implementam é uma "camada" entre o processador e a aplicação em C# ou VB.Net, Figura 16.

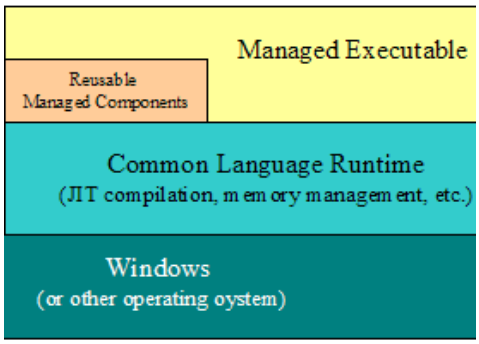



Figura 16 Estrutura de execução de código tipo *managed* [20]

Com esta implementação, os métodos da aplicação apenas são compilados quando necessários, é chamado de JIT (*Just In Time*), controlado pelo CLR. Os ganhos desta implementação são diversos, entre eles, a segurança, as threads e o controlo da memória são mais seguros e estáveis porque o código antes de ser executado pelo processador é analisado pelo CLR impedindo a ocorrência de possíveis problemas.

Existe também o código *unmanaged* que está associado às linguagens de programação anteriores à tecnologia .Net, por exemplo C e C++. A programação de aplicações com este tipo de código produz um código máquina que será executado directamente pelo processador sem que exista nenhuma "camada" de análise e controlo do código, apenas o sistema operativo, Figura 17.

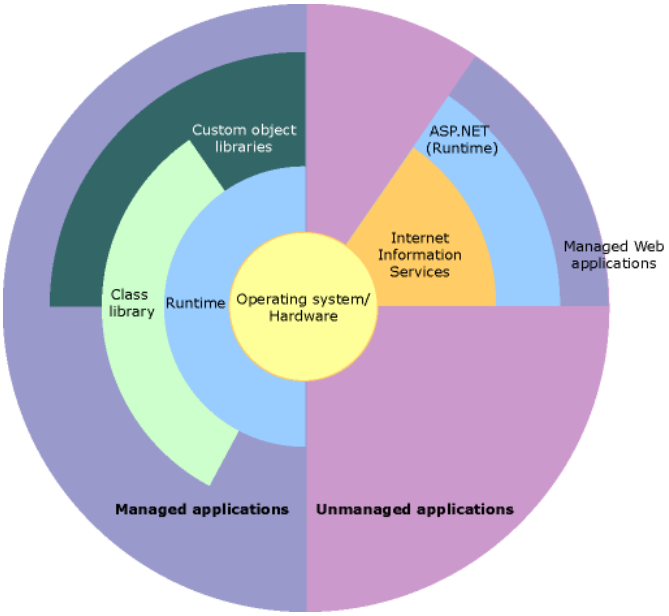



Figura 17 Diferença entre a estrutura de execução de código do tipo *managed* e *unmanaged* [21]

Ao contrário do código *manage*, o código *unmanaged* não possui segurança, estando a cargo do programador a implementação correcta das threads, controlo de memória, entre outros.

As vantagens do código *managed* sobre o código *unmanaged*, tendo em conta as linguagens apresentadas, são as seguintes: segurança, estabilidade, controlo da memória e implementação de ambientes visuais em Windows Forms.

Então as linguagens em código *managed* são as mais indicadas para esta aplicação?

Depende da aplicação, tendo em conta a aplicação deste projecto, as linguagens em código *managed* não são a melhor opção. Apesar de estas serem mais seguras e simples de implementar, possuem uma desvantagem importante, a velocidade de execução.

Aplicações em código *unmanaged* são mais rápidas do que aplicações em código *managed*, devido ao facto de as aplicações em código *managed* necessitarem de ser compiladas durante a execução (JIT) e analisadas pelo CLR antes de executar.

O teste seguinte, Figura 18, demonstra um caso onde foi usado a .Net Compact Framework com uma aplicação em C# onde é aplicado num pino de entrada uma onda quadrada com 10Hz de frequência e reproduzido o sinal noutro pino de saída.

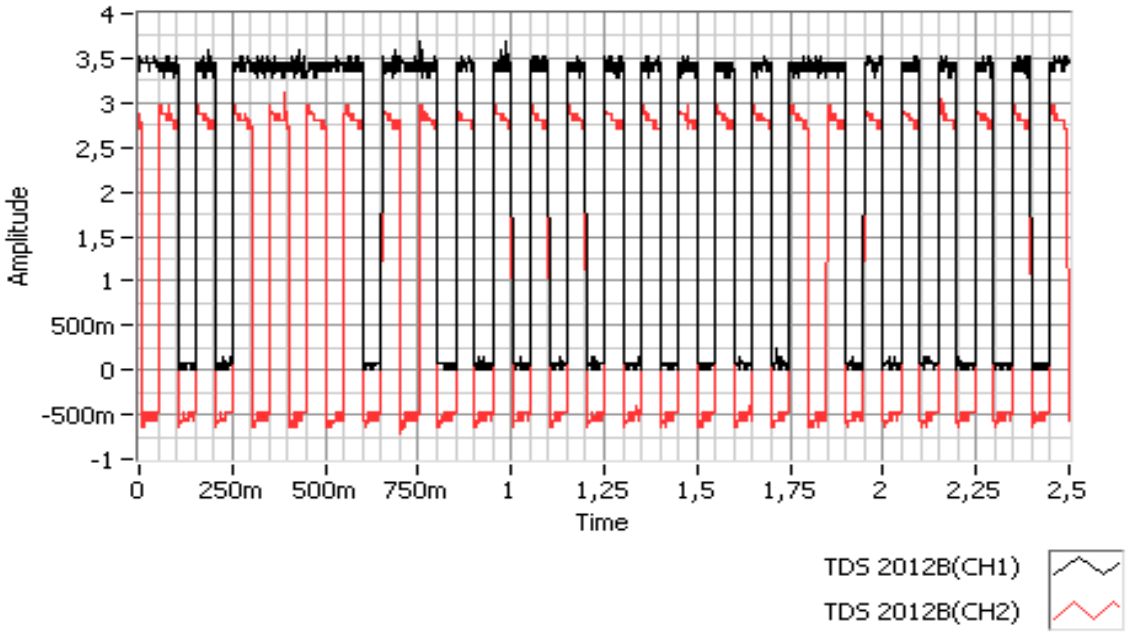



Figura 18 Reprodução de uma onda quadrada de 10Hz com uma aplicação do tipo *managed*

O sinal vermelho indica a onda quadrada de entrada e o sinal preto indica o sinal de saída. É possível observar que a onda de saída não é exactamente igual à onda de entrada, falhando algumas interrupções apenas com uma frequência de 10Hz.

Esta situação acontece porque o CLR possui várias ferramentas de controlo, tais como o GC (*Garbage Collector*) que é executado quando necessário, sem controlo por parte do programador, o que acontece neste caso, acabando a aplicação por perder algumas interrupções referentes ao pino de entrada do sinal.

Apesar desta situação é possível verificar que o sinal de saída está em fase com o sinal de entrada, ou seja, o desfasamento é pequeno como se pode verificar na Figura 19, é da ordem dos 320us (microsegundos).

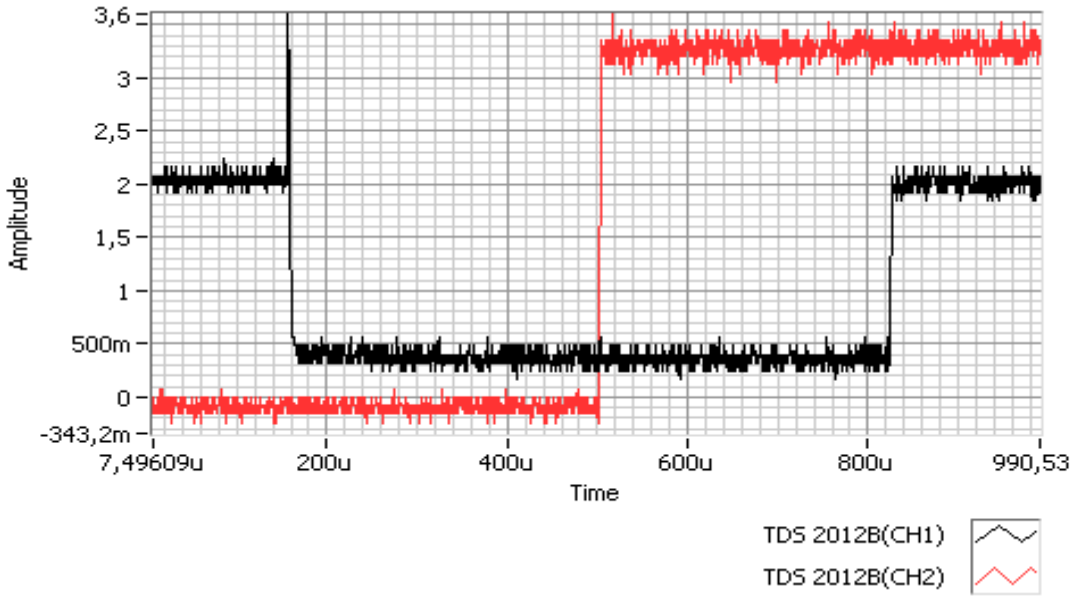



Figura 19 Desfasamento entre ondas com uma aplicação do tipo *managed*

Realizando os mesmos testes para código *unmanaged*, com uma aplicação em C++ a efectuar a repetição do sinal de entrada, é possível verificar uma melhoria substancial dos resultados obtidos, Figura 20.

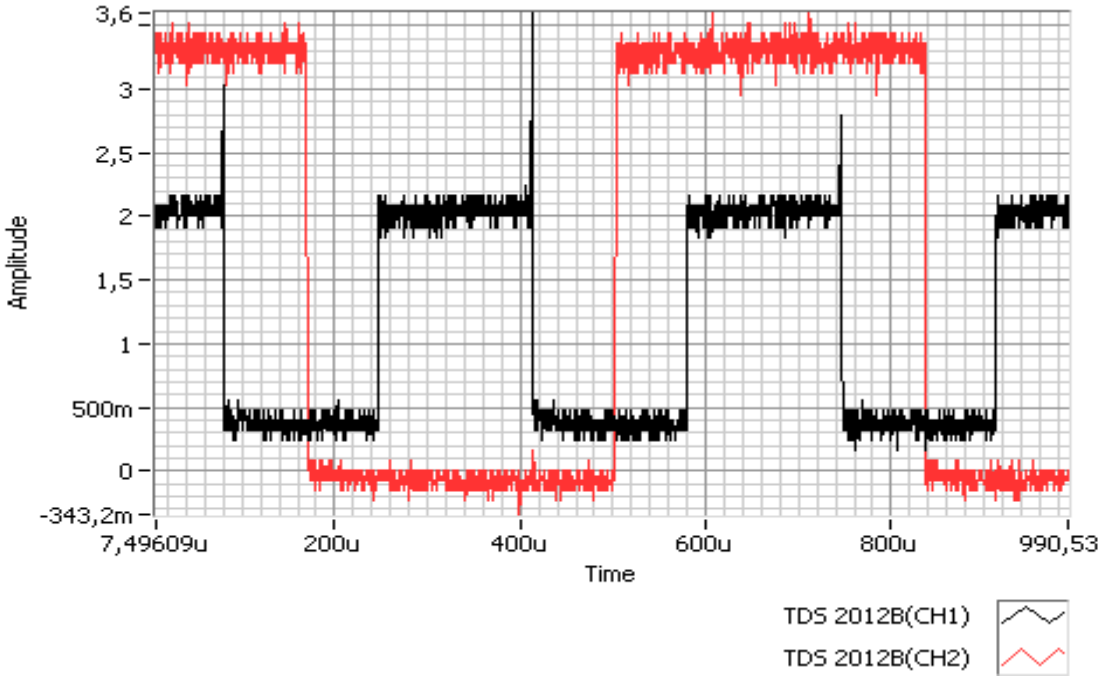



Figura 20 Reprodução de uma onda quadrada de 3KHz com aplicação do tipo *unmanaged*

Com este teste é possível verificar que o sinal de entrada tem uma frequência de aproximadamente 3KHz e o desfasamento do sinal de saída é da ordem dos 80us (micro segundos). Com estes testes, conclui-se que a linguagem que melhor cumpre com os requisitos da aplicação é do tipo *unmanaged*, ou seja, neste caso C++ porque possuí uma rápida resposta a sinais de entrada.

### 3.5.2 GUI

Com a escolha do C++ como linguagem de programação a usar para o desenvolvimento da aplicação de controlo do sistema de aparafusamento automático é necessário definir como se deve implementar o ambiente gráfico.

Para a implementação de um ambiente gráfico existem várias soluções, entre as quais, MFC, WxWidgets, Qt da Nokia, entre outras. Restringindo-se às três soluções anteriores e efectuando uma breve análise, qual será a mais adequada, Tabela 3?

Tabela 3 Análise entre as *frameworks* disponíveis para implementar um ambiente gráfico em WinCE

| | Windows CE | Usabilidade | Suporte | Licença | VS2008 |
|---|---|---|---|---|---|
| **MFC** | Sim | 3/5 | 5/5 | Proprietária | Incluído |
| **WxWidgets** | Limitado | 4/5 | 4/5 | Livre | Compilação |
| **Qt** | Sim | 4/5 | 4/5 | Proprietária ou Livre | Fornecido |

Analisando a tabela anterior, que efectua uma comparação entre algumas características genéricas de cada *framework* é possível verificar que a *framework* Qt é a mais equilibrada em todas as características. Analisando cada ponto temos que em termos de suporte para o Windows CE este está presente no MFC e no Qt, embora o WxWidgets também o suporte, este é limitado não apresentando tantas características como os outros.

Na usabilidade o Qt e o WxWidgets são mais simples de usar, devido ao facto de serem *frameworks* mais recentes e completas, no entanto o Qt poderá ser considerado mais amigável em algumas situações, como por exemplo, em termos do mecanismo de ligação entre sinais de envio/recepção de classes ou funções, o chamado *Signals&Slots*, que será descrito posteriormente.

No suporte a cada *framework* o MFC leva vantagem devido ao facto de ser mais antigo e existir muitos exemplos de implementação em diversos meios de suporte, como os livros, a internet, os fóruns e a própria biblioteca da Microsoft.

Relativamente às licenças, o MFC é o único proprietário, estando este incluído no Visual Studio 2008, contudo, o Qt possui uma versão *open source* e também uma licença profissional que inclui mais suporte para o cliente, a versão livre apenas dispõe dos fóruns de ajuda.

Como ultima característica em análise temos, a inclusão no Visual Studio 2008, em relação ao MFC este vem incluído sem problemas, o Qt possui *plugins* que

permitem a inclusão da sua *framework* no Visual Studio 2008, funcionando sem problemas críticos e com um *designer* para desenvolver o ambiente gráfico, Figura 21, o WxWidgets apenas é capaz de compilar no Visual Studio 2008, ficando a cargo de outra aplicação o design do ambiente gráfico.

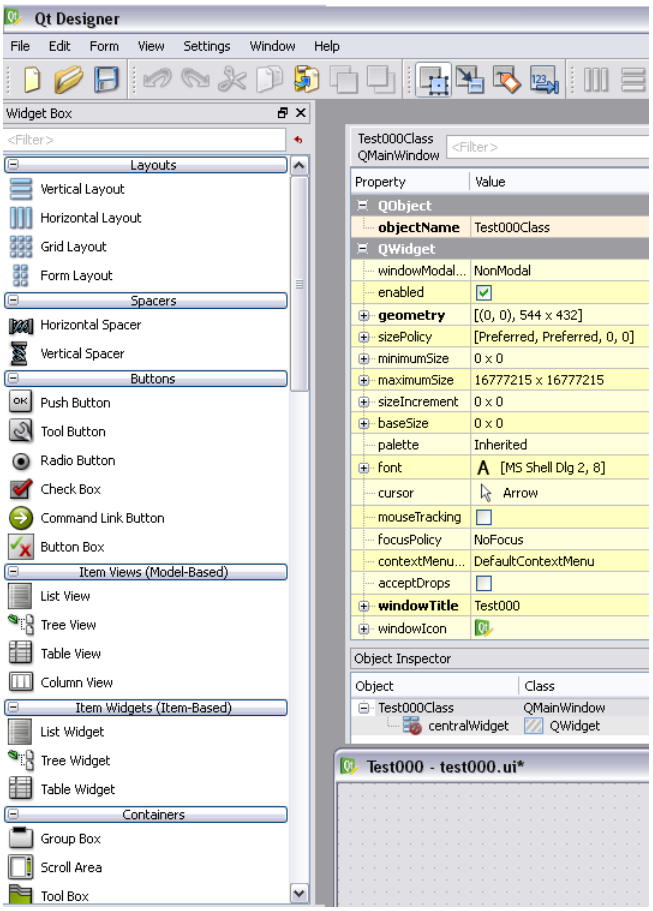



Figura 21 Ferramenta de desenvolvimento para a Framework Qt

Como resultado da análise das *frameworks* anteriores, a implementação da aplicação é efectuada em C++, optando-se pela *framework* Qt, por ser a mais equilibrada em todas as características, para desenvolver o ambiente gráfico para o sistema de aparafusamento automático.

## 3.6 HARDWARE DE EXPANSÃO

Dada a natureza do sistema a controlar, este necessita de trabalhar com os níveis de tensão de 0V a 24V disponíveis na célula de aparafusamento automático, contudo estes níveis de tensão não são compatíveis com os da placa Orchid da Toradex. As E/S (entradas e saídas) da placa de expansão funcionam com níveis de tensão entre 0V e 3.3V, o que requer uma placa de interface para que seja possível interagir com os vários componentes da célula de trabalho, para além deste problema, o número de E/S não é suficiente para interagir com todos os componentes.

Para solucionar esta questão foi necessário desenvolver placas de expansão que permitam ter as E/S isoladas necessárias para este projecto e com a flexibilidade necessária para projectos futuros, com este objectivo analisaram-se as interfaces existentes na placa de expansão Orchid, seleccionando-se a que melhor se adapta às placas de expansão de E/S a desenvolver.

Dos vários protocolos existentes, entre os quais o SPI, $I^2C$, RS232, USB e Ethernet é necessário analisar qual se adequa melhor à implementação das placas de expansão. Numa breve análise dos protocolos USB e Ethernet foi possível verificar que estes são bastante flexíveis e fiáveis, mas o custo e tempo de implementação é elevado para o tipo de placas desejadas, outro protocolo é o RS232 desenvolvido para ligações ponto a ponto, o que se torna pouco flexível no caso de existir mais do que uma placa de expansão de E/S.

Em última análise surgem os protocolos de $I^2C$ e SPI que permitem uma comunicação do tipo série com diversos dispositivos de expansão existentes no mercado para E/S, como por exemplo, da Microchip (www.microchip.com), tais como o MCP23S17 para SPI e o MCP23017 para $I^2C$.

### 3.6.1 PROTOCOLO SPI VS I²C

Os dois protocolos implementam uma comunicação do tipo série, como já referido anteriormente, mas o modo de funcionamento e os requisitos de cada uma deles é diferente como se poderá verificar nos próximos pontos.

## Protocolo SPI

O protocolo SPI (*Serial Peripheral Interface*) utiliza um barramento de comunicação síncrono de dados que opera em *full-duplex* (uma linha para transmissão e outra para recepção), com uma metodologia do tipo *master-slave* onde um dispositivo central controla os acessos e as velocidades de transmissão para os dispositivos *slave*.

Os dados são transmitidos de uma forma sequencial através de dois *shift registers*, um no dispositivo *master* e outro no *slave*, estando o *master* a controlar a velocidade de transmissão, Figura 22.

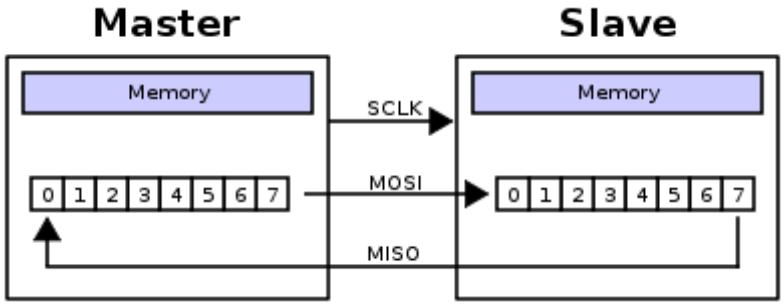



Figura 22 Implementação da comunicação ponto a ponto através do protocolo SPI [22]

Este tipo de implementação requer quatro ligações, uma de *clock* (SCLK), uma de selecção (SS) e duas de dados (MISO e MOSI), onde todas as ligações são unidireccionais, Figura 23.

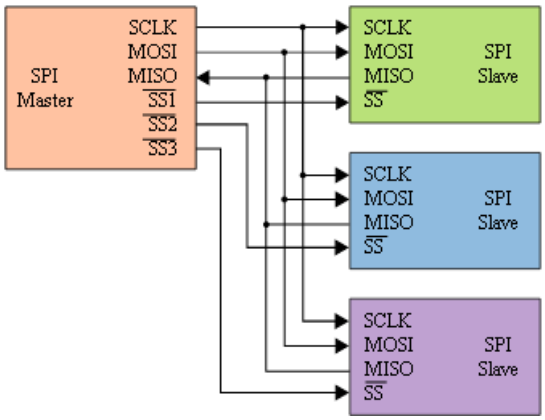



Figura 23 Implementação da comunicação através do protocolo SPI [22]

Como se pode verificar na figura anterior, o número de ligações aumenta conforme o número de dispositivos a controlar, o que se torna numa desvantagem deste protocolo. Este tipo de implementação só faz sentido se à partida se conhecer o número de dispositivos a controlar.

Para colmatar este problema o protocolo SPI tem uma segunda forma de implementação, onde os dados fluem de um dispositivo para outro sem necessidade de seleccionar qual é o destinatário, este tipo de implementação é chamado de *Daisy Chain*, Figura 24.

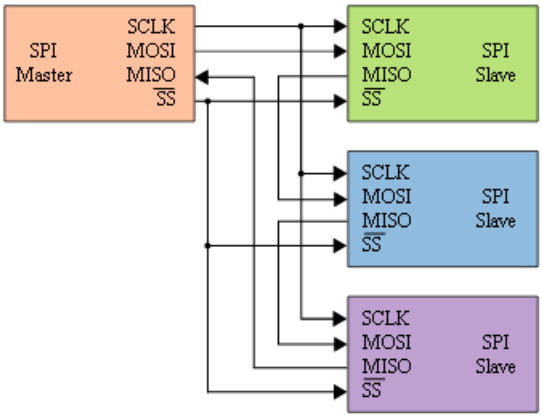



Figura 24 Implementação da comunicação por SPI em *Daisy Chain* [22]

Esta implementação permite manter o número de condutores do barramento constante, ou seja, é possível acrescentar dispositivos no barramento sem que este seja modificado, mas mesmo esta implementação possui uma desvantagem que se prende com a flexibilidade uma vez que a disposição dos vários dispositivos é importante.

Na primeira forma de implementação do SPI, o sinal de selecção (SS) seleccionava o dispositivo destino, na segunda implementação, como todos os dispositivos possuem *shift registers*, os dados passam entre eles até que o *master* decida terminar a comunicação, neste momento os dispositivos analisam os dados presentes no *shift register*, ou seja, o número de pulsos de *clock* indica a deslocação dos dados até ao dispositivo destino.

Se for introduzido um dispositivo no inicio do barramento, este irá influenciar o destino dos dados porque provoca uma atraso no barramento, necessitando agora de mais pulsos de *clock* para que os dados cheguem ao mesmo destinatário, daí a importância de informar o *master* da introdução de mais um dispositivo. Este problema torna o barramento igualmente pouco flexível.

O protocolo SPI tem como vantagens principais:

- Comunicação *full duplex*
- Taxa de transmissão pode chegar até 70MHz
- Grande flexibilidade na transmissão de dados, não limitado a 8 bits
- Implementação sem necessidade de hardware específico
- Necessidade de poucas ligações

E como desvantagens possui:

- Não possui qualquer controlo do fluxo de dados
- Não existe confirmação da recepção dos dados
- Apenas pode existir um *master*
- Não existe um formato standard para a sua implementação
- Distância de transmissão muito reduzida

## Protocolo I²C

Relativamente ao protocolo $I^2C$ (*Inter-Integrated Circuit*) este utiliza igualmente um barramento síncrono de dados mas operando em *half-duplex* (uma linha para transmissão e recepção), com uma metodologia *multi-master* onde pode existir mais do que um dispositivo *master* a controlar o envio dos dados no barramento para os dispositivos *slave*, em que estes também se podem tornar em dispositivos *master* para transmitirem, Figura 25.

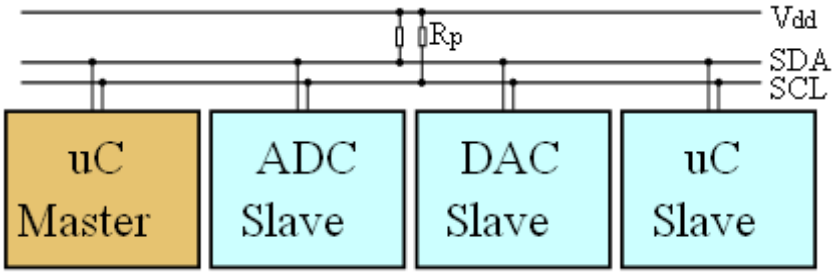



Figura 25 Comunicação através do barramento de $I^2C$ [23]

Este tipo de implementação requer apenas duas ligações, uma de *clock* (SCL) e uma de dados (SDA) onde ambas são bidireccionais. Além das ligações é necessária a colocação de uma resistência de *pull-up* em cada uma das linhas, porque o sistema usa ligações em colector aberto, ou seja, não define o valor de tensão nas linhas, usando-se as resistências ligadas a 5V ou 3.3V para o fazer.

Os dispositivos com comunicação $I^2C$ utilizam endereços para os identificar, ao contrário do SPI que possui um pino de selecção (SS), estes endereços tornam o protocolo $I^2C$ mais flexível, não ficando os dispositivos dependentes da sua posição no barramento nem de linhas de selecção.

Apesar de uma maior flexibilidade no barramento, os dispositivos com comunicação $I^2C$ possuem outros problemas, como por exemplo, baixa velocidade de transmissão de dados por se tratar de um barramento *half-duplex* e porque a velocidade do barramento é normalmente de 400kb/s, em certos casos específicos pode alcançar 1Mb/s e 3.4Mb/s.

Com velocidades de transmissão de 400kb/s o $I^2C$ não é um barramento muito rápido, é sim um barramento flexível, fácil de usar e implementar como se pode verificar pelas frames usadas, Figura 26.

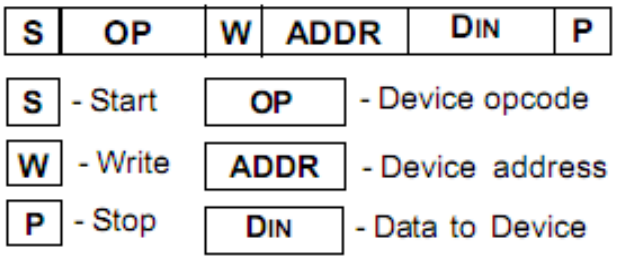



Figura 26 Frame de comunicação por $I^2C$ usada com o dispositivo MCP23017 [24]

Apesar de estas variarem dependendo do dispositivo com que se vai comunicar, existe uma base que é comum na maior parte dos dispositivos, o endereço do dispositivo seguido do endereço do registo, Figura 27.

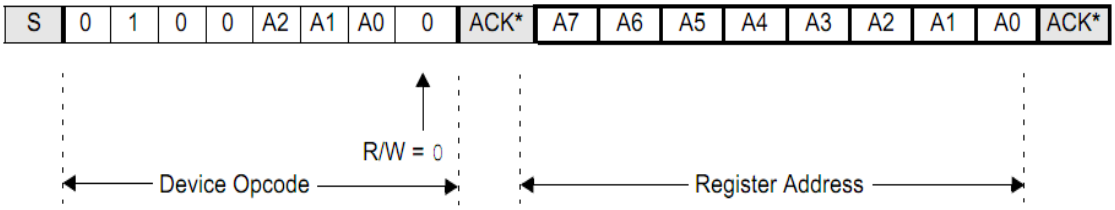



Figura 27 Sequência de inicial de cada frame de comunicação por $I^2C$ [24]

Normalmente os fabricantes de dispositivos remotos com capacidade de comunicação por $I^2C$, como a Microchip com o dispositivo MCP23017, definem uma parte do endereço que é fixa e a outra parte é definida pelo utilizador através de pinos externos no dispositivo, como se pode verificar na figura anterior no campo do *Device Opcode* onde o A2, A1 e A0 são os valores nos pinos externos.

O protocolo $I^2C$ tem como vantagens principais:

- Possui controlo de dados através do *acknowledgment*.
- Implementação sem necessidade de hardware específico.
- Necessidade de apenas duas ligações.
- Formato das frames com pequenas variações.
- Múltiplos *masters*.

E como desvantagens possui:

- Baixa velocidade de transmissão, inferior ao SPI.
- Limitado por alguns fabricantes no número de dispositivos por barramento.
- Distância de transmissão muito reduzida, não deve ser superior a 3 metros.

Tendo em conta as características dos dois protocolos analisados (SPI e $I^2C$), na tabela seguinte analisa-se qual o que melhor se adequa ao desenvolvimento das placas de expansão de E/S.

Tabela 4 Análise entre os barramentos SPI e $I^2C$

| | Velocidade | Alcance | Controlo de dados | Flexibilidade (Plug&Play) | Facilidade de implementação |
|---|---|---|---|---|---|
| **SPI** | 5/5 | 3/5 | 1/5 | 3/5 | 4/5 |
| **$I^2C$** | 3/5 | 3/5 | 3/5 | 5/5 | 5/5 |

Analisando a Tabela 4, que efectua uma comparação entre algumas características dos protocolos SPI e $I^2C$, verifica-se que um barramento por $I^2C$ possui mais vantagens sobre um barramento por SPI, se a velocidade não for um factor determinante e se o sistema não tiver necessidade de flexibilidade do tipo *Plug&Play* nem de controlo de dados.

Tendo em vista a implementação das placas de expansão para o controlo de uma célula de trabalho industrial, é de todo desejável que exista algum mecanismo de controlo de dados de forma a garantir a comunicação entre a unidade de controlo e as placas de expansão, neste caso o barramento por $I^2C$ é o único que satisfaz este requisito.

Com uma velocidade de 400Kb/s o $I^2C$ não é um barramento muito rápido, mas capaz de cumprir os requisitos temporais desejados (1ms), possuindo controlo de dados e maior flexibilidade para acrescentar placas de expansão no barramento.

### 3.6.2 I²C EXPANSÃO DAS SAÍDAS

A placa de expansão de saídas desenvolvida, Figura 28, é capaz de comunicar por $I^2C$, possuindo dezasseis saídas independentes opto-isoladas, cada uma capaz de suportar uma corrente máxima de 150mA.

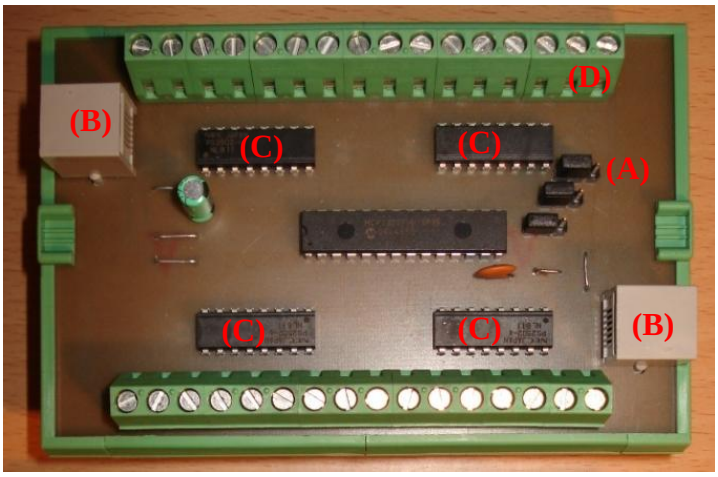



Figura 28 Placa de expansão de saídas por $I^2C$

Cada placa de expansão possuí três *jumpers* (A) para seleccionar o endereço a usar e dois conectores RJ11 (B) para o barramento de $I^2C$, deste modo é possível expandir o barramento com mais placas de expansão num máximo de oito por barramento, quatro opto-acopladores (C) para isolar as saídas e conectores duplos e triplos (D) para interface.

Para a comunicação por $I^2C$ com a placa de expansão foi usado o integrado MCP23017 da Microchip, que permite a expansão de E/S por $I^2C$.

Este é o componente central das placas de expansão, permitindo que cada pino deste seja configurado como entrada ou saída, criando a possibilidade de desenvolver uma placa de expansão de entradas que se discutirá posteriormente.

Para além deste, existem os integrados que efectuam o isolamento das E/S do MCP23017, visto que estas funcionam com tensões entre 0V e 5V e a célula entre tensões de 0V e 24V. Estes são os PS2502-4 do fabricante NEC e tem saídas a transístor em configuração *Darlington*, Figura 29, esta configuração permite uma corrente mais elevada entre o colector e o emissor.

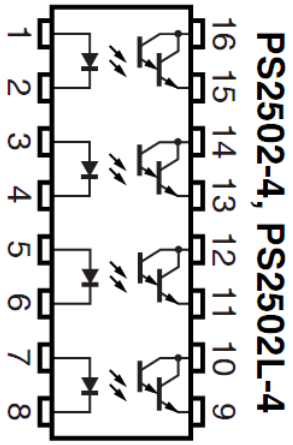



Figura 29 Opto isolador PS2502-4 da NEC [25]

Desta forma, é possível controlar dispositivos cujas tensões de funcionamento variem entre 0V e 24V do lado do transístor através da ligação visível na Figura 30:

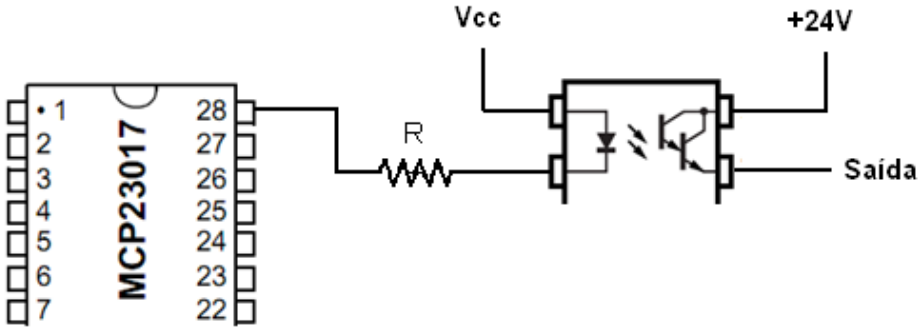



Figura 30 Ligação entre o IC de expansão por $I^2C$ e os opto isoladores

Nesta configuração o led emissor que permitir que o transístor conduza é activado quando a saída do MCP23017 tiver o valor lógico 0 (zero), ou seja, quando a tensão no pino for 0V, desta forma o transístor vai conduzir colocando 24V na saída.

Para a selecção do endereço para cada placa de expansão é disponibilizado, através de três *jumpers*, a possibilidade de configurar três bits do endereço do protocolo $I^2C$, Figura 31.

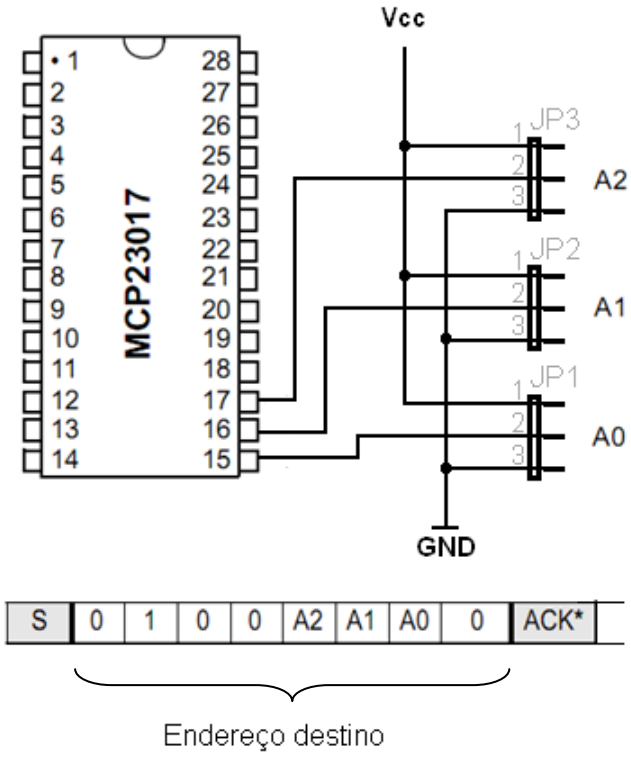



Figura 31 Selecção do endereço de cada placa de expansão

Assim, é possível colocar no máximo $2^{3\ bits} = 8$ placas de expansão no barramento através da interligação dos conectores RJ11 de cada placa, que transportam os seguintes sinais: 5V de alimentação para as placas, o GND (*Ground*), o sinal de dados (SDA) e o sinal de *clock* (SCL) do barramento $I^2C$.

Resumindo, esta placa possui os componentes descritos na Tabela 5, o circuito final desta está no Anexo II.

Tabela 5 Lista de componentes da placa de expansão de saídas por $I^2C$

| Componente | Quantidade | Referência |
|---|---|---|
| **MCP23017** | 1 | IC3 |
| **Conector RJ11** | 2 | J1, J2 |
| **Opto-acoplador NEC PS2502** | 4 | IC1, IC2, IC4, IC5 |
| **Condensador 10uF** | 1 | C1 |
| **Condensador 100nF** | 1 | C2 |
| ***Jumpers*** | 3 | JP1, JP2, JP3 |
| **Resistência 1KΩ SMD** | 16 | R1 a R16 |
| **Conectores de ligação com 2 saídas** | 16 | X1 a X16 |

### 3.6.3 I²C EXPANSÃO DAS ENTRADAS

A placa de expansão de entradas desenvolvida, Figura 32, capaz de comunicar por $I^2C$ possui dezasseis entradas independentes opto-isoladas, com cada entrada capaz de suportar tensões máximas de 26V.

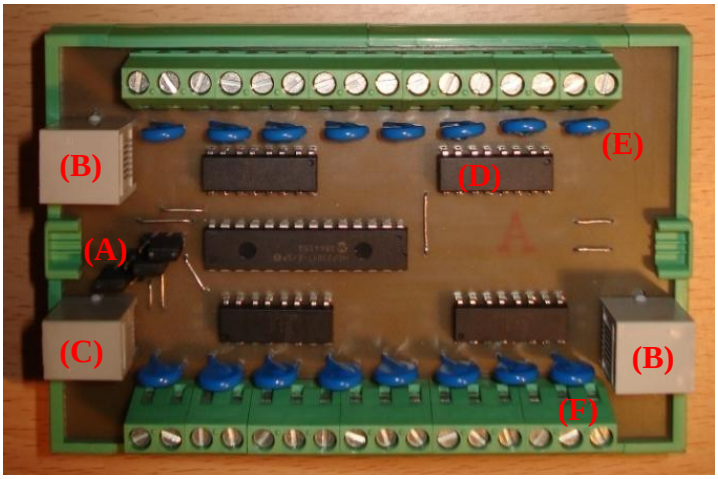



Figura 32 Placa de expansão de entradas por $I^2C$

Cada placa de expansão possuí três *jumpers* (A) para seleccionar o endereço a usar, dois conectores RJ11 (B) para o barramento de $I^2C$, um conector RJ9 (C) com as interrupções geradas pelo MCP23017 que serão transmitidas à unidade de controlo, quatro opto-isoladores (D) para isolar as entradas, dezasseis varistores (E) de protecção para 26V e conectores duplos e triplos (F) para interface.

Como referido anteriormente para a comunicação por $I^2C$ com a placa de expansão foi usado o integrado MCP23017 da Microchip, que permite a expansão de E/S por $I^2C$.

Este é o componente central das placas de expansão, permitindo que neste caso cada pino seja configurado como entrada.

Para além deste, existem os integrados que efectuam o isolamento das E/S do MCP23017, visto que estas funcionam com tensões entre 0V e 5V e a célula entre tensões de 0V e 24V. Estes são os KP1040 do fabricante Cosmo com saídas a transístor, Figura 33.

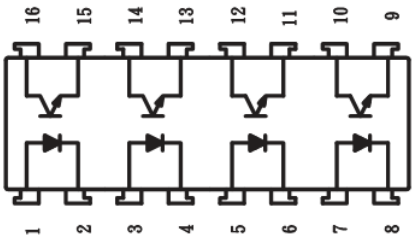



Figura 33 Opto isolador KP1040 da Cosmo [26]

Desta forma, é possível receber sinais de dispositivos (sensores, controladores) cujas tensões de saída variem entre 0V e 24V do lado do transístor através da ligação visível na imagem seguinte:

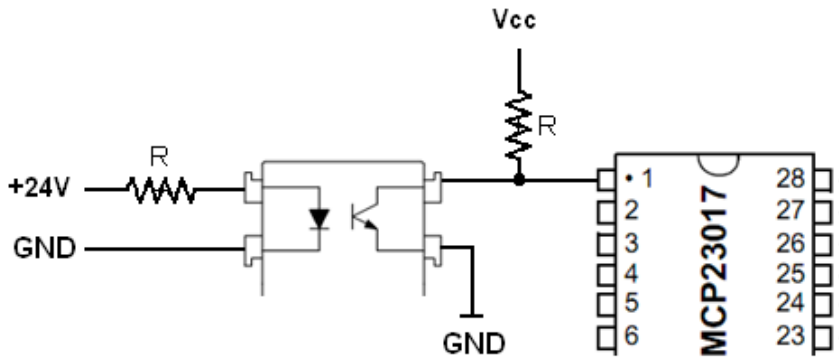



Figura 34 Ligação entre o IC de expansão por $I^2C$ e o opto isolador de entrada

Nesta configuração o led emissor que permitir que o transístor conduza é activado quando alimentado com 24V, no caso da aplicação para a célula de trabalho, assim o transístor conduz para GND colocando do valor lógico 0 no pino de entrada do MCP23017, caso contrário a resistência de *pull-up* coloca o valor lógico 1.

A selecção do endereço é efectuada da mesma forma, como foi anteriormente explicado na placa de expansão de saídas, assim é possível colocar no máximo $2^{3\ bits} = 8$ placas de expansão no barramento. Contudo, com o limite máximo de 8 placas de expansão por barramento de $I^2C$ não é possível colocar 8 placas de entrada e de saída, mas sim um total de 8 placas dos dois tipos, por exemplo, 5 placas de expansão de entrada e 3 de saída. Resumindo, a placa de expansão de entradas é composta pelos componentes descritos na Tabela 6, o circuito final desta está no Anexo III.

Tabela 6 Lista de componentes da placa de expansão de entradas por $I^2C$

| Componente | Quantidade | Referência |
|---|---|---|
| **MCP23017** | 1 | IC3 |
| **Conector RJ11** | 2 | J1, J2 |
| **Conector RJ9** | 2 | J3 |
| **Opto-acoplador Cosmo KP1040** | 4 | IC1, IC2, IC4, IC5 |
| **Condensador 10uF** | 1 | C1 |
| **Condensador 100nF** | 1 | C2 |
| ***Jumpers*** | 3 | JP1, JP2, JP3 |
| **Resistência 1KΩ SMD** | 16 | – |
| **Resistência 390Ω SMD** | 16 | – |
| **Varistor 07K20** | 16 | 07K20 |
| **Conectores de ligação com 2 saídas** | 16 | X1 a X16 |

## 3.7 SOFTWARE DE TESTES

Após o desenvolvimento do hardware necessário para o interface entre a célula de aparafusamento automático, realizou-se numa primeira fase o desenvolvimento de aplicações que permitam interagir a plataforma de desenvolvimento Orchid da Toradex com as placas de expansão.

O âmbito destas aplicações de teste é a análise/estudo da melhor forma de implementar o código para a comunicação por $I^2C$ na plataforma Orchid, para efectuar a interacção com as placas de expansão de forma a atingir os requisitos necessários para a implementação do sistema de aparafusamento automático.

Para o desenvolvimento de tais aplicações usou-se a linguagem C++ no ambiente de desenvolvimento da Microsoft, o Visual Studio 2008 juntamente com a framework Qt da Nokia, como referido anteriormente.

Assim, para iniciar o desenvolvimento sobre o sistema embebido é preciso ligar a placa *Orchid* a um PC instalando as bibliotecas necessárias e as aplicações fornecidas pela *Toradex*, instalando também o *ActiveSync* da Microsoft.

Após a configuração inicial é possível ver o ambiente de trabalho do sistema embebido da *Toradex* num PC com o *ActiveSync* instalado, através de uma ligação USB com a placa e com a aplicação *RemoteDisplay* da *Toradex*, Figura 35.

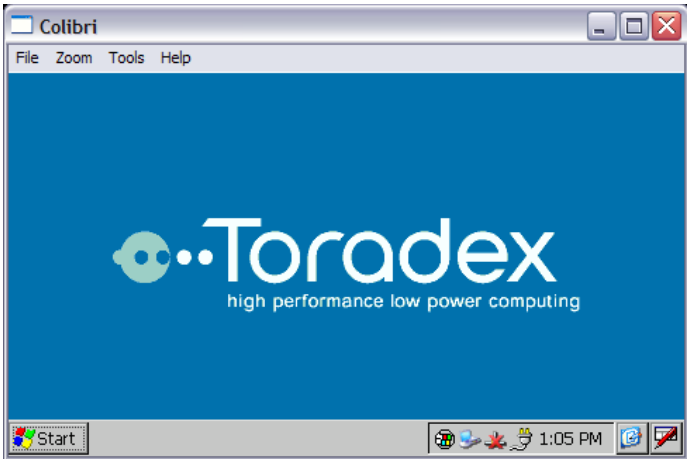



Figura 35 Ambiente remoto do sistema embebido da Toradex

Neste ponto é necessário configurar correctamente o Visual Studio 2008 para funcionar com o sistema embebido e com a Framework Qt, no Anexo I são descritos os passos necessários para uma configuração correcta das livrarias da Framework Qt.

### 3.7.1 GPIO CONTROL (QTGPIO)

Para que seja possível interagir com as placas de expansão é necessário configurar os pinos externos da placa Orchid, Figura 36, configurando-os com entradas ou saídas.

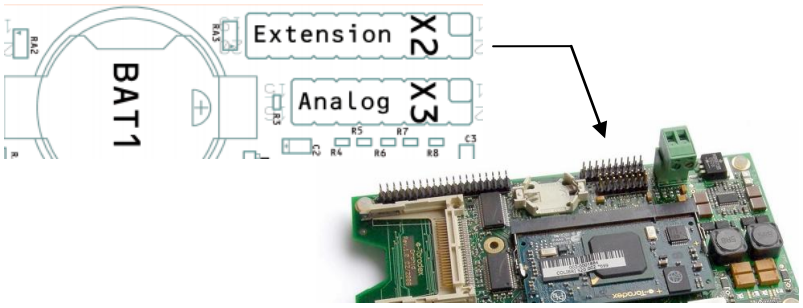



Figura 36 Localização dos pinos de entrada e saída da placa Orchid

Para além de possuir os pinos necessários para comunicar por $I^2C$ também possui pinos de E/S genéricos e para SPI, Figura 37, em que os pinos que suportam o $I^2C$ e o SPI são multiplexados, possuindo mais do que uma função, neste caso E/S genéricas.

| Pin Nr. | SODIMM PIN | Colibri 270 Vx.xx Signal (Function) | IO Type | Voltage | Pullup/Pulldown |
|---|---|---|---|---|---|
| 1 | SODIMM Pin 26 | nRESET_IN | I | +3V3 | Pullup |
| 2 | GND | GND | PWR | | |
| 3 | SODIMM Pin 87 | nRESET_OUT | O | +3V3 | |
| 4 | +3V3 | +3V3 | PWR | | |
| 5 | SODIMM Pin 194 | I2C_DATA (GPIO 118) | IO | +3V3 | Pullup (connected to on Board RTC) |
| 6 | SODIMM Pin 196 | I2C_CLK (GPIO 117) | IO | +3V3 | Pullup (connected to on Board RTC) |
| 7 | SODIMM Pin 88 | SSPCLK (GPIO 23) | IO | +3V3 | |
| 8 | SODIMM Pin 86 | SSPFRM (GPIO 24) | IO | +3V3 | |
| 9 | SODIMM Pin 92 | SSPTXD (GPIO 25) | IO | +3V3 | |
| 10 | SODIMM Pin 90 | SSPRXD (GPIO 26) | IO | +3V3 | |
| 11 | +5V | +5V | PWR | | |
| 12 | SODIMM Pin 135 | GPIO 35 | IO | +3V3 | |
| 13 | SODIMM Pin 133 | GPIO 37 | IO | +3V3 | |
| 14 | SODIMM Pin 127 | GPIO 36 | IO | +3V3 | |
| 15 | SODIMM Pin 107 | GPIO 79 | IO | +3V3 | |
| 16 | SODIMM Pin 105 | GPIO 15 | IO | +3V3 | |
| 17 | SODIMM Pin 106 | GPIO 80 | IO | +3V3 | |
| 18 | SODIMM Pin 73 | GPIO 52 | IO | +3V3 | Pullup (100k) |
| 19 | SODIMM Pin 55 | GPIO 19 | IO | +3V3 | Pullup (100k) |
| 20 | GND | GND | PWR | | |

Figura 37 Descrição dos pinos disponíveis na placa Orchid [27]

Para se conseguir modificar a função de cada pino de expansão, direcção e valor lógico é necessário conhecer os registos associados presentes no *datasheet* do processador PXA270. Após a consulta deste é possível identificar os diversos tipos de registos associados à manipulação de cada pino de E/S, Figura 38.

| Physical Address | Name | Description |
|---|---|---|
| 0x40E0_ 0000 | GPLR0 | GPIO Pin-Level register GPIO<31:0> |
| 0x40E0_ 0004 | GPLR1 | GPIO Pin-Level register GPIO<63:32> |
| 0x40E0_ 0008 | GPLR2 | GPIO Pin-Level register GPIO<95:64> |
| 0x40E0_ 000C | GPDR0 | GPIO Pin Direction register GPIO<31:0> |
| 0x40E0_ 0010 | GPDR1 | GPIO Pin Direction register GPIO<63:32> |
| 0x40E0_ 0014 | GPDR2 | GPIO Pin Direction register GPIO<95:64> |
| 0x40E0_ 0018 | GPSR0 | GPIO Pin Output Set register GPIO<31:0> |
| 0x40E0_ 001C | GPSR1 | GPIO Pin Output Set register GPIO<63:32> |
| 0x40E0_ 0020 | GPSR2 | GPIO Pin Output Set register GPIO<95:64> |
| 0x40E0_ 0024 | GPCR0 | GPIO Pin Output Clear register GPIO<31:0> |
| 0x40E0_ 0028 | GPCR1 | GPIO Pin Output Clear register GPIO <63:32> |
| 0x40E0_ 002C | GPCR2 | GPIO pin Output Clear register GPIO <95:64> |
| 0x40E0_ 0030 | GRER0 | GPIO Rising-Edge Detect Enable register GPIO<31:0> |

Figura 38 Registos associados a cada pino de E/S [27]

Na figura anterior é possível verificar os diversos registos responsáveis pela direcção, valor, função de cada E/S. Nos registos é igualmente indicado o endereço de cada um, o qual servirá para que as aplicações consigam modificar os valores, modificando assim o funcionamento de cada E/S, Figura 39.

| Physical Address | Name | Description |
|---|---|---|
| 0x40E0_ 0034 | GRER1 | GPIO Rising-Edge Detect Enable register GPIO<63:32> |
| 0x40E0_ 0038 | GRER2 | GPIO Rising-Edge Detect Enable register GPIO<95:64> |
| 0x40E0_ 003C | GFER0 | GPIO Falling-Edge Detect Enable register GPIO<31:0> |
| 0x40E0_ 0040 | GFER1 | GPIO Falling-Edge Detect Enable register GPIO<63:32> |
| 0x40E0_ 0044 | GFER2 | GPIO Falling-Edge Detect Enable register GPIO<95:64> |
| 0x40E0_ 0048 | GEDR0 | GPIO Edge Detect Status register GPIO<31:0> |
| 0x40E0_ 004C | GEDR1 | GPIO Edge Detect Status register GPIO<63:32> |
| 0x40E0_ 0050 | GEDR2 | GPIO Edge Detect Status register GPIO<95:64> |
| 0x40E0_ 0054 | GAFR0_L | GPIO Alternate Function register GPIO<15:0> |
| 0x40E0_ 0058 | GAFR0_U | GPIO Alternate Function register GPIO<31:16> |
| 0x40E0_ 005C | GAFR1_L | GPIO Alternate Function register GPIO<47:32> |
| 0x40E0_ 0060 | GAFR1_U | GPIO Alternate Function register GPIO<63:48> |
| 0x40E0_ 0064 | GAFR2_L | GPIO Alternate Function register GPIO<79:64> |
| 0x40E0_ 0068 | GAFR2_U | GPIO Alternate Function register GPIO <95:80> |
| 0x40E0_ 006C | GAFR3_L | GPIO Alternate Function register GPIO<111:96> |
| 0x40E0_ 0070 | GAFR3_U | GPIO Alternate Function register GPIO<120:112> |

Figura 39 Registo associado à função actual do pino de E/S [27]

Todos os registos são de 32 bits e possuem o seguinte formato, Figura 40:

Physical Address 0x40E0_000C — GPDR0 — GPIO Controller

User Settings

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PD31 | PD30 | PD29 | PD28 | PD27 | PD26 | PD25 | PD24 | PD23 | PD22 | PD21 | PD20 | PD19 | PD18 | PD17 | PD16 | PD15 | PD14 | PD13 | PD12 | PD11 | PD10 | PD9 | reserved | | | | PD4 | PD3 | reserved | PD1 | PD0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Access | Name | Description |
|---|---|---|---|
| <31:9> | R/W | PD*x* | GPIO Pin Direction 'x' (where x = 9 to 31)<br>This field configures the direction of each GPIO.<br>0 = Pin configured as an input.<br>1 = Pin configured as an output. |
| <8:5> | — | — | reserved |
| <4:3> | R/W | PD*x* | GPIO Pin Direction 'x' (where x = 3 to 4)<br>This field configures the direction of each GPIO.<br>0 = Pin configured as an input.<br>1 = Pin configured as an output. |
| <2> | — | — | reserved |
| <1:0> | R/W | PD*x* | GPIO Pin Direction 'x' (where x = 0 to 1)<br>This field configures the direction of each GPIO.<br>0 = Pin configured as an input.<br>1 = Pin configured as an output. |

Figura 40 Descrição do registo de direcção de um pino de E/S [27]

No caso da figura anterior o registo GPDR0 indica a direcção de cada E/S desde o pino 0 até 31, onde cada bit da palavra indica se este está configurado com entrada (0) ou como saída (1). Para cada registo é indicado o significado de cada bit e as combinações de bits, no caso das funções de cada E/S, necessárias para seleccionar o funcionamento desejado.

Após a compreensão do funcionamento dos registos mais importantes para as E/S é necessário conseguir modificá-los a partir das aplicações a criar em C++. Para isso a Toradex fornece algumas bibliotecas para facilitar o acesso e controlo do processador (*CoProcLib.h*, *IntLib.h* e *MapRegLib.h*) onde a biblioteca *CoProcLib.h* permite obter e configurar algumas informações sobre o sistema embebido, entre as quais a frequência de funcionamento, a versão do processador e os modos de energia deste. A biblioteca *IntLib.h* fornece algumas funções que permitem configurar a ISR e IST para cada E/S, a biblioteca *MapRegLib.h* permite o controlo dos registos do processador, não só das E/S mas de todos desde que se saiba os endereços e a organização destes.

Analisando os exemplos e as bibliotecas anteriores retêm-se que as funções mais importantes são: ***void* MapRegister(DWORD pa);*** (1)

***void UnMapRegister(volatile void* pRegs);*** (2)

A primeira função (1) permite alocar uma zona de memória do processo actual, em que esta mapeia os registos do processador PXA270 que foram passados como argumento através do seu endereço, retornando um ponteiro. Este ponteiro contém o endereço da memória que mapeia os registos, bastando agora associar uma estrutura de dados para que se possa modificar os registo do processador, esta estrutura deverá ter o mesmo tamanho da memória alocada e com palavras do mesmo tamanho, assim a estrutura de dados para as E/S genéricas seria a seguinte:

```
typedef volatile struct GPIODATA
    {           DWORD gplr0;        // Level Reg. Bank 0
                DWORD gplr1;        // Level Reg. Bank 1
                DWORD gplr2;        // Level Reg. Bank 2
                DWORD gpdr0;        // Direction Reg. Bank 0
                DWORD gpdr1;        // Direction Reg. Bank 1
                DWORD gpdr2;        // Direction Reg. Bank 2
                DWORD gpsr0;        // Output Set Reg. Bank 0
                DWORD gpsr1;        // Output Set Reg. Bank 1
                DWORD gpsr2;        // Output Set Reg. Bank 2
                DWORD gpcr0;        // Output Clr Reg. Bank 0
                DWORD gpcr1;        // Output Clr Reg. Bank 1
                DWORD gpcr2;        // Output Clr Reg. Bank 2
                DWORD grer0;        // Ris. Edge Detect Enable Reg. Bank 0
                DWORD grer1;        // Ris. Edge Detect Enable Reg. Bank 1
                DWORD grer2;        // Ris. Edge Detect Enable Reg. Bank 2
                DWORD gfer0;        // Fal. Edge Detect Enable Reg. Bank 0
                DWORD gfer1;        // Fal. Edge Detect Enable Reg. Bank 1
                DWORD gfer2;/       / Fal. Edge Detect Enable Reg. Bank 2
                DWORD gedr0;        // Edge Detect Status Reg. Bank 0
                DWORD gedr1;        // Edge Detect Status Reg. Bank 1
                DWORD gedr2;        // Edge Detect Status Reg. Bank 2
                DWORD gafr0_l;      // Alt. Function Select Reg.[  0:15 ]
                DWORD gafr0_u;      // Alt. Function Select Reg.[ 16:31 ]
                DWORD gafr1_l;      // Alt. Function Select Reg.[ 32:47 ]
                DWORD gafr1_u;      // Alt. Function Select Reg.[ 48:63 ]
                DWORD gafr2_l;      // Alt. Function Select Reg.[ 64:79 ]
                DWORD gafr2_u;      // Alt. Function Select Reg.[ 80:95 ]
                DWORD gafr3_l;      // Alt. Function Select Reg.[ 96:111]
                DWORD gafr3_u;      // Alt. Function Select Reg.[112:120]
                DWORD res1[35];
                DWORD gplr3;        // Level Detect Reg. Bank 3
                DWORD res2[2];
                DWORD gpdr3;        // Data Direction Reg. Bank 3
                DWORD res3[2];
                DWORD gpsr3;        // Pin Output Set Reg. Bank 3
                DWORD res4[2];
                DWORD gpcr3;        // Pin Output Clr Reg. Bank 3
                DWORD res5[2];
                DWORD grer3;        // Ris. Edge Detect Enable Reg. Bank 3
                DWORD res6[2];
                DWORD gfer3;        // Fal. Edge Detect Enable Reg. Bank 3
                DWORD res7[2];
                DWORD gedr3;        // Edge Detect Status Reg. Bank 3
    };
```

Com esta estrutura de dados e com o endereço base dos registos (0x40E00000) falta apenas chamar a função (1) e atribuir o ponteiro a uma variável do tipo "GPIODATA" e modificar as diversas variáveis desta para alterar o valor dos registos do processador.

No final é de todo conveniente libertar a memória para que esta não produza efeitos indesejados nos registos, através da segunda função (2) enunciada anteriormente.

Sendo possível agora aceder aos registos das E/S é possível criar uma aplicação capaz de controlar os diversos pinos de expansão presentes na placa Orchid, para isso usa-se o Visual Studio 2008 e a framework Qt para desenvolver a aplicação que permita visualizar o estado de cada pino e modificar a sua direcção. Esta aplicação foi especialmente dimensionada para poder ser visualizada e controlada através do ecrã táctil disponível, possui como base o controlo *WidgetIO* que é responsável pelo controlo de cada E/S, Figura 41.

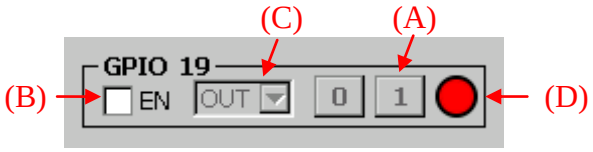



Figura 41 Controlo *WidgetIO*

Este controlo foi desenvolvido de forma a simplificar a implementação do controlo sobre as E/S disponíveis sendo constituído por diversos controlos, tais como o QPushButton (A), o QCheckBox (B), o QComboBox (C), o WidgetLED (D) e a classe GPIO para acesso aos registos, como referido anteriormente.

Como resultado da utilização deste controlo a aplicação final de controlo dos pinos de expansão ficou como na Figura 42:

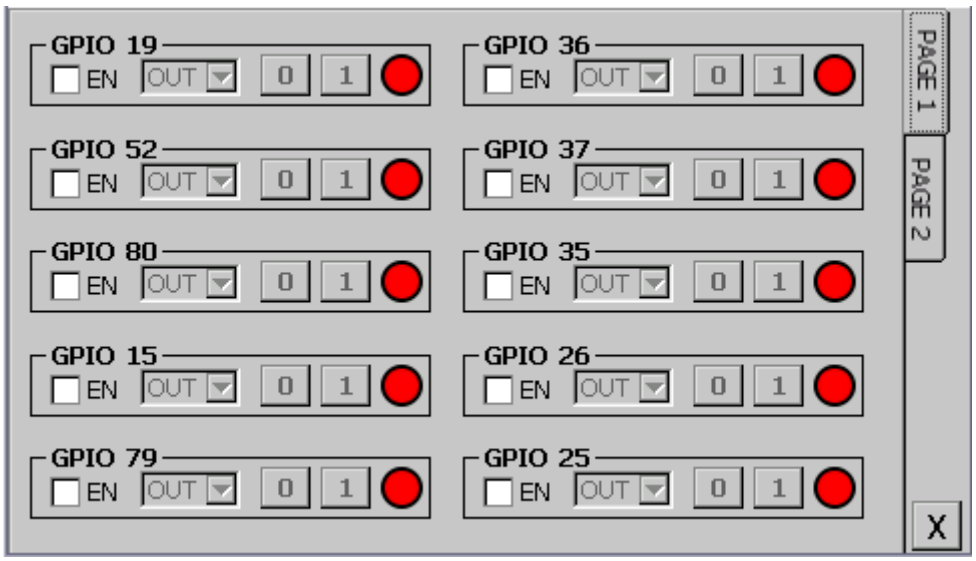



Figura 42 Aplicação em Qt para controlar os pinos da placa Orchid

Na figura anterior é possível também verificar que existem duas páginas para que seja possível expor todas as E/S da placa Orchid.

No diagrama seguinte, Figura 43, é descrito o funcionamento do controlo *WidgetIO*.

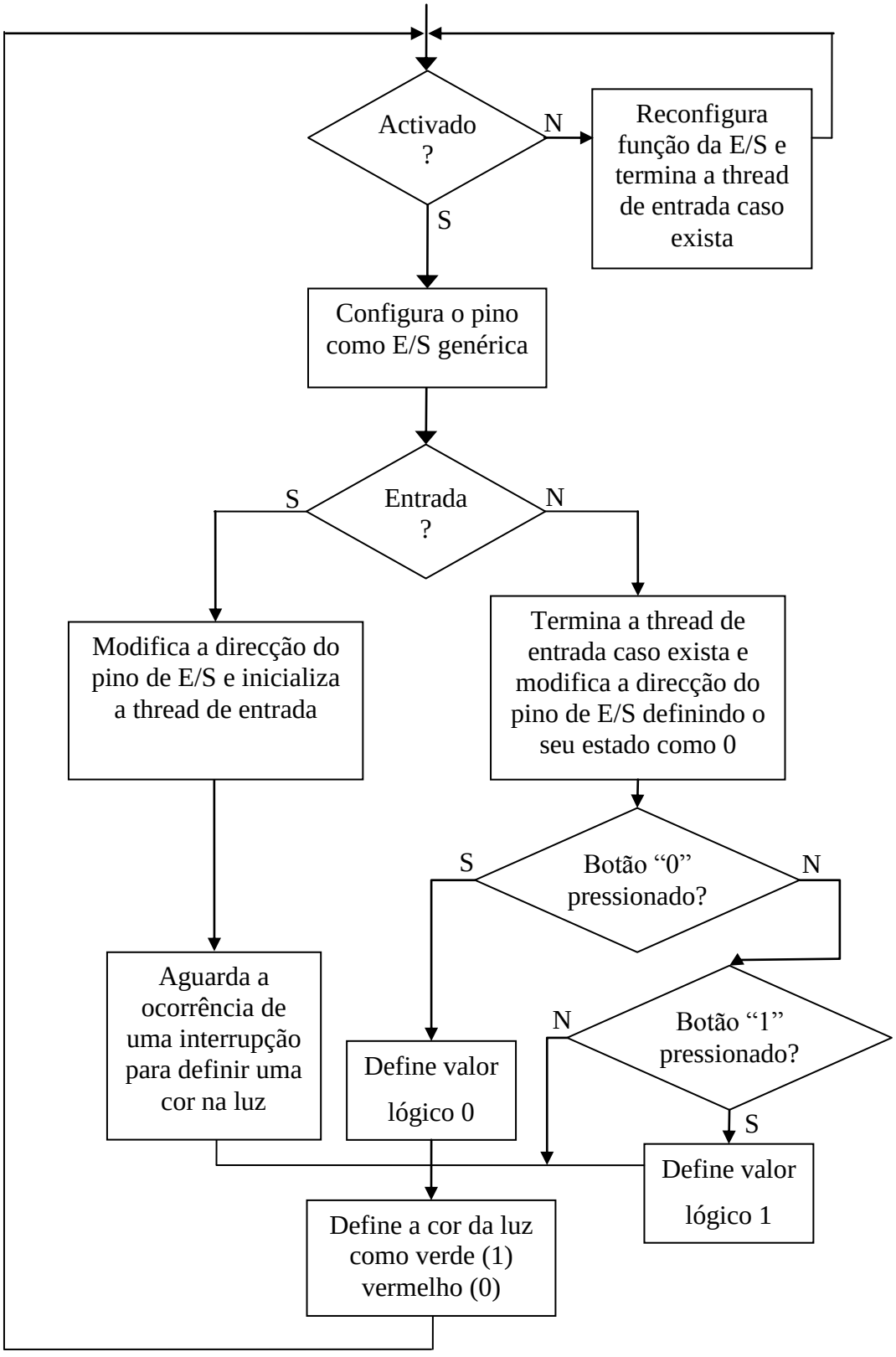



Figura 43 Diagrama de funcionamento do controlo *WidgetIO*

Com este controlo definido para todas as E/S é possível então alterar a direcção de cada pino, o valor deste e activar ou desactivar a função de E/S genérica associada ao pino, Figura 44, no caso de se desactivar o controlo este voltará a colocar a E/S na função anteriormente definida.

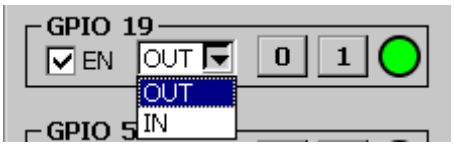



Figura 44 Definição do sentido do pino de E/S

Para além da activação e desactivação do controlo que permite modificar a função associada a cada E/S automaticamente, este pode ser configurado para funcionar como saída ou entrada, assim no caso de ser configurado como saída os botões "0" e "1" estarão activados para que o utilizador possa definir o valor, a luz verde apenas sinalizará o valor definido, "1" para verde e "0" para vermelho.

No caso de este estar configurado como entrada o funcionamento do controlo altera-se, passando a executar uma thread para verificar a alteração do valor da E/S associada, estando os botões desactivados e apenas a luz muda de cor dependendo do valor lógico 1 ou 0. É usada uma IST associada à interrupção de cada E/S configurada através da função "setupINT()" presente no ficheiro "WidgetIO.cpp" descrevendo-se de seguida como foi implementada.

Em primeiro lugar define-se a E/S à qual se vai associar a IST.

```
// GPIO Specific Section
dwGpio = m_io;
```

De seguida, através da função "GetGPIOIrq(dwGpio)" fornecida pela Toradex, obtêm-se o IRQ (*Interrupt Request)* associado à E/S.

```
dwIrq=GetGPIOIrq(dwGpio);
if(!dwIrq) MessageBox(NULL, L"Error INIT 1!", L"Interrupt", MB_OK);
```

Seguidamente define-se se a interrupção será nos dois sentidos, no descendente ou ascendente do sinal de entrada.

```
// Set Edge to trigger (Rising, Falling or both)
dwEdge = GPIO_EDGE_FALLING | GPIO_EDGE_RISING;
if(!SetGPIOIrqEdge(dwGpio, dwEdge))
        MessageBox(NULL, L"Error INIT 2!", L"Interrupt", MB_OK);
```

Agora regista-se um evento que aguardará pela interrupção da E/S, que será posteriormente associado.

```
// Create an Event to wait on
hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
if ( hEvent == NULL)
        MessageBox(NULL, L"Error INIT 3!", L"Interrupt", MB_OK);
```

Neste ponto obtêm-se a interrupção para o kernel associada ao IRQ.

```
// Get the SYSINTR that corresponds to dwIrq
dwSysIntr=RequestSysInterrupt(dwIrq);
if (!dwSysIntr)
        MessageBox(NULL, L"Error INIT 4!", L"Interrupt", MB_OK);
```

Associando-se agora o evento criado anteriormente à interrupção do kernel registada como interrupção associada à E/S desejada.

```
// Link our Event with the SYSINTR
if ( !InterruptInitializeCompat(dwSysIntr, hEvent, NULL, 0) )
        MessageBox(NULL, L"Error INIT 5!", L"Interrupt", MB_OK);
```

Agora que se configurou um evento que pode ser usado para analisar o estado da E/S à qual este foi associado, falta apenas criar uma thread que possa verificar a ocorrência do evento, para isso a seguinte thread foi definida:

```
while(1)
{       // Wait for Event (Interrupt)
        if (WaitForSingleObject(hEvent, INFINITE) == WAIT_OBJECT_0)
        {
                if(m_gpio->GetGPIOLevel(m_io) == 0) m_led->setState(false);
                else m_led->setState(true);

                InterruptDoneCompat(dwSysIntr);
        }
}
```

Dentro do ciclo infinito descrito acima usa-se uma função que aguarda pela sinalização de um evento, a função *WaitForSingleObject(hEvent ,INFINITE).* Desta forma aguarda indefinidamente até que o evento "hEvent", associado à interrupção numa entrada, fique sinalizado indicando a ocorrência de uma interrupção, de seguida analisa-se o valor lógico presente na entrada, definindo a cor do led como verde ou vermelha. No fim a função *InterruptDoneCompat(dwSysIntr)* liberta a interrupção associada ao evento ocorrido.

### 3.7.2 PWM CONTROL (QTPWM)

Após o desenvolvimento de uma primeira aplicação que possibilita a interacção com as E/S da placa Orchid, desenvolveu-se uma segunda aplicação que permite explorar a capacidade de PWM desta placa, Figura 45. Para isso utilizou-se os conhecimentos adquiridos anteriormente e explorou-se mais algumas capacidades da Framework Qt, como a utilização de gráficos.

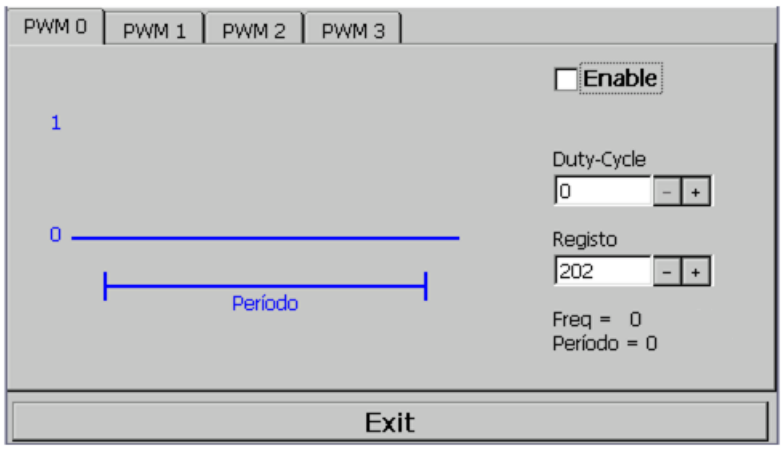



Figura 45 Aplicação em Qt para controlar o módulo de PWM

Esta aplicação permite seleccionar até quatro saídas de PWM presentes na placa Orchid, Figura 46, é através da modificação dos registos associados a este módulo que é possível indicar para cada uma dessas saídas a função actual, a frequência e o *duty cycle* correspondente.

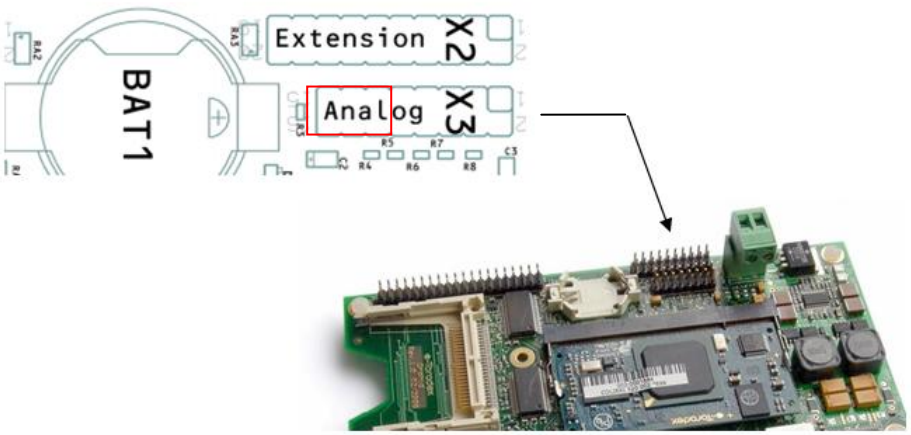



Figura 46 Localização dos pinos de PWM disponíveis na placa Orchid

Os registos associados ao módulo de PWM do processador PXA270 são os seguintes, Figura 47:

| Address | Name | Description | Page |
|---|---|---|---|
| 0x40B0_0000 | PWMCR0 | PWM 0 Control register | 23-7 |
| 0x40B0_0004 | PWMDCR0 | PWM 0 Duty Cycle register | 23-8 |
| 0x40B0_0008 | PWMPCR0 | PWM 0 Period register | 23-9 |
| 0x40B0_000C | — | reserved | — |
| 0x40B0_0010 | PWMCR2 | PWM 2 Control register | 23-7 |
| 0x40B0_0014 | PWMDCR2 | PWM 2 Duty Cycle register | 23-8 |
| 0x40B0_0018 | PWMPCR2 | PWM 2 Period register | 23-9 |
| 0x40B0_001C– 0x40BF_FFFC | — | reserved | — |
| 0x40C0_0000 | PWMCR1 | PWM 1 Control register | 23-7 |
| 0x40C0_0004 | PWMDCR1 | PWM 1 Duty Cycle register | 23-8 |
| 0x40C0_0008 | PWMPCR1 | PWM 1 Period register | 23-9 |
| 0x40C0_000C | — | reserved | — |
| 0x40C0_0010 | PWMCR3 | PWM 3 Control register | 23-7 |
| 0x40C0_0014 | PWMDCR3 | PWM 3 Duty Cycle register | 23-8 |
| 0x40C0_0018 | PWMPCR3 | PWM 3 Period register | 23-9 |
| 0x40C0_001C–0x40CF_FFFC | — | reserved | — |

Figura 47 Registos associados ao controlo do PWM [27]

O tamanho dos registos é idêntico aos já apresentados anteriormente, são ambos de 32bits, contudo cada registo varia o comportamento de uma saída de PWM modificando a frequência, duty cycle e funcionamento, para cada saída de PWM os seguintes registos devem ser alterados PWMCRx (*Control register*), PWMDCRx (*Duty Cycle*) e PWMPCRx (*Period Control*).

Para se aceder a estes registos através da aplicação executou-se os procedimentos já mencionados para a aplicação anterior, contudo a estrutura de dados para aceder ao módulo de PWM é a seguinte:

```
typedef volatile struct DATA
    {       DWORD PWMCR;    //Registo de Controlo
            DWORD PWMDCR;   //Duty Cycle
            DWORD PWMPCR;   //Período
    };
```

Com esta estrutura é possível então modificar os registos associados a uma saída de PWM, bastando modificar o endereço base (0x40B00000, 0x40B00010, 0x40C0000 ou 0x40C00010) para aceder às outras saídas.

Desta forma é possível controlar as quatro saídas de PWM presentes na placa Orchid, criando-se então uma aplicação para testar o controlo do módulo e implementando-se de uma forma visual a onda gerada com a modificação dos registos, Figura 48 e Figura 49.

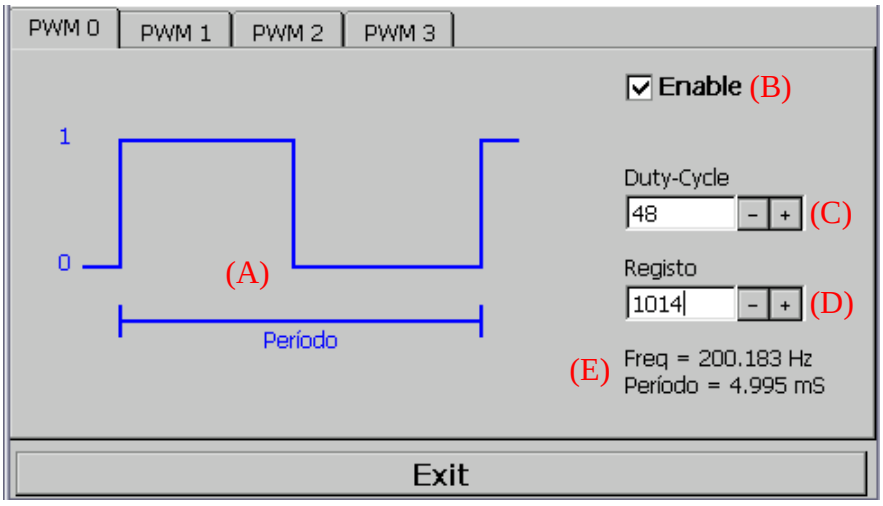



Figura 48 Exemplo 1 do funcionamento da aplicação de PWM

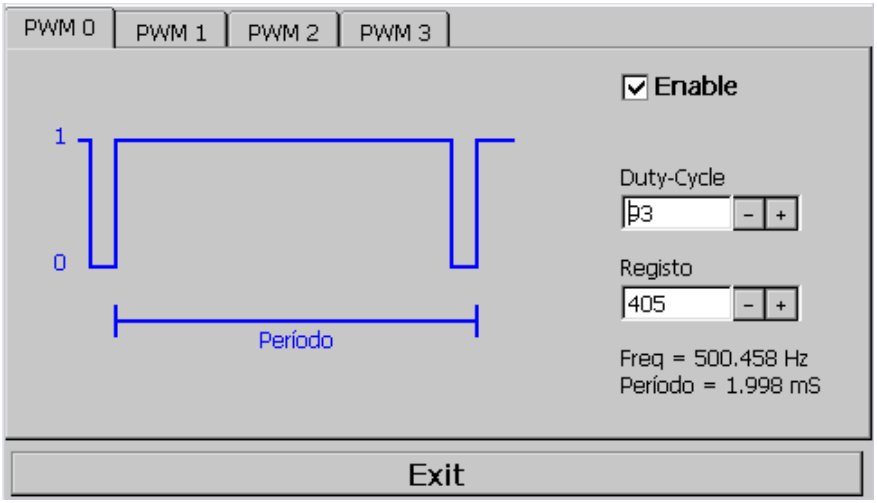



Figura 49 Exemplo 2 do funcionamento da aplicação de PWM

Como se pode verificar na Figura 48, a aplicação possui uma zona (A) onde se implementa de uma forma gráfica a onda gerada na saída. Além desta existe um QCheckBox (B) que permite activar/desactivar a saída de PWM, dois QSpinBox (C)(D) que permitem modificar os registos de referentes ao *duty cycle* e ao período, respectivamente e duas QLabel (E) que informam sobre o valor da frequência e do período.

### 3.7.1 I²C LEITURA/ESCRITA (QTI2C)

Com o conhecimento e experiência adquirida nas aplicações anteriores é possível agora desenvolver uma aplicação que permita controlar o módulo de $I^2C$, possibilitando controlar neste caso as placas de saídas e de entradas, Figura 50.

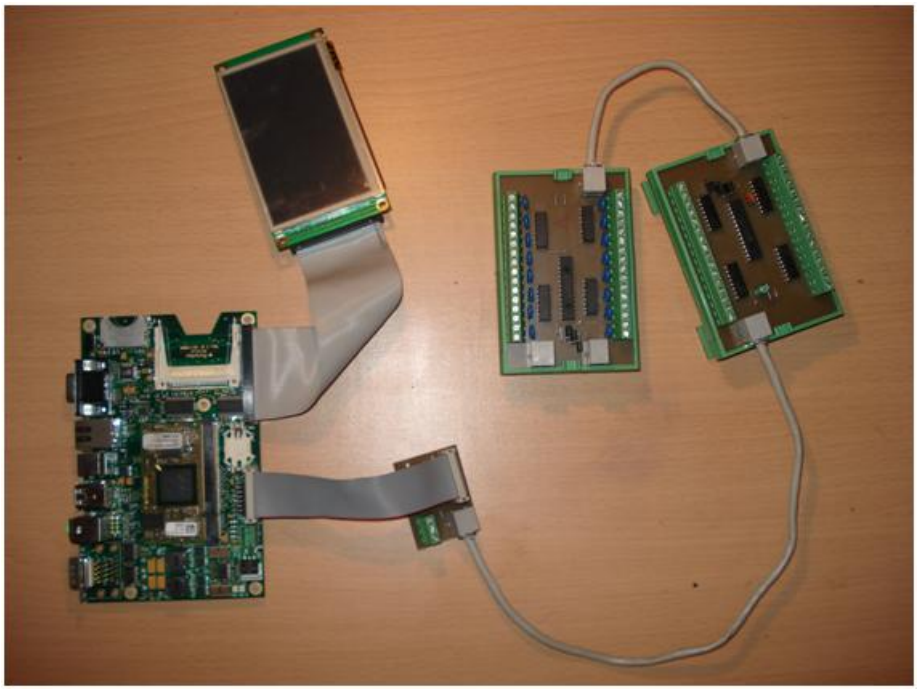


Figura 50 Sistema embebido da Toradex ligado às placas de expansão de E/S

Num barramento por $I^2C$ as linhas de SCL e SDA necessitam de resistências de *pull-up* para definir os níveis lógicos, como já referido estas resistências existem na placa Orchid, não sendo neste caso necessário aplicar nenhuma. É necessário também reservar dois pinos de E/S genérica para as interrupções fornecidas pela placa de expansão de entradas, é através destes que o sistema terá o conhecimento se existiu uma mudança dos valores lógicos nas entradas da placa de expansão, Figura 51.

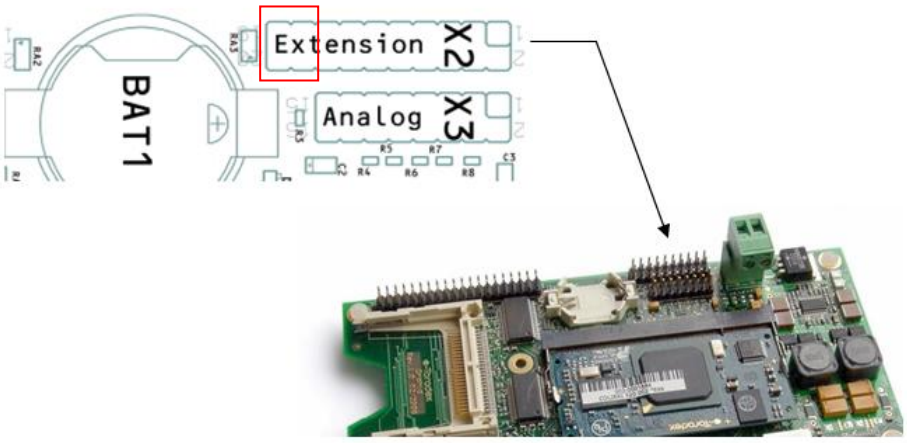



Figura 51 Localização dos pinos reservados para a interrupção externa

Para aceder ao módulo de $I^2C$ presente no processador PXA270, implementou-se uma estrutura de dados que permite mapear os registos associados ao módulo de $I^2C$, Figura 52, como já foi exemplificado nas aplicações descritas anteriormente.

| Address | Name | Description | Page |
|---|---|---|---|
| 0x4030_1680 | IBMR | $I^2C$ Bus Monitor register | 9-30 |
| 0x4030_1684 | — | reserved | — |
| 0x4030_1688 | IDBR | $I^2C$ Data Buffer register | 9-29 |
| 0x4030_168C | — | reserved | — |
| 0x4030_1690 | ICR | $I^2C$ Control register | 9-23 |
| 0x4030_1694 | — | reserved | — |
| 0x4030_1698 | ISR | $I^2C$ Status register | 9-26 |
| 0x4030_169C | — | reserved | — |
| 0x4030_16A0 | ISAR | $I^2C$ Slave Address register | 9-28 |
| 0x4030_16A4–<br>0x403F_FFFC | — | reserved | — |

Figura 52 Registos associados ao controlo do módulo de $I^2C$ [27]

O tamanho dos registos é idêntico aos já apresentados anteriormente, são ambos de 32bits, o registo IBMR (*Bus Monitor Register*) permite analisar os valores actuais dos pinos SCL e SDA, o registo IDBR (*Data Buffer Register*) guarda o valor de 8bits a transmitir, o ICR (*Control Register*) permite configurar diversos parâmetros relacionados com a transmissão dos dados (velocidade, interrupções, *master*/*slave*, etc.), o ISR (*Status Register*) indica a ocorrência de erros, interrupções, estado do barramento, modo de funcionamento e o registo ISAR (*Slave Address Register*) guarda o endereço do dispositivo *slave*.

Para se aceder aos registos do $I^2C$ é utilizada a seguinte estrutura de dados para os mapear:

```
typedef volatile struct I2C_STRUCT
    {
        DWORD IBMR;      //2 bits - I2C Bus Monitor Registers
        DWORD res1;
        DWORD IDBR;      //8 bits - I2C Data Buffer Register
        DWORD res2;
        DWORD ICR;       //16 bits - I2C Control Register
        DWORD res3;
        DWORD ISR;       //11 bits - I2C Status Register
        DWORD res4;
        DWORD ISAR;      //7 bits - I2C Slave Address Registers
    };
```

Com esta estrutura é então possível mapear e aceder aos registos do $I^2C$ de forma a implementar a comunicação com as placas de expansão, utilizando-se como endereço base o endereço que indica a posição do primeiro registo (0x40301680) do $I^2C$ e a função *void* MapRegister(DWORD pa)*.

Com o acesso aos registos implementado é necessário implementar rotinas que possam ser usadas para o controlo do $I^2C$, como por exemplo, rotinas de enviar/receber dados, detectar erros no barramento, detectar placas de expansão, entre outras.

Para uma melhor compreensão do funcionamento desta aplicação é possível analisar o seguinte diagrama, Figura 53.

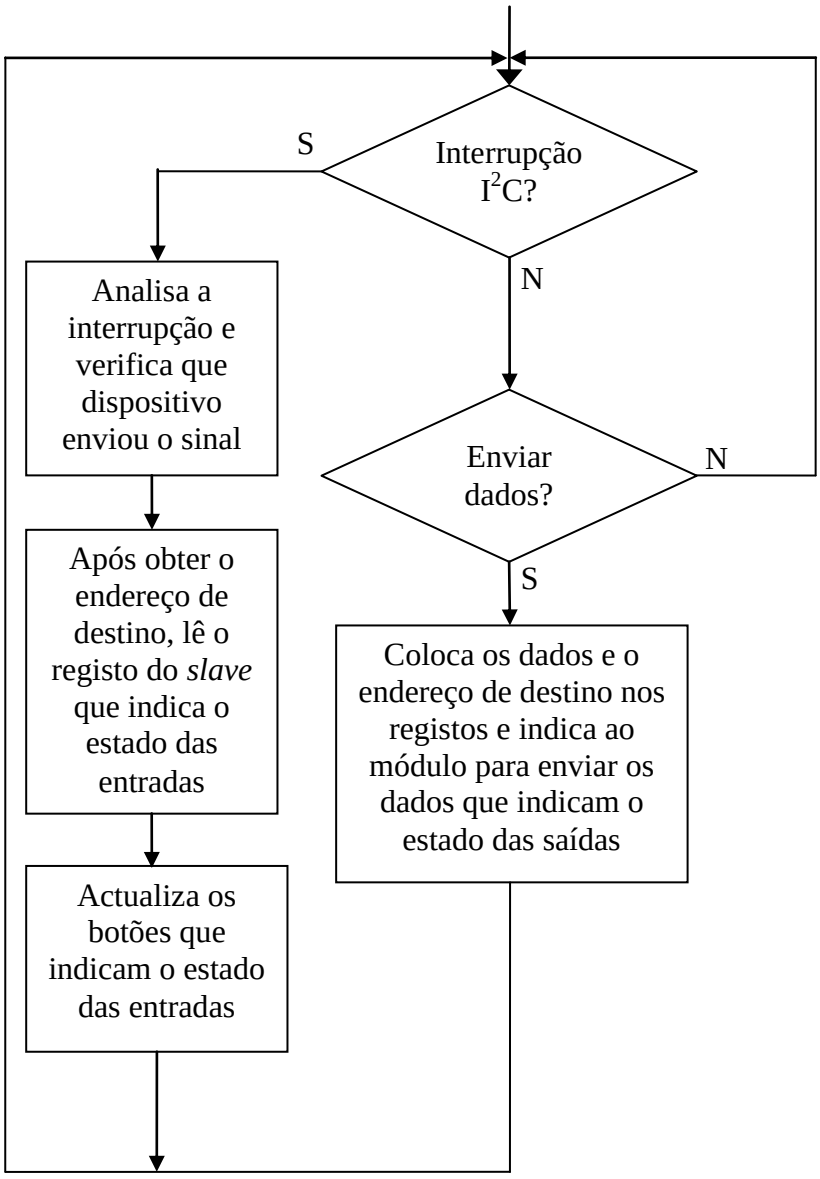



Figura 53 Diagrama de funcionamento da aplicação de $I^2C$

A actualização dos botões pode ser despoletada através de interrupção ou por *pooling*, nesta caso, a actualização foi implementada por interrupção, visto existirem sinais disponíveis de interrupção nas placas de expansão de entradas.

Desta forma e como explica o diagrama anterior, Figura 53, sempre que uma interrupção ocorre uma IST é sinalizada enviando de seguida um sinal a indicar que é necessário efectuar uma leitura das entradas de uma placa de expansão, ou seja, depois da sinalização e de terminar a *thread*, é indicado ao módulo de $I^2C$ para efectuar uma leitura dos registos de entrada do *slave* que enviou o sinal de interrupção. Com a obtenção do valor das entradas é possível actualizar os botões com o estado actual das entradas das placas de expansão detectadas.

### *Envio de dados*

Para efectuar a comunicação com as placas de expansão é usada a seguinte trama para enviar dados para o IC MCP23017, Figura 54.

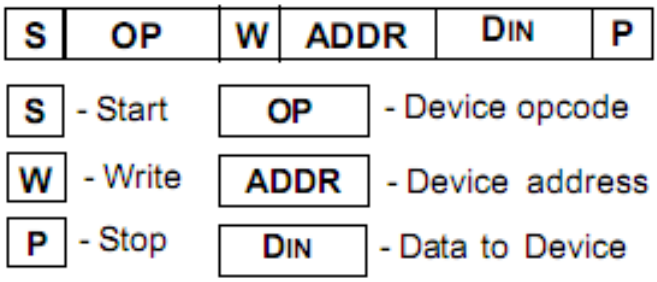



Figura 54 Descrição do campo de *OP* (endereço de destino e R/W) [24]

O endereço de um dispositivo *slave* no barramento de $I^2C$ é indicado no campo OP, este é constituído por 4 bits fixos, 3 bits programáveis e 1 bit para indicar escrita ou leitura, Figura 55.

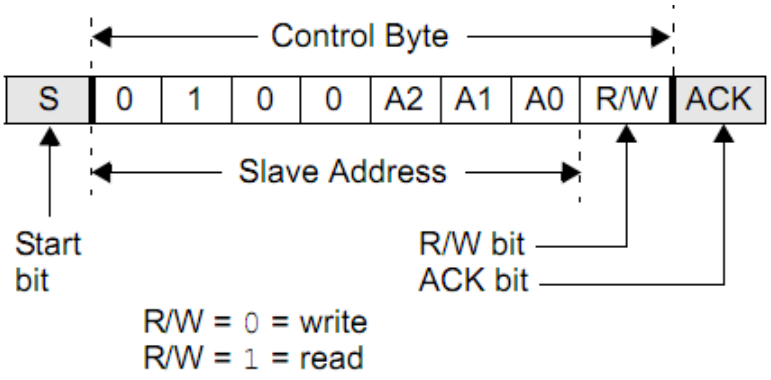



Figura 55 Descrição do campo de *OP* (endereço de destino e R/W) [24]

Os bits A2, A1 e A0 são definidos no dispositivo *slave* através de três pinos ligados a VCC ou a GND, no final de cada byte enviado um bit de ACK (*Acknowledgment*) é enviado para o *master* indicando a recepção correcta dos dados.

Após o envio do endereço do dispositivo (OP) é enviado o endereço do registo a modificar (ADDR) que neste caso é 12H para o porto GPA e 13H para o porto GPB, ambos de 8 bits, recebendo no final do envio do *byte* um ACK indicando a recepção dos dados, Figura 56.

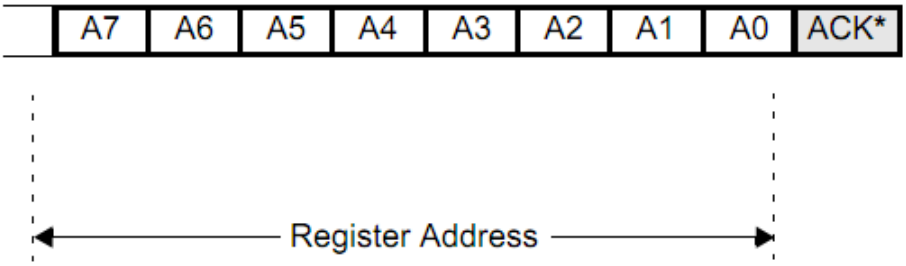



Figura 56 Descrição do campo referente ao endereço destino no dispositivo de $I^2C$ [24]

Após o envio do endereço do dispositivo e do endereço do registo envia-se de seguida o valor (DIN) a colocar no registo indicado, em que cada bit indica o valor a ser colocado nos pinos do porto, Figura 57.

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| GP7 | GP6 | GP5 | GP4 | GP3 | GP2 | GP1 | GP0 |
| bit 7 | | | | | | | bit 0 |

Figura 57 Descrição do registo de controlo do valor dos pinos do IC MCP23017 [24]

Para simplificar a comunicação com as placas de saída foi criada uma função onde foi encapsulada a trama, a função é a *SetIOPin(int device, int pin, bool value)* onde se indica o dispositivo destino, qual o pino a actuar e o valor a aplicar-lhe, tornando desta forma o envio de dados mais simples.

### *Recepção de dados*

Relativamente à recepção de dados através do barramento de $I^2C$, usada quando ocorre uma interrupção numa placa de expansão de entradas, obtendo-se o registo que indica o estado dos pinos do MCP23017 através de duas etapas distintas.

A primeira etapa requer que o apontador de registo interno do MCP23017 seja colocado no registo que se quer ler, para isso é efectuada uma escrita no registo a ler, contudo em vez de se escrever um valor efectua-se um *restart* (SR), Figura 58

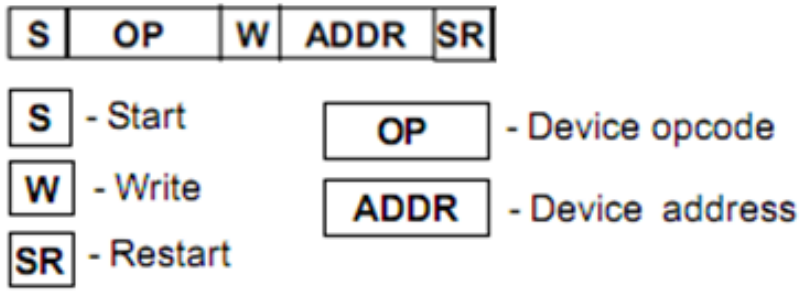



Figura 58 Frame para deslocação do apontador do IC MCP23017 [24]

Ao ser enviado um *restart* (SR) na trama o dispositivo de destino fica à espera que uma nova trama seja enviada, assim a segunda etapa é o envio de uma nova trama que em vez de ser do tipo escrita (W) é do tipo leitura (R) e em vez de o *master* indicar qual o registo a ler (indicado no inicio da transmissão) é o *master* que obtém o valor do registo previamente seleccionado enviado pelo *slave*, Figura 59.

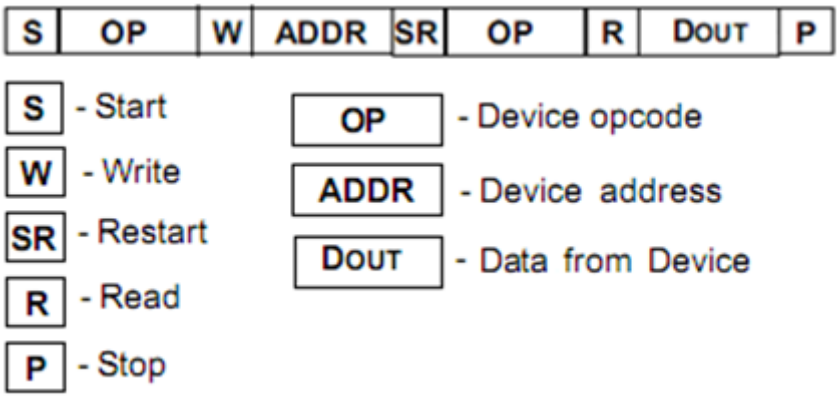



Figura 59 Frame para leitura de um registo do IC MCP23017 [24]

O *master* é capaz de obter o valor do registo do *slave* devido à sua mudança de funcionamento, passando de *master-transmitter* para *master-receiver*, desta forma o *master* comanda o sinal de *clock* (SCL) mas o pino de dados (SDA) no *master* é colocado como entrada até aparecer um *stop bit*.

Nas Figuras 60 e 61 é possível verificar o funcionamento da aplicação em sincronia com as placas de expansão, para isso são usados leds para facilitar a visualização.

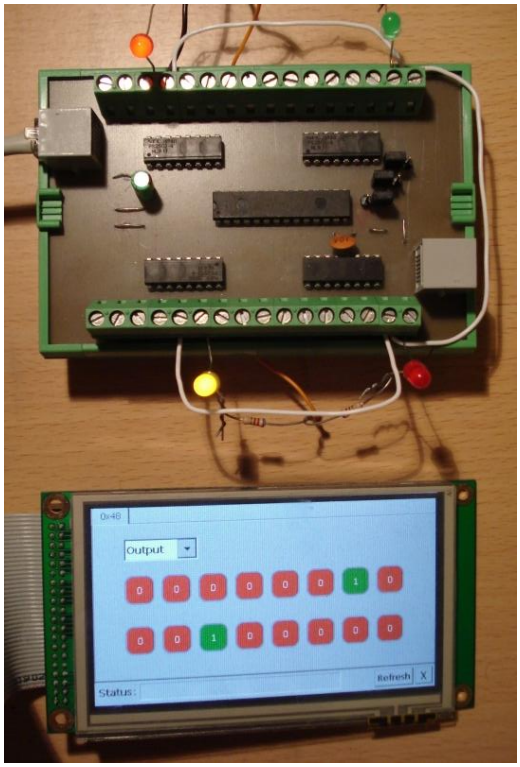



Figura 60 Exemplo 1 do funcionamento da aplicação QtI2C

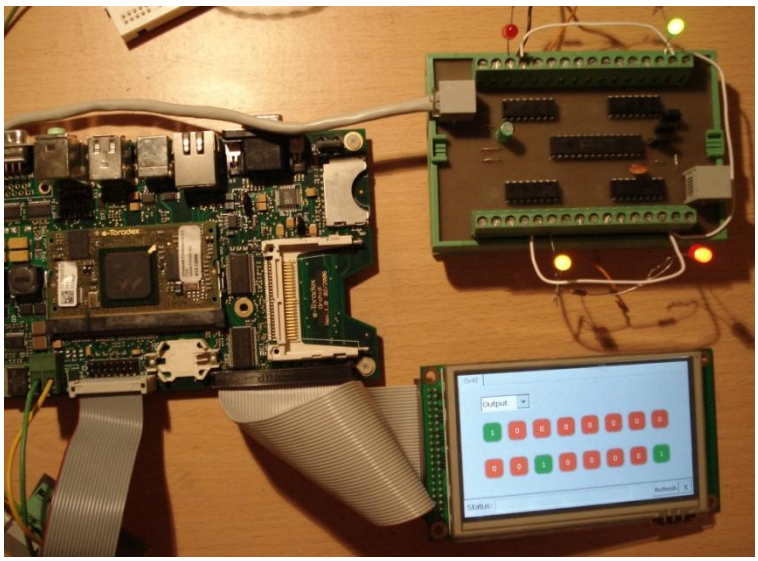


Figura 61 Exemplo 2 do funcionamento da aplicação QtI2C em vista geral

O ambiente visual desenvolvido, Figura 62, para o funcionamento com as placas de expansão é constituído por diversos controlos básicos, bem como um controlo desenvolvido especialmente para a aplicação.

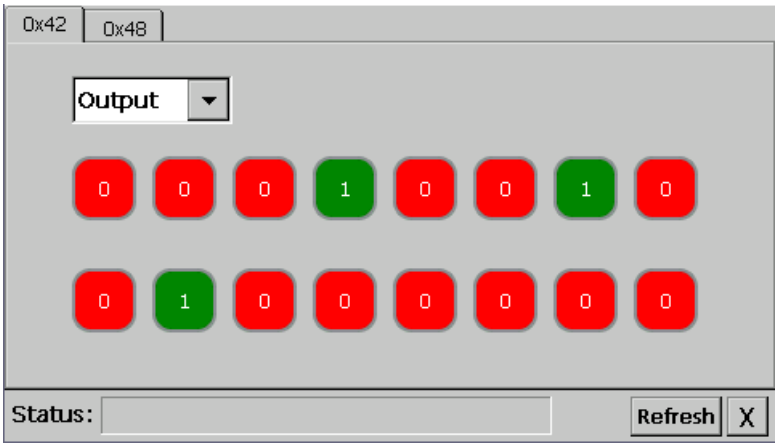



Figura 62 Exemplo da aplicação QtI2C

Entre os controlos básicos usados está o QLabel, o QComboBox para seleccionar o funcionamento da placa (Input/Output), o QTabWidget para disponibilizar diversas placas no mesmo ecrã, o QProgressBar que indica o estado da procura das placas no barramento e o QPushButton para fechar a aplicação e iniciar a procurar de novas placas no barramento.

Para o controlo das saídas e indicação do valor das entradas foi criado um novo tipo de botão, baseado no QPushButton, este botão fica vermelho se o valor nas placas de expansão for 0 e muda para verde se o valor for 1.

## 3.8 TESTES DE RESPOSTA TEMPORAL

Com o objectivo de aplicar este sistema no controlo e monitorização de dados de uma célula de aparafusamento é necessário analisar as capacidades de resposta temporal entre as placas de expansão por $I^2C$ (placa de saídas e a placa de entradas).

Para esta análise uma pequena aplicação corre na placa Colibri, o funcionamento da aplicação é o mais simples possível, aguardando pela ocorrência de uma interrupção e quando esta ocorrer efectua uma leitura da placa de expansão de entradas, reflectindo o valor das entradas na placa de expansão de saídas.

Numa primeira análise mais detalhada das várias etapas é possível verificar em primeiro o tempo necessário para enviar os dados do sistema embebido até à placa de expansão de saídas, Figura 63.

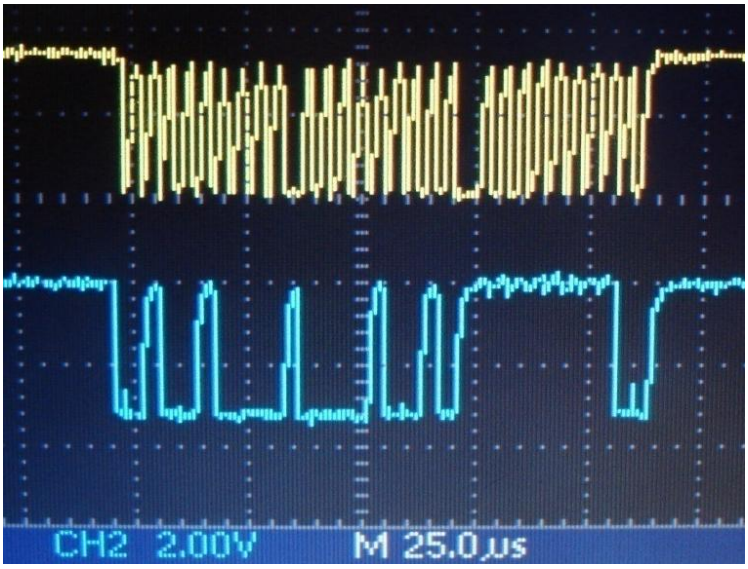



Figura 63 Actividade dos sinais SCL e SDA numa transmissão por $I^2C$

A figura anterior demonstra a transmissão pelo barramento de $I^2C$ a 400Kbs, onde o sinal de cor amarela é a linha de SCL (linha de *clock*) e o sinal de cor azul é a linha de SDA (linha de dados), o tempo necessário para transmitir uma frame com o endereço do dispositivo, endereço e valor de um registo, no total de 3 bytes, é aproximadamente 120us.

Ao tempo anterior é necessário somar o tempo que demora para que uma interrupção seja atendida pelo sistema embebido, porque desde o momento que a *thread* é despoletada até se produzir um resultado numa saída existe um atraso que deve ser considerado para o cálculo final do tempo mínimo necessário para produzir uma reacção, Figura 64.

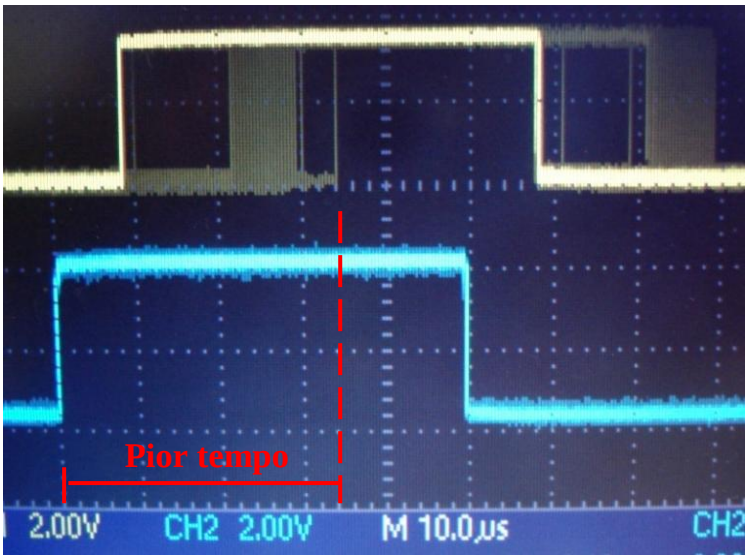



Figura 64 Tempo de resposta a uma interrupção na placa Orchid

A figura anterior demonstra a variação entre uma onda quadrada de 10KHz (linha de cor azul) introduzida no sistema embebido e a reprodução da mesma num pino de saída (linha de cor amarela) do sistema. É possível verificar que existe um atraso entre os dois sinais, este atraso é provocado pelo tempo necessário para que a ISR possa ser executada (tempo do passo 1 mais o tempo do passo 2) e envie um sinal para que a IST execute a *thread* (tempo do passo 3 mais o tempo do passo 4) que modifica o valor do pino de saída (tempo do passo 5), Figura 65.

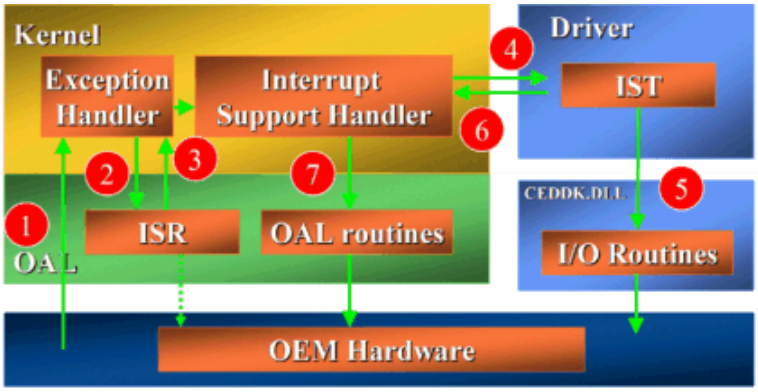



Figura 65 Sequencia de execução do atendimento a uma interrupção [28]

Este tempo varia dependendo da carga exigida ao processador em cada instante, no entanto como se pode verificar na Figura 64, o tempo nunca é superior a aproximadamente 35us podendo ser tão reduzido como 8us.

Para a leitura dos dados através do barramento de $I^2C$ é necessário enviar 2 bytes e receber 2 bytes, como se verificou anteriormente, sabendo-se que este transmite e recebe aproximadamente a 400Kbs, o tempo de transmissão previsto de 1 byte mais o bit de ACK é de aproximadamente 45us, então o tempo de leitura de um registo das placas de expansão de entradas será:

$$Tempo\ de\ leitura = 4 * 45us = 180us$$

Assim somando o tempo de atraso das interrupções mais o tempo de leitura de um registo e mais o tempo de escrita de um registo obtêm-se:

$$Tempo\ de\ total = 35us + 180us\ + 120us\ = 335us$$

Este é o tempo necessário para produzir uma resposta numa saída da placa de expansão, quando recebe um sinal na placa de expansão de entradas. Contudo, este valor modifica-se em função da carga do processador, da configuração dos registos do módulo de $I^2C$ e do tempo de resposta dos próprios integrados de expansão por $I^2C$ (MCP23017). Por motivos de segurança duplicou-se o tempo de resposta obtida para 670us de forma a possuir uma referência temporal que será sempre cumprida, visto que em situações normais só demorará no máximo 335us.

Como teste final, aplicou-se uma onda quadrada com frequência de 1KHz numa entrada da placa de expansão de entradas, observando-se uma onda quadrada na placa de expansão de saídas com a mesma frequência mas desfasada no tempo (tempo de transmissão mais o tempo indeterminado de processamento), Figura 66.

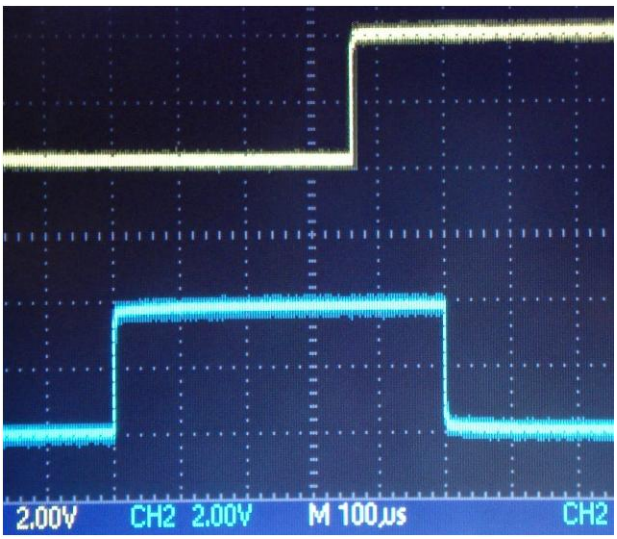



Figura 66 Resposta temporal das placas de expansão a uma onda quadrada

## 3.9 CONCLUSÕES

Neste capítulo foram analisados os requisitos temporais para sistemas embebidos aplicados as sistemas industriais, concluindo-se que o tempo de resposta mais adequado seria de 1ms. Para aplicar o sistema embebido em sistemas industriais foi desenvolvido placas de expansão de entradas e saídas através de um barramento com o protocolo $I^2C$, resultando num sistema versátil capaz de possuir no máximo de 8 placas de expansão de entradas ou saída num máximo de 128 E/S.

Para facilitar o controlo destas placas e pensando na facilidade de utilização, discutiu-se as diferenças entre diversos ambientes gráficos, resultando numa escolha sobre a Framework Qt, permitindo a criação de ambientes gráficos de grande qualidade.

No final realizou-se testes sobre o sistema implementado e analisou-se um dos principais requisitos iniciais (tempo resposta máximo de 1ms) o qual foi cumprido, visto o tempo de resposta ser de aproximadamente 335us, duplicando por segurança este tempo obtêm-se um tempo de 670us, ficando igualmente abaixo da referência de 1ms definida inicialmente.

Com esta análise é possível concluir que este sistema embebido pode ser implementado no controlo e monitorização de células industriais, permitindo a expansão de E/S conforme as necessidades de uma forma pouco complexa.

# CAPÍTULO 4

# 4 CÉLULA DE APARAFUSAMENTO AUTOMÁTICO

Com a exploração do sistema embebido da Toradex e o desenvolvimento das placas de expansão concluído é agora possível aplicar o sistema embebido no controlo e monitorização de uma célula de aparafusamento automático, Figura 67.

O desenvolvimento de um sistema de aparafusamento automático de PCB (Placas de Circuito Impresso) foi proposto pela empresa SILGAL, com o objectivo de reduzir os custos relativos aos equipamentos necessários mantendo o sistema flexível para que este possa ser integrado numa linha de montagem capaz de aparafusar vários PCB a chassis diferentes, nesta perspectiva, foi decidida a utilização do sistema embebido da Toradex.

Neste capítulo será descrito os requisitos impostos pelo cliente, o hardware utilizado para o desenvolvimento da célula de trabalho e a sua implementação.


Figura 67 Célula de aparafusamento automático de PCBs

## 4.1 REQUISITOS DO CLIENTE

### 4.1.1 SOFTWARE

O sistema de aparafusamento pedido pelo cliente possuí vários requisitos relativos ao software, mais precisamente no respeitante à informação que será apresentada ao operador e como esta deve ser guardada para efeitos de histórico. Os principais requisitos foram os seguintes:

- Possibilidade de escolha do programa de aparafusamento
- Indicação se os parafusos foram bem ou mal aparafusados
- Indicação do progresso do aparafusamento
- Contagem do tempo de aparafusamento
- Contagem das placas bem e mal aparafusadas
- Contagem dos parafusos gastos (estimativa)
- Indicação do estado da célula de aparafusamento
- Guardar os dados estatísticos (nº placas bem e mal aparafusadas, tempo médio, estimativa de parafusos gastos) referentes ao dia de trabalho

### 4.1.2 HARDWARE

Relativamente ao hardware requerido pelo cliente foi feita referência à forma de como o sistema deveria receber o chassi e o PCB, aparafusá-lo e como deveria efectuar a sua saída da célula. Os principais requisitos foram os seguintes:

- A entrada e saída do chassi e do PCB deve ser efectuada através de tapetes rolantes (*conveyors).*
- O chassi e a placa devem ser centrados antes do aparafusamento
- A aparafusadora a utilizar será eléctrica podendo-se ajustar o *torque*
- Deve existir um sistema de entrega automática de parafusos à aparafusadora
- O sistema de posicionamento da aparafusadora deve ser suficientemente rápido para que uma placa de dezasseis parafusos fique pronta num tempo máximo de 90 segundos, incluindo o tempo de entrada e saída.

Os requisitos acima descritos são meramente guias para a implementação da célula, podendo ser modificados de forma a melhorar ou facilitar o processo de aparafusamento, o hardware não mencionado refere-se a pequenos sensores ou actuadores utilizados para o correcto funcionamento da célula de aparafusamento.

## 4.2 HARDWARE SELECCIONADO

Seguidamente descreve-se o hardware mais importante seleccionado para a implementação da célula de aparafusamento, sendo explicado o funcionamento e a função de cada um no sistema.

### 4.2.1 EIXOS E CONTROLADORES AEB

A implementação do sistema de posicionamento foi realizada recorrendo-se a três eixos independentes, ou seja, uma montagem cartesiana de eixos accionados com motores de passo, Figura 68:

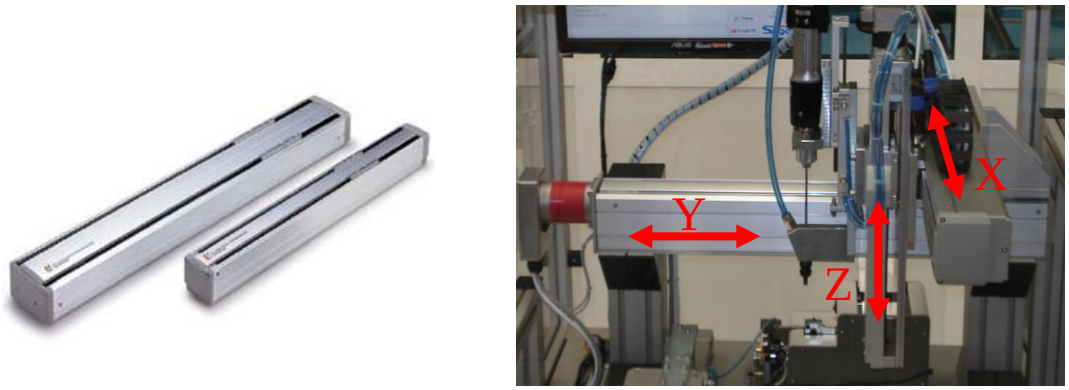



Figura 68 Montagem dos eixos lineares SMC LJ1 [29]

Os eixos são do tipo *Ball Screw*, onde o movimento rotacional do veio central accionado pelo motor de passo é transformado em movimento linear, desta forma é possível fazer deslocar uma base ao longo do eixo. Os eixos X e Y possuem um deslocamento de 500mm e 300mm para o eixo Z, todos estes eixos são da marca SMC.

Os motores de passo utilizados para o accionamento dos eixos são da marca AEB Robotics, Figura 69, são motores bipolares com um *encoder* óptico interno com duas saídas desfasadas, à excepção do motor do eixo Z que não possui *encoder* mas possui um travão electromagnético.

Figura 69 Motor de passo AEB MOTOE56LR [30]

Para o controlo foi utilizado os controladores recomendados para os motores de passo, os RBT5 também da AEB Robotics, Figura 70.

Figura 70 Controlador de eixos RBT5 para motores de passo da AEB Robotics [31]

Estes controladores permitem ser pré-programados com os movimentos desejados, possuindo capacidade para 255 programas diferentes seleccionados através da combinação de 8 entradas ($2^8 = 256$), onde o programa 0 está implementado de fábrica como sendo a procura pela posição *HOME*. Possui para além das 8 entradas para selecção do programa mais 4 entradas e 5 saídas genéricas.

Possui uma entrada de *strobe* que indica o arranque do programa seleccionado, três entradas para os sensores de fim de curso e *home*, uma entrada de emergência, uma entrada de reset e uma entrada de *hold* que permite parar o programa na posição actual e continuar mais tarde, Figura 71.

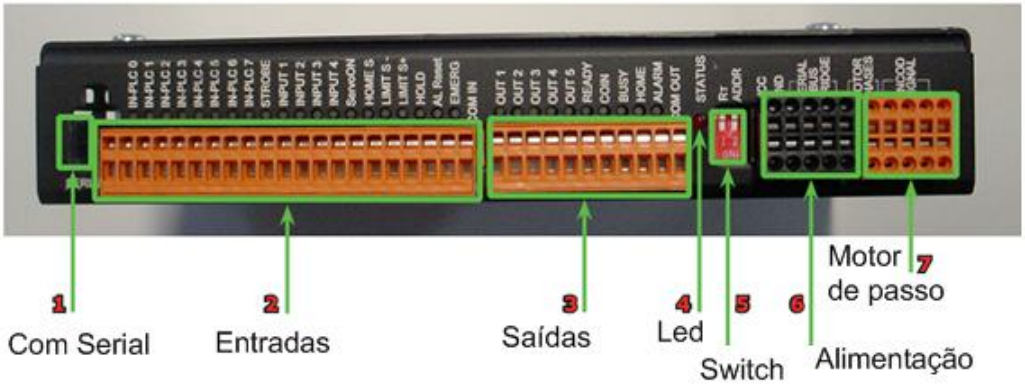



Figura 71 Descrição dos conectores disponíveis no controlador RBT5 [31]

Para a programação dos eixos o fabricante fornece um software específico para estes controladores, o qual permite entre outras opções configurar os parâmetros mecânicos de forma a melhorar a performance e precisão do sistema de eixos, Figura 72.

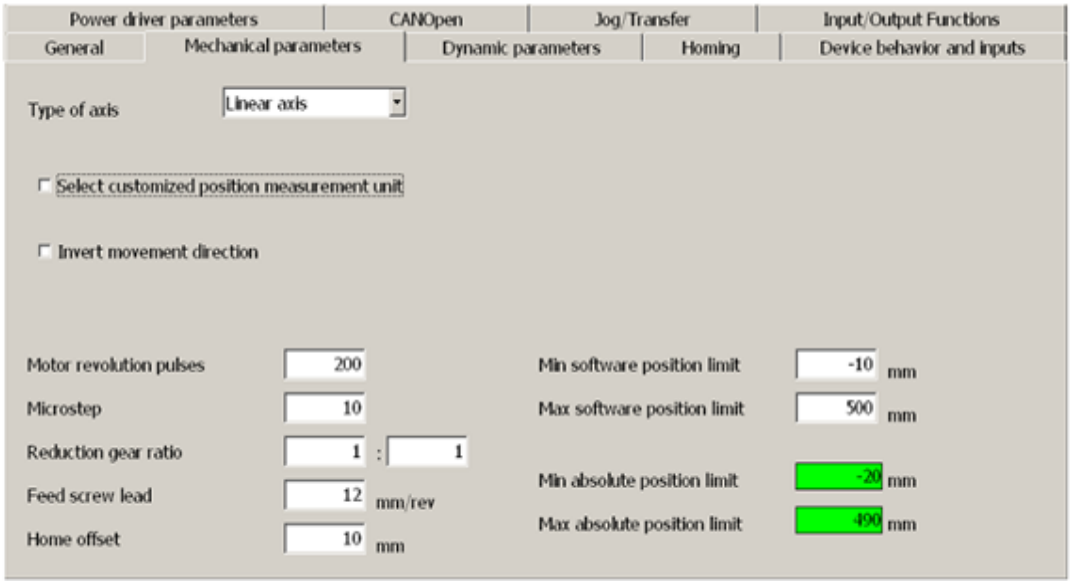



Figura 72 Software de configuração dos parâmetros mecânicos [31]

Para programar uma sequência de movimentos o software implementa uma lista de instruções que serão executados sequencialmente. Para se definir uma instrução existem pequenos blocos pré-definidos com as instruções básicas de controlo do movimento e das E/S, além destas, existem instruções condicionais e de espera, Figura 73.


Figura 73 Lista de controlos disponíveis no software fornecido [31]

Posteriormente serão melhor detalhadas todas as instruções, bem como o processo de programação dos controladores e o mecanismo de sincronismo implementado.

### 4.2.2 DRIVERS E MOTORES DC

A entrada do chassi e do PCB na máquina de aparafusamento foi efectuada através de *conveyors* de correias, Figura 74, para isso foram utilizados dois motores DC de 24V com redução, com um *torque* de 0.2Nm e 230rpm à saída da caixa redutora, Figura 75.

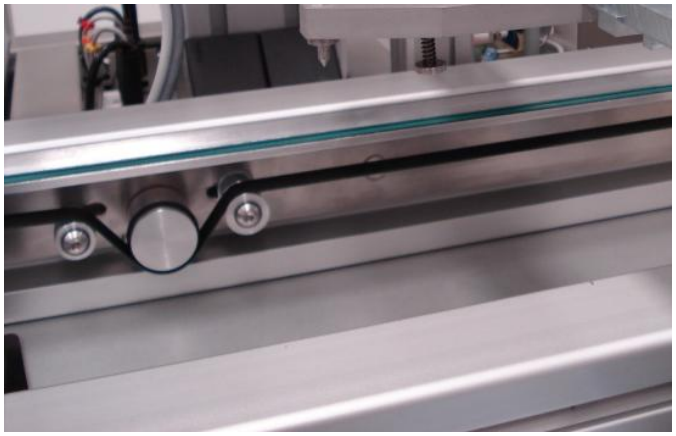


Figura 74 *Conveyor* de entrada/saída da célula de trabalho


Figura 75 Motor DC 24V 230rpm da RS [32]

Para o controlo dos motores DC utilizou-se um driver da marca MVA modelo MVA2001 que permite um sentido de rotação dos motores possibilitando o controlo até dois motores, com ajuste de velocidade e corrente máxima para os dois, Figura 76.


Figura 76 Placa de controlo dos motores DC

### 4.2.3 APARAFUSADORA ELÉCTRICA

O sistema de aparafusamento é constituído por uma aparafusadora HIOS CL-6000 com regulação do *torque* aplicado durante o aparafusamento através do ajuste de uma mola de pressão presente na ponta da aparafusadora. A aparafusadora é controlada por um driver específico para esta que possibilita o ajuste da velocidade, o arranque ou a paragem do aparafusamento, Figura 77.


Figura 77 Aparafusadora eléctrica CL-6000 e controlador da marca HIOS

O driver da aparafusadora é alimentado a 230V e possui dois sinais de entrada, um de START que inicia o aparafusamento e um de RESET que interrompe a aparafusadora, possui ainda o FINISH como sinal de saída, indicando a obtenção do *torque* desejado, Figura 78, sendo possível configurar de uma a três contagens, ou seja, o sinal de FINISH é obtido quando ocorre N vezes o *torque*, no caso da figura anterior o FINISH é dado ao primeiro sinal de *torque*.

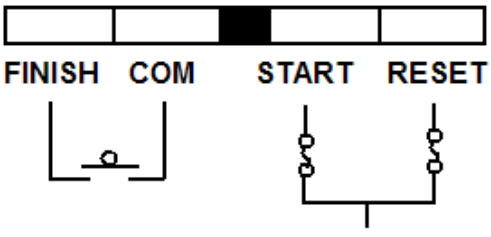



Figura 78 Sinais de comando e sinalização do controlador da aparafusadora

### 4.2.4 DISPENSADOR DE PARAFUSOS

Para efectuar a dispensa dos parafusos foi utilizado um sistema automático de dispensa dos parafusos da marca JANOME modelo Quicher NSR, Figura 79.

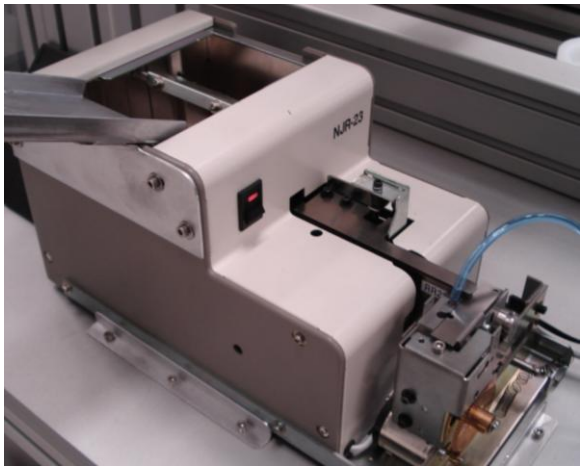


Figura 79 Dispensador automático de parafusos da marca JANOME

Este sistema possui um depósito de parafusos que através de uma guia vibratória alinha e faz deslizar os parafusos até à outra extremidade da guia, na extremidade os parafusos estão alinhados e prontos para serem puxados para um tubo que os enviará para a ponta da aparafusadora eléctrica.

Para enviar um parafuso aplica-se um sinal durante um segundo no Quicher para que este possa ir buscar um parafuso, um sensor colocado no tubo fornece feedback sobre se passou algum parafuso ou não, no caso de ocorrer algum problema no envio do parafuso é efectuado o pedido de um segundo parafuso.

### 4.2.5 ACTUADORES PNEUMÁTICOS

Os actuadores pneumáticos utilizados na concepção da célula de aparafusamento são todos da marca SMC, Figura 80. São utilizados como *stopper* para o chassi de entrada, para subir e descer a aparafusadora eléctrica e na centragem da placa com o chassi em que foram usados neste caso dois actuadores.

Figura 80 Esquerda: mesa pneumática. Direita: actuador pneumático [33]

Todos os actuadores possuem duas entradas de ar possibilitando que o cilindro no interior se mova em duas direcções, cada entrada pode ser regulada em pressão permitindo regular a velocidade do cilindro.

### 4.2.6 VÁLVULAS PNEUMÁTICAS E FILTROS

As válvulas pneumáticas utilizadas são da marca SMC e permitem controlar os actuadores pneumáticos da mesma marca, Figura 81, estas funcionam a 24V e efectuam a passagem do ar para os actuadores ou a descarga destes em sentido contrário.

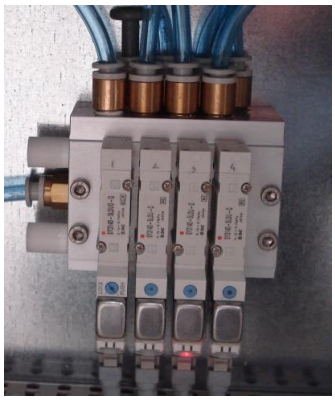


Figura 81 Conjunto de válvulas pneumáticas da marca SMC

Além das válvulas para os actuadores é necessária uma válvula para o sopro do parafuso, para que o parafuso seja levado para a extremidade da aparafusadora.

Na entrada de ar é utilizado um filtro de ar com regulador de pressão, Figura 82, de forma a garantir ar limpo sem impurezas que danifiquem os cilindros e uma pressão constante, nunca excedendo os valores máximos admitidos pelo sistema pneumático.

Figura 82 Válvula de pressão e filtro de entrada de ar da célula

### 4.2.7 SISTEMA DE SEGURANÇA

Para a protecção do operador e da célula de aparafusamento, foi construída uma estrutura metálica em alumínio com as laterais em acrílico de forma a impedir a entrada de objectos que possam interferir com o processo de aparafusamento. Além desta estrutura, existe uma porta de acesso para manutenção que possui um detector de porta aberta da marca *Schmersal*, Figura 83. É especialmente concebido para estas aplicações porque possui um mecanismo de fecho com um actuador separado que não permite que este seja sabotado.

Figura 83 Mecanismo de fecho da porta de acesso [34]

### 4.2.8 SINALIZAÇÃO

A indicação do estado de funcionamento de uma célula industrial é um requisito obrigatório, em muitos casos é utilizada sinalização luminosa, Figura 84.

Possui luzes para informar o operador se a célula está em funcionamento (luz verde), com necessidade de atenção (luz amarela) ou que não está em funcionamento (luz vermelha).

Figura 84 Torre sinalizadora da Moeller [35]

## 4.3 IMPLEMENTAÇÃO DO SISTEMA

A implementação da célula de aparafusamento automático passou por diversas fases de desenvolvimento até que estivesse de acordo com os requisitos impostos pelo cliente. Como é usual no desenvolvimento de máquinas customizadas, a fase inicial constou de uma avaliação global acerca das partes mecânicas, pneumáticas, eléctricas/electrónicas e arquitectura de software que iriam ser integradas na célula.

De seguida será efectuada uma breve descrição sobre as partes mecânica, pneumática, eléctrica e electrónica dos mecanismos envolvidos, ficando a arquitectura de software para o capítulo seguinte.

### 4.3.1 DESCRIÇÃO DA ESTRUTURA MECÂNICA

Em termos estruturais a célula é composta por duas partes fundamentais:

- Estrutura inferior
- Estrutura superior

A estrutura inferior, Figura 85, é construída em perfil de alumínio extrudido (80x80), uma base em alumínio (800x800x20), um tampo (800x800x20) em alumínio e painéis laterais em chapa de alumínio com 6mm de espessura. As ligações são feitas por aparafusamento dos perfis estruturais através de acessórios adequados.

Esta estrutura alberga os sistemas pneumáticos e eléctricos, bem como o sistema embebido da Toradex para o controlo da célula de aparafusamento.

Figura 85 Estrutura inferior da célula de trabalho

O desenho em baixo, Figura 86, representa a estrutura inferior em AutoCAD. Nesta pode-se encontrar as dimensões de atravancamento, 800 x 725 x 800 mm (largura x altura x profundidade).

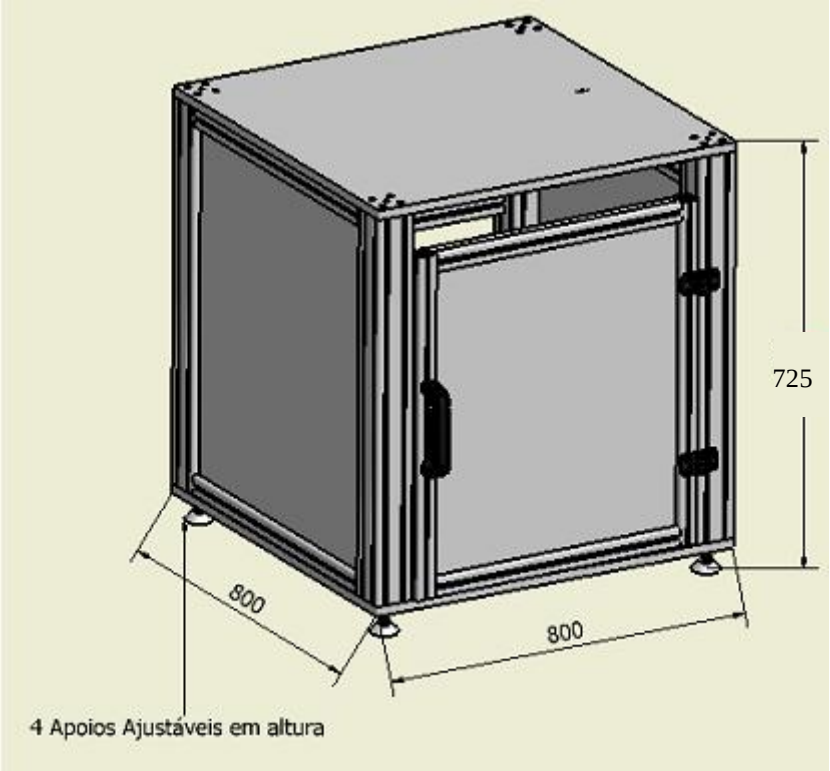



Figura 86 Desenho da estrutura inferior com dimensões

Esta estrutura comporta quatro apoios metálicos ajustáveis que permitem o nivelamento e a regulação em altura da máquina. A máquina foi desenvolvida de forma a apresentar uma altura máxima inferior a 1650mm que contempla uma altura de trabalho de cerca de 900mm. A estrutura e os seus apoios ajustáveis permitem a regulação da célula em altura de aproximadamente 35mm. Esta estrutura deve ser mantida nivelada, para o bom funcionamento do sistema de eixos.

A estrutura é fechada através de painéis (chapas) encaixados nos perfis de alumínio e o painel frontal, possui uma porta de acesso ao interior e uma gaveta:

i. Porta Frontal e gaveta da estrutura inferior, Figura 87 – Partes através das quais se tem acesso a um teclado e demais sistemas eléctricos e de controlo da célula:

   A. Porta de acesso ao hardware

   B. Gaveta de interface com o operador

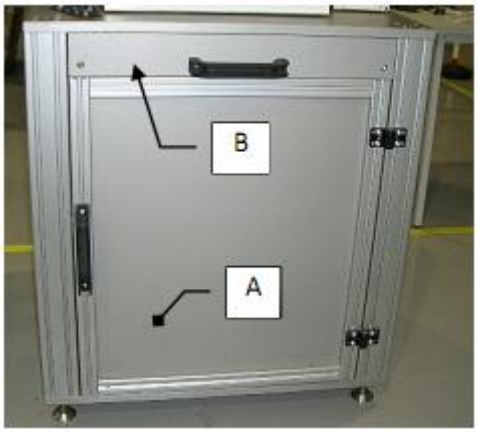



Figura 87 Portas de acesso da estrutura inferior

ii. Porta de acesso traseiro, Figura 88 – Nestes painéis é possível por intermédio de uma chave restrita e após a sua abertura, obter acesso às fontes de alimentação do sistema eléctrico e às válvulas do sistema pneumático.

A. Porta de acesso ao quadro eléctrico
B. Porta de acesso ao quadro pneumático

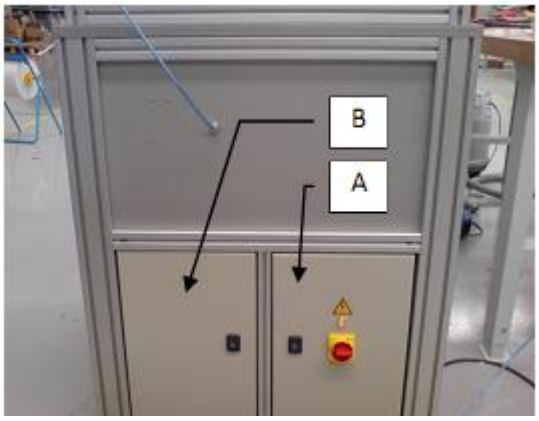



Figura 88 Portas de acesso ao quadro eléctrico e pneumático da estrutura inferior

### 4.3.2 SISTEMA PNEUMÁTICO

Na implementação do sistema pneumático utilizaram-se quatro válvulas, descritas anteriormente, para controlo dos diversos actuadores pneumáticos e para o sopro do parafuso. A utilização de filtros e reguladores de ar é exigida de forma a garantir que os limites de pressão admissíveis pelo sistema nunca são ultrapassados.

O diagrama de ligações seguinte demonstra o sentido do fluxo do ar nos diversos actuadores e válvulas utilizadas e como estas foram interligadas, Figura 89:

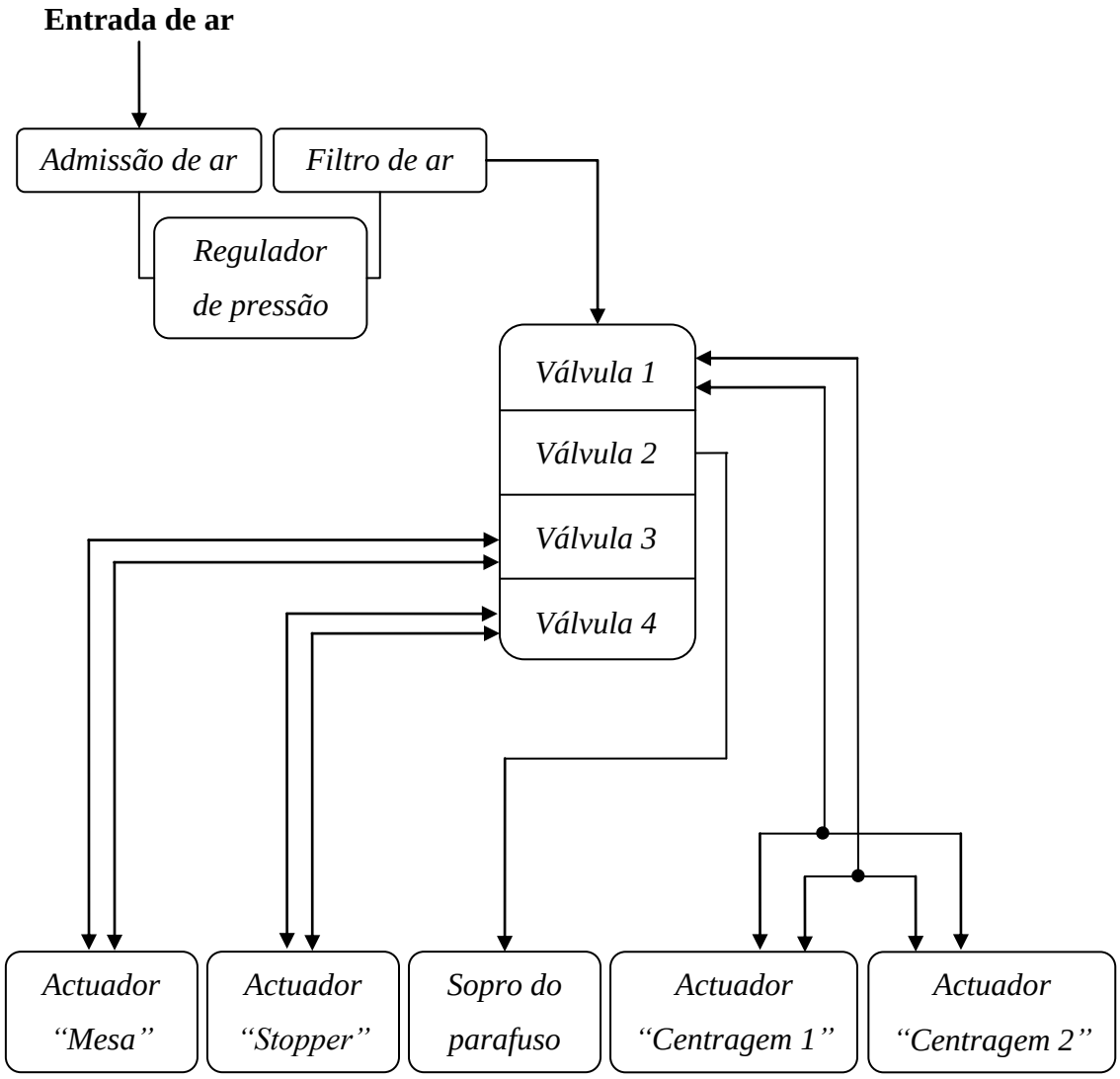



Figura 89 Diagrama das ligações pneumáticas da célula de trabalho

No diagrama as setas indicam o sentido do ar a partir de cada válvula, desta forma a válvula dois apenas deixa sair ar em direcção ao tubo que leva o parafuso até à ponta da aparafusadora, no entanto a válvula um permite controlar dois actuadores nos dois sentidos, quando uma entrada é colocada sobre pressão a outra retorna o ar presente no cilindro do actuador.

Na Figura 90 é possível verificar a localização de cada um dos actuadores pneumáticos e do sopro do parafuso.

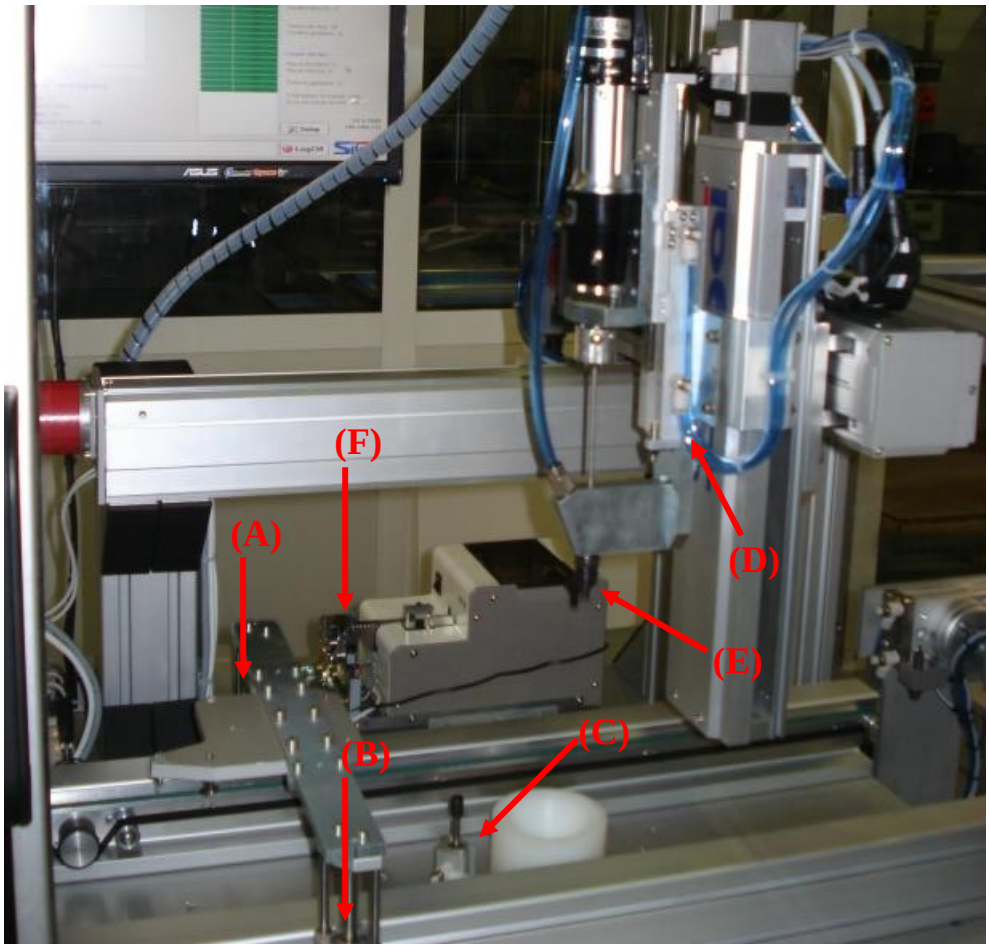



Figura 90 Descrição da localização dos actuadores pneumáticos

Os actuadores pneumáticos (A) e (B) na figura anterior, são actuados em simultâneo sendo utilizados na centragem do PCB com o chassi através de pernos guia. O actuador (C) é denominado por *stopper* e a principal função deste é bloquear a passagem do chassi que vem nos *conveyors* para que o sistema de centragem possa descer com precisão sobre o mesmo. O actuador (D) é uma mesa pneumática que permite, neste caso, acoplar uma aparafusadora eléctrica e deslocar 100mm para cima para que o parafuso possa ser colocado na ponta da aparafusadora (E).

Para alimentar a aparafusadora, além de um dispensador de parafusos é necessária a aplicação de um sopro (F) para que o parafuso seja transportado através de um tubo com um diâmetro aproximado ao da cabeça deste, até ao bico da aparafusadora.

### 4.3.3 SISTEMA DE ENTRADA DO CHASSI E CENTRAGEM

A célula de aparafusamento automático foi desenvolvida com o objectivo de integrar uma linha de produção automatizada, para tal a entrada da placa na célula deve ser efectuada de uma forma automática. Com este objectivo, utilizou-se um *conveyor* de correias, descrito anteriormente, ajustado ao chassi que o leva até à zona de centragem e posteriormente até à saída da célula de trabalho.

O controlo do *conveyor* depende dos sinais de entrada e saída, obtidos através dos sinais de *Smema* enviados pelas máquinas anterior e posterior, sinais que indicam se as máquinas estão ocupadas ou livres, também é usado um sinal de posição (sensor óptico) permitindo assim que este desloque o chassi para a posição destinada em cada fase do processo de aparafusamento, no diagrama seguinte é possível analisar as acções associadas ao *conveyor*, Figura 91.

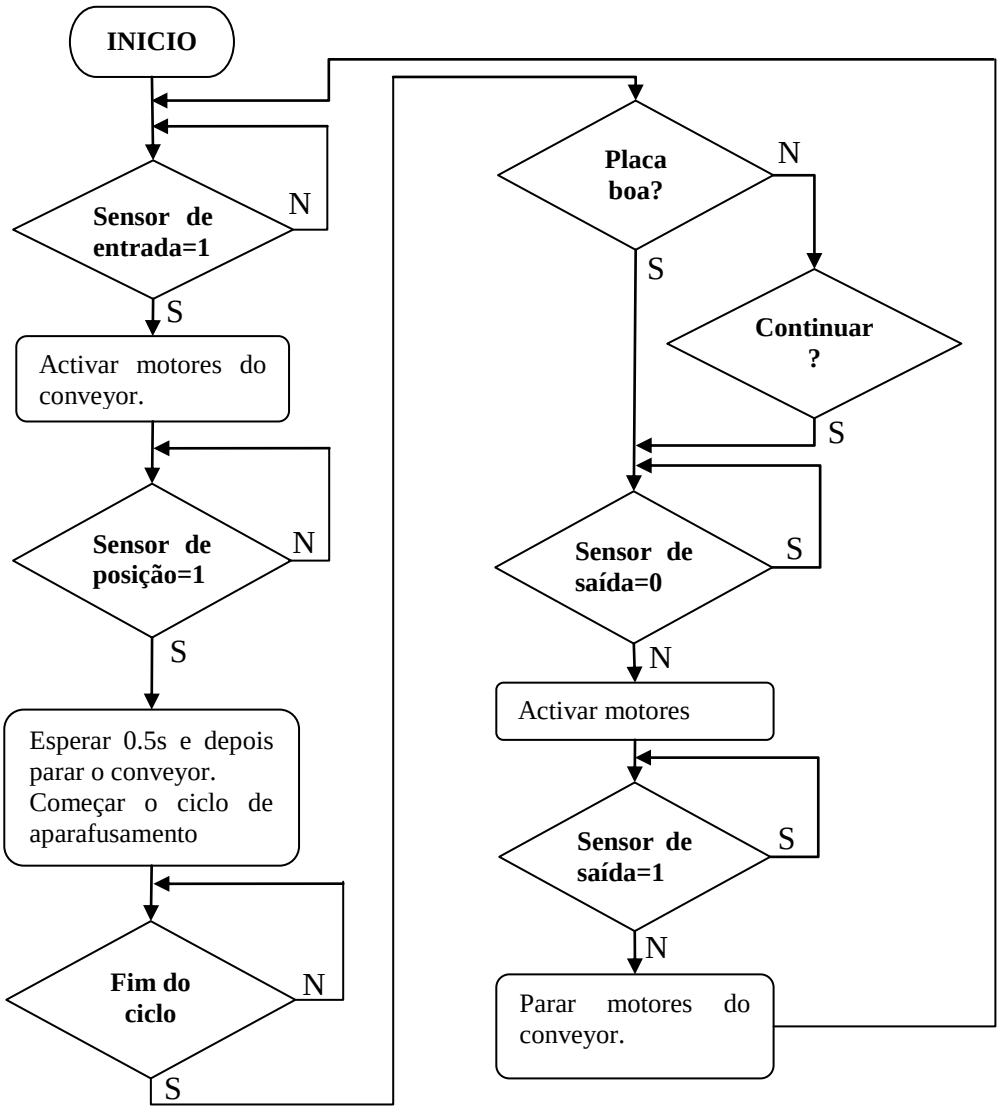



Figura 91 Diagrama de funcionamento da sequência de entrada e saída de placas

Após a entrada do chassi na célula existe um sensor óptico colocado antes do *stopper* que detecta a presença da placa e do chassi na zona de centragem, Figura 92.

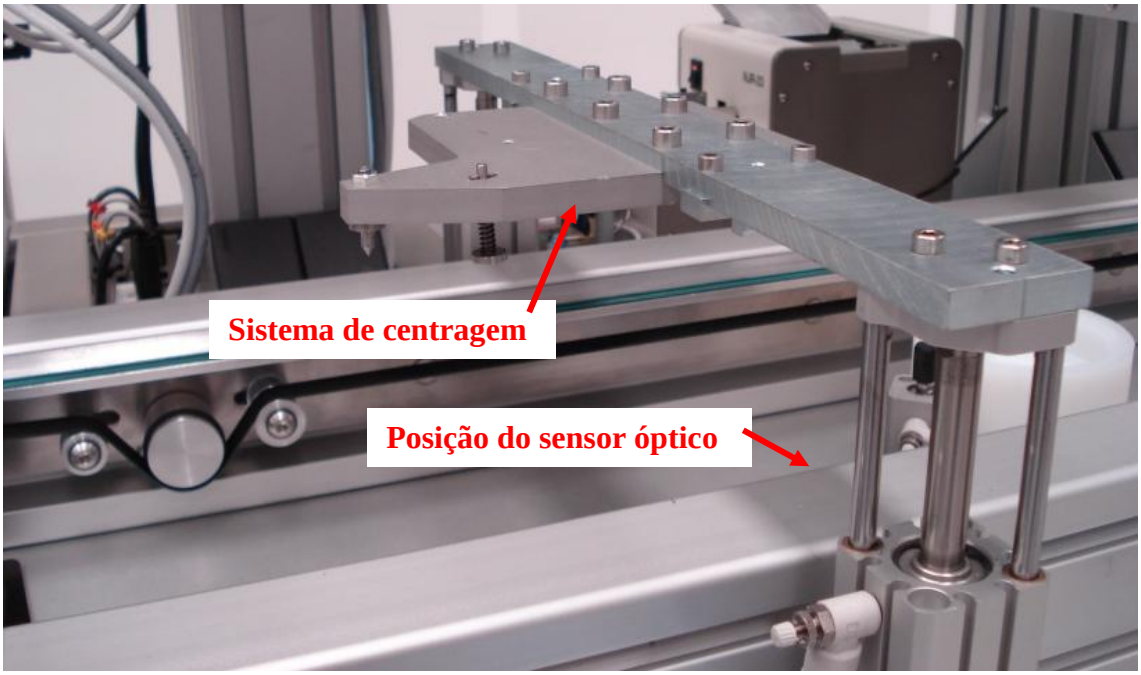



Figura 92 Sistema de centragem do PCB com o chassi

Este sensor é responsável por informar o sistema que existe um PCB para aparafusar, iniciando-se de seguida a centragem do PCB com o chassi. Para a centragem são utilizados dois actuadores pneumáticos que fazem descer uma barra de centragem com cavidades e pinos de centragem feitos à medida do PCB e do chassi, efectuando-se a centragem dos dois elementos.

Um dos actuadores pneumáticos possui sensores magnéticos que permitem verificar se a descida da barra de centragem foi efectuada com sucesso, caso este não seja actuado num tempo máximo de cinco segundos após a detecção de uma placa na zona de centragem, o sistema entra em emergência, indicando um possível problema na centragem da placa.

Esta detecção é de extrema importância, garantindo desta forma que o chassi está bem orientado para o aparafusamento do PCB ao chassi, evitando possíveis problemas mecânicos durante o aparafusamento, má orientação do chassi ou mesmo encravamento do sistema de centragem. Caso não existisse feedback poderia ocorrer uma colisão da aparafusadora com o sistema de centragem.

Através do diagrama seguinte, Figura 93, é possível visualizar as acções associadas à centragem descritas anteriormente.

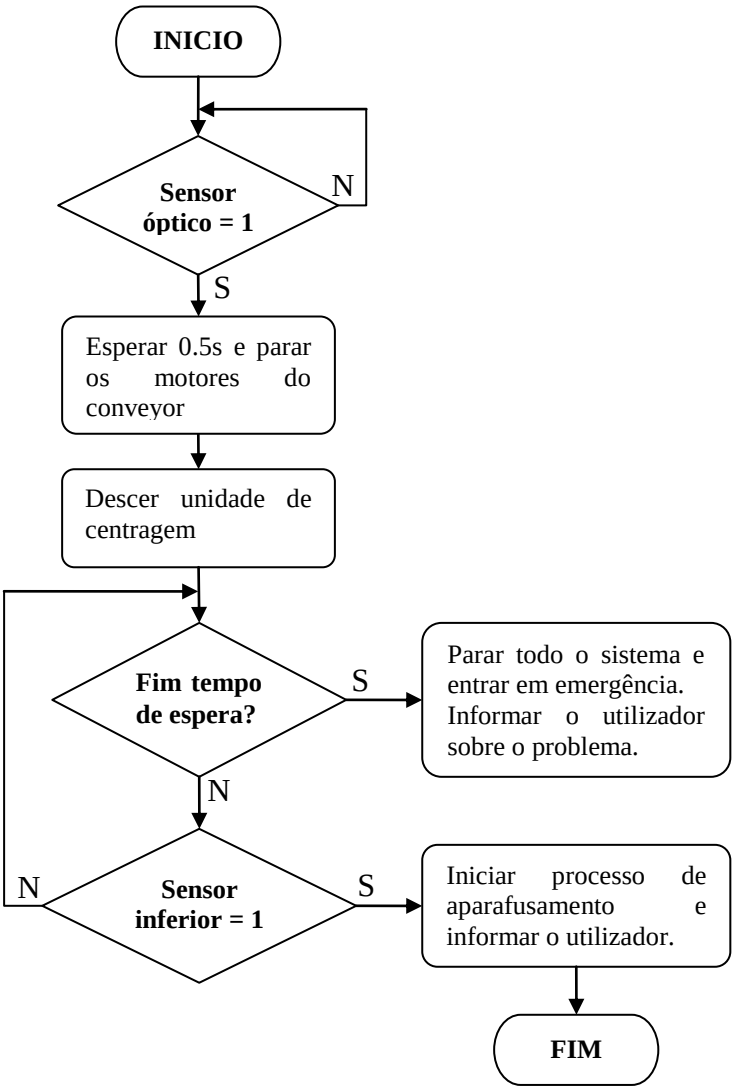



Figura 93 Diagrama de funcionamento da sequência de centragem do PCB

### 4.3.4 SISTEMA DE APARAFUSAMENTO

O sistema de aparafusamento é composto por uma aparafusadora eléctrica, um sistema de três eixos lineares com motores de passo e um dispensador de parafusos. O sistema tem como objectivo o aparafusamento de dezasseis parafusos num tempo máximo de noventa segundos com uma alimentação automática de parafusos.

Para a dispensa dos parafusos utilizou-se um Quicher, já referido anteriormente, que dispõe os parafusos numa guia vibratória e que por via de um sinal eléctrico de 24V retira um parafuso da guia e coloca-o num tubo com ligação à ponta da aparafusadora, onde o parafuso fica suspenso, após um sopro de ar, num bico especialmente concebido para o efeito, Figura 94.

Figura 94 Bico de suspensão do parafuso

Este bico permite segurar o parafuso na posição correcta para o aparafusamento, quando a aparafusadora desce devido à actuação na mesa pneumática, o *Bit* (ferramenta de aparafusamento acoplada à aparafusadora) empurra o parafuso para fora do bico que o segura, aparafusando-o no sitio desejado.

Todo este sistema de envio e aparafusamento do parafuso requer uma sequência optimizada de acções de forma a minimizar os tempos de espera, para optimizar este envio utilizou-se um sensor indutivo da marca *Di-Soric* modelo IR10, Figura 95.


Figura 95 Sensor de passagem do parafuso

Este sensor é responsável pela detecção da passagem do parafuso pelo tubo após este ter sido enviado pelo Quicher e efectuando-se o sopro de imediato, garantindo também que se consiga detectar problemas no envio do parafuso. O diagrama seguinte descreve a sequência de acções implementadas para o envio do parafuso, Figura 96.

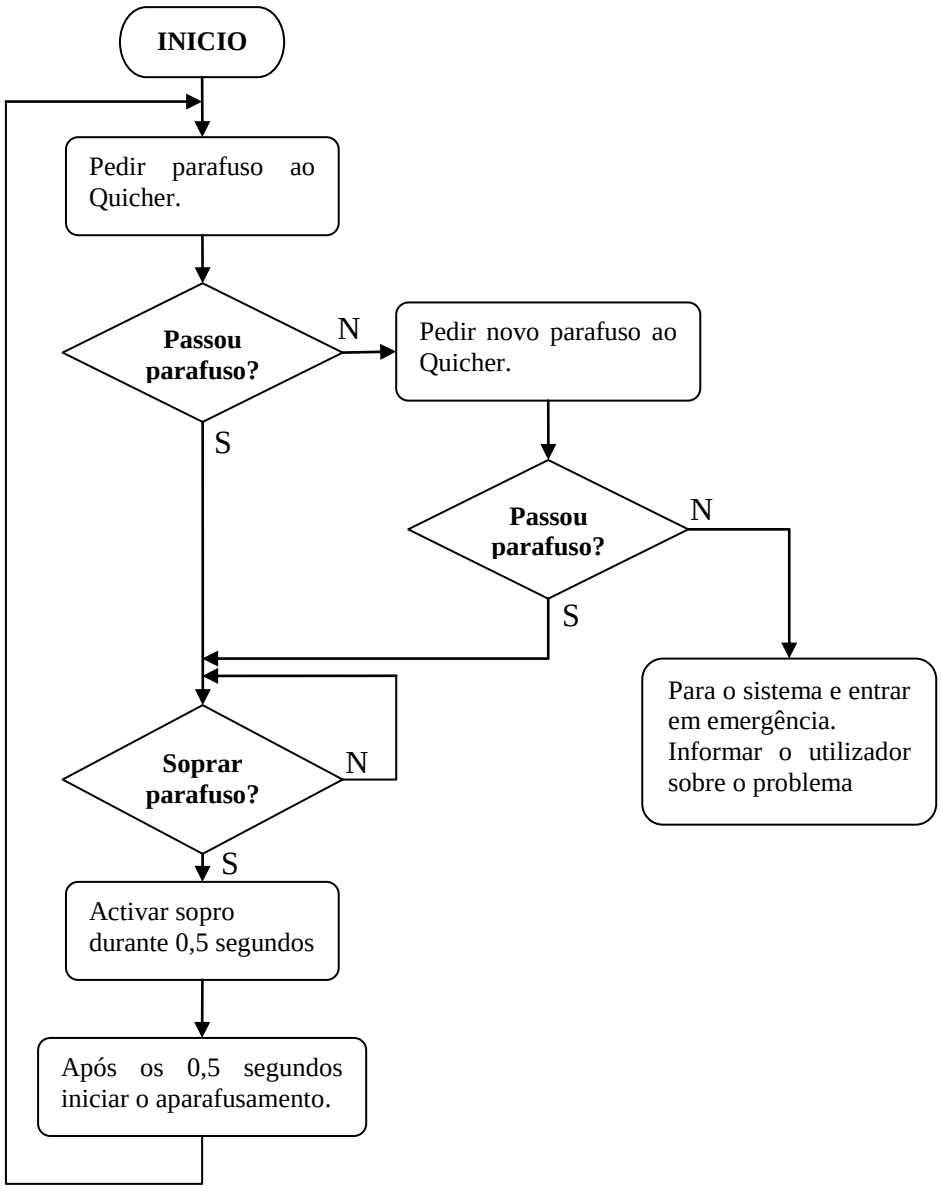



Figura 96 Diagrama de funcionamento da sequência de envio do parafuso

Desta forma o envio do parafuso está optimizado porque existe sempre um parafuso no tubo pronto a ser enviado para a aparafusadora e quando este é enviado é colocado de imediato outro no tubo. Este é apenas soprado quando a aparafusadora termina um aparafusamento, não deixando assim que a aparafusadora fique em espera devido à alimentação dos parafusos.

### 4.3.5 CONTROLADORES DE EIXOS AEB

Para movimentar a aparafusadora pelos diversos pontos de aparafusamento, utilizaram-se três eixos lineares com motores de passo controlados pelos controladores de eixos da marca AEB Robotics, referidos anteriormente. Uma grande vantagem destes controladores é ligação directa aos motores, porque além de controladores também são *drivers* de motores de passo.

Como se tratam de controladores independentes, foi necessário implementar um sistema de sequencionamento entre os três eixos, utilizando-se para isso as entradas e saídas disponíveis em cada um. Este método de sequencionamento requereu igualmente que a implementação em cada um dos controladores fosse mais complexa de forma a conseguir criar um sistema de posicionamento que cumprisse com os requisitos que cliente exigiu.

O diagrama seguinte, Figura 97, indica o sinal de arranque dos eixos (Start), o sinal para o inicio do aparafusamento (Aparafusa), o sinal do próximo parafuso (Continuar), o sinal de fim de ciclo, o sinal de erro no aparafusamento (Erro) e os sinais de sequencionamento entre eixos necessários usados para a implementação do sistema.

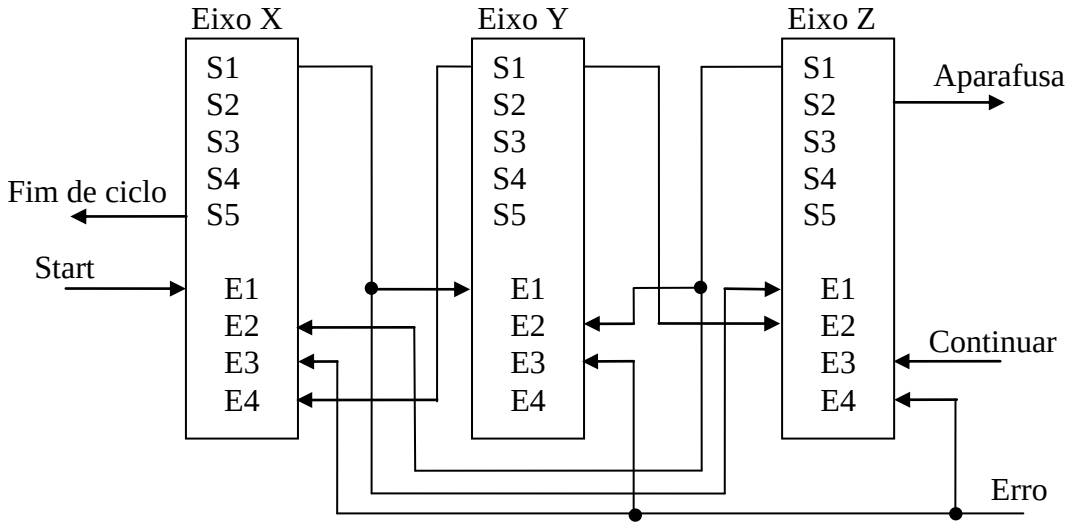



Figura 97 Diagrama de ligações entre os controladores AEB

Com a implementação das ligações descritas no diagrama anterior o sistema de eixos é capaz de coordenar os movimentos de cada um dos seus controladores, garantindo uma execução sequencial dos movimentos.

Contudo é necessário implementar a análise dos sinais em cada um dos controladores, para que estes consigam interpretar o seu significado e efectuar o posicionamento nos momentos correctos, assim, para se conseguir programar os controladores utiliza-se o software fornecido por estes, Figura 98.

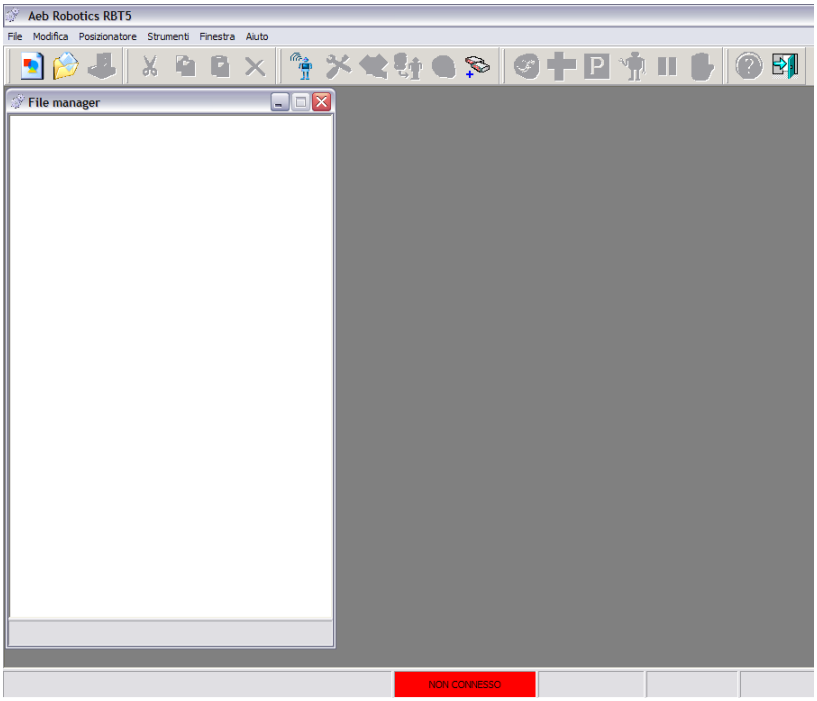



Figura 98 Ambiente de programação dos controladores AEB

Neste ambiente é possível programar e configurar os parâmetros mecânicos dos três eixos, não se descreverá o seu funcionamento porque a documentação é bastante completa e com diversos exemplos, esta pode ser consultada no site (http://www.aeb-robotics.com) e efectuar o download gratuito do software.

A sequência de movimentos é programada numa folha do tipo Excel indicando sequencialmente as acções a executar, escolhendo-se uma acção das seguintes disponíveis: mover para um ponto, modificar saídas, modificar entrada, saltar para a linha, esperar um tempo, enviar um pulso e *branch* (desvio na sequência de execução).

Com as acções disponibilizadas o seguinte bloco de código foi implementado nos eixos X e Y, Figura 99, de forma a posicionar os respectivos eixos na coordenada definida e aguardar pelo sinal do eixo Z para se movimentar para a coordenada seguinte.

| | Coord. (mm) | Speed (mm/s) | Acc. (mm/s²) | Dec. (mm/s²) | Outputs status | Time (ms) | Inputs | Ref. |
|---|---|---|---|---|---|---|---|---|
| 1 - COMMENT » | POINT | | | | | | | |
| 2 - POINT | 70 | 200 | 1000 | 1000 | | | | |
| 3 - COMMENT | EIXO ON POINT | | | | | | | |
| 4 - SET OUTPUTS | | | | | 1 | | | |
| 5 - COMMENT | WAIT FOR NEXT (UP) | | | | | | | |
| 6 - WAIT INPUTS | | | | | | | I2 | |
| 7 - COMMENT | CLEAR SIGNALS | | | | | | | |
| 8 - SET OUTPUTS | | | | | | | | |
| 9 - COMMENT | ERROR ?? | | | | | | | |
| 10 - BRANCH | | | | | | | - i3 | 1 » 191 |
| 11 - COMMENT | ERROR -> WAIT FOR NEXT (DOWN) | | | | | | | |
| 12 - WAIT INPUTS | | | | | | | i2 | |
| 13 - COMMENT | POINT FOR CLEAN | | | | | | | |
| 14 - POINT | 0 | 200 | 1000 | 1000 | | | | |
| 15 - COMMENT | EIXO ON POINT | | | | | | | |
| 16 - SET OUTPUTS | | | | | 1 | | | |
| 17 - COMMENT | WAIT FOR NEXT (UP) | | | | | | | |
| 18 - WAIT INPUTS | | | | | | | I2 | |
| 19 - COMMENT | CLEAR SIGNALS | | | | | | | |
| 20 - SET OUTPUTS | | | | | | | | |
| 21 - COMMENT | WAIT FOR NEXT (DOWN) | | | | | | | |
| 22 - WAIT INPUTS | | | | | | | i2 | |

Figura 99 Sequência de acções para o eixo X e Y por cada coordenada

Descrevendo a imagem anterior, o primeiro passo indica a coordenada em milímetros para o qual o eixo em questão se deve deslocar, após o movimento informa que se encontra na posição e aguarda por um sinal que permita que este continue. Depois analisa o sinal de entrada de erro e decide se deve continuar para a próxima coordenada de aparafusamento ou para a zona de limpeza do bico da aparafusadora, caso não ocorra erro a execução do programa salta da linha 10 para a linha 23, que não está presente na imagem anterior, mas é um novo bloco de código idêntico ao descrito.

No caso de se deslocar para a zona de limpeza a aparafusadora apenas se desloca para baixo por meio da mesa pneumática, limpando o bico de possíveis parafusos encravados, depositando-os num depósito para o efeito.

O bloco de código indicado é repetido quantas vezes for necessário dependendo dos pontos de aparafusamento desejados.

Para o eixo Z o bloco de código segue o mesmo esquema, no entanto este possui mais algumas acções, porque é este que sinaliza o sistema para começar o aparafusamento e dá o sinal de avanço para os eixos X e Y. Na Figura 100 é possível verificar o bloco de código implementado no eixo Z.

| | Coord. (mm) | Speed (mm/s) | Acc. (mm/s²) | Dec. (mm/s²) | Outputs status | Time (ms) | Inputs | Ref. |
|---|---|---|---|---|---|---|---|---|
| 1 - COMMENT » | WAIT FOR XY | | | | | | | |
| 2 - WAIT INPUTS | | | | | | | I1 I2 | |
| 3 - COMMENT | POINT | | | | | | | |
| 4 - POINT | 169 | 100 | 1000 | 1000 | | | | |
| 5 - COMMENT | INIT APARAF | | | | | | | |
| 6 - PULSED OUTPUT | | | | | 2 | 100 | | |
| 7 - WAIT INPUTS | | | | | | | I3 | |
| 8 - BRANCH | | | | | | | - i4 | 1 » 308 |
| 9 - WAIT INPUTS | | | | | | | i3 | |
| 10 - COMMENT | CLEANING | | | | | | | |
| 11 - POINT | 140 | 100 | 1000 | 1000 | | | | |
| 12 - SET OUTPUTS | | | | | 1 | | | |
| 13 - DELAY | | | | | | 100 | | |
| 14 - SET OUTPUTS | | | | | | | | |
| 15 - WAIT INPUTS | | | | | | | I1 I2 | |
| 16 - POINT | 169 | 100 | 1000 | 1000 | | | | |
| 17 - PULSED OUTPUT | | | | | 2 | 100 | | |
| 18 - WAIT INPUTS | | | | | | | I3 | |
| 19 - WAIT INPUTS | | | | | | | i3 | |
| 20 - POINT | 140 | 100 | 1000 | 1000 | | | | |
| 21 - SET OUTPUTS | | | | | 1 | | | |
| 22 - DELAY | | | | | | 100 | | |
| 23 - SET OUTPUTS | | | | | | | | |

Figura 100 Sequência de acções para o eixo Z por cada coordenada

No bloco de código presente na figura anterior é possível verificar que este varia entre as posições 169 mm e 140 mm, estas são as posições de aparafusamento e de deslocação do eixo Z. Para cada movimento dos eixos X e Y o eixo Z efectua dois movimentos, um de descida e inicio de aparafusamento e outro de subida e sinalização de avanço para os outros eixos. O diagrama seguinte, Figura 101, exemplifica as acções efectuadas para posicionamento da aparafusadora.

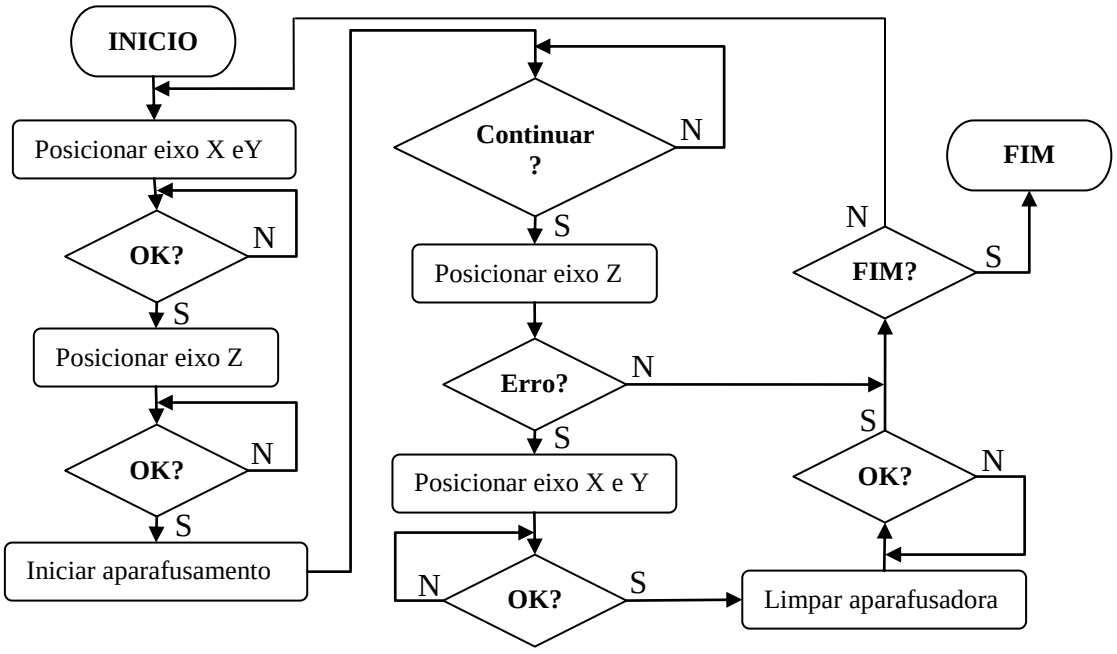



Figura 101 Diagrama de funcionamento da sequência de posicionamento e aparafusamento

### 4.3.6 DIAGRAMA DE LIGAÇÕES

A figura seguinte mostra um diagrama simplificado das ligações entre os diversos dispositivos colocados na célula de aparafusamento automático.

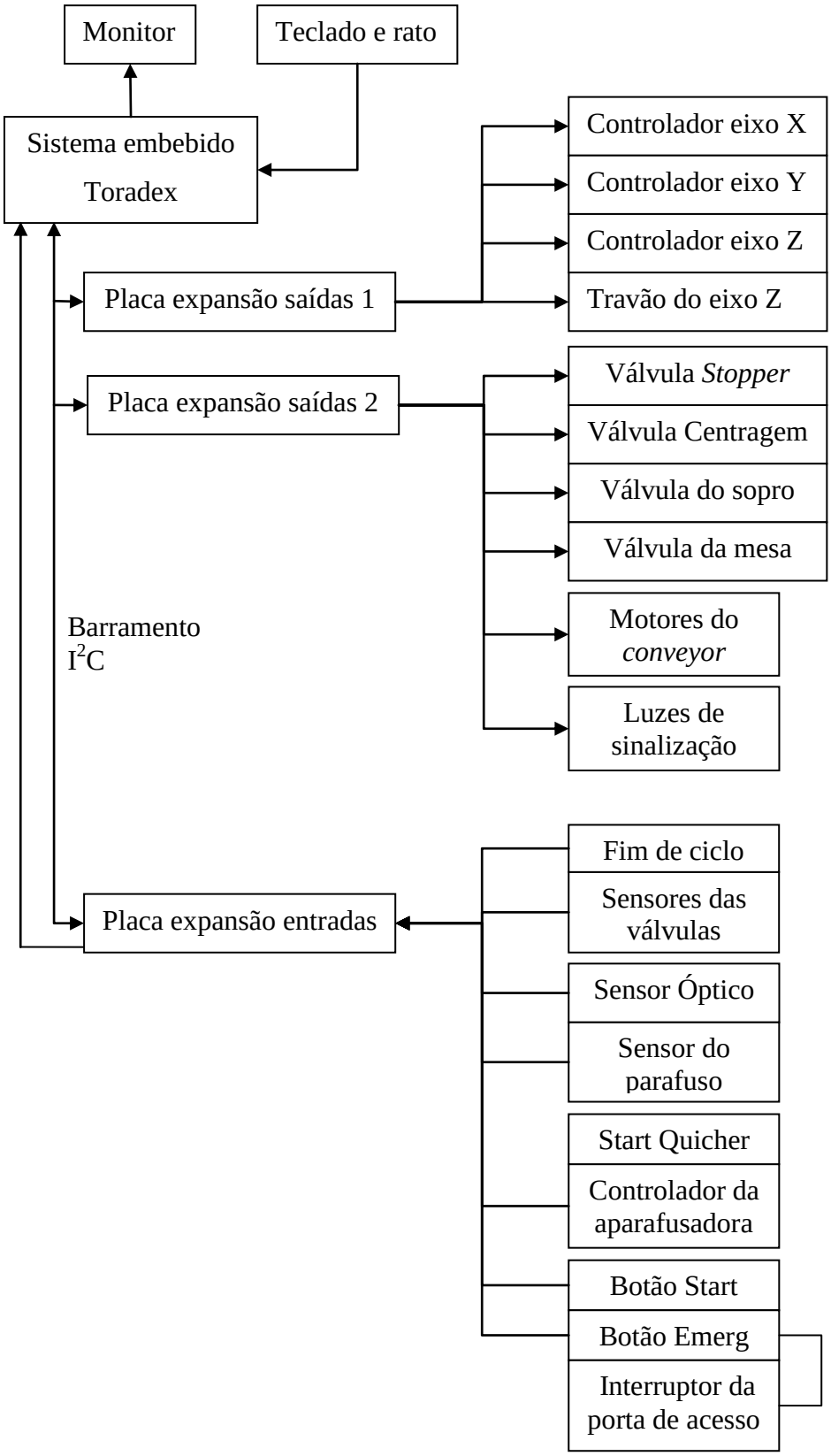



Figura 102 Diagrama simplificado das ligações entre os diversos dispositivos

O diagrama anterior indica o sentido dos dados ou sinais entre os diversos dispositivos do sistema, possui como dispositivo central o sistema embebido Toradex, este está conectado a um monitor, um rato e um teclado permitindo a interacção com o utilizador.

A iteração do sistema com os diversos sensores e actuadores é efectuada através das placas de expansão ligadas ao sistema embebido através do barramento de $I^2C$, existindo duas placas de saída e uma de entrada. A placa de entradas possuiu um sinal de interrupção que é ligado às entradas da placa Orchid, permitindo desta forma detectar a ocorrência de uma interrupção provocada pela placa de expansão de entradas quando ocorre uma mudança de estado nas suas entradas.

No caso botão de emergência e da protecção da porta de acesso, estes estão em série e com contactos normalmente fechados colocando 24V na entrada da placa, retirando os 24V em caso de emergência.

## 4.4 CONCLUSÕES

Durante o desenvolvimento da célula ocorreram diversas modificações de forma a conseguir-se cumprir os objectivos propostos, a utilização do sistema pneumático é sem dúvida uma mais-valia para a implementação, tornando o sistema menos complexo de controlar e ao mesmo tempo dando garantias do seu funcionamento.

A utilização de um dispensador de parafusos com guia vibratória com um sistema mecânico que permite a colocação de um parafuso sempre que solicitado e com um sopro de ar facilitou a colocação dos parafusos no bico da aparafusadora, apesar de este requerer um ajuste cuidado para que os parafusos não encravem na saída.

Contudo a utilização de controladores independentes para cada eixo causou alguns problemas para efectuar o sequencionamento dos movimentos, tornando o sistema de posicionamento pouco flexível e um pouco complexo de programar, retirando esta dificuldade o sistema final cumpria os requisitos impostos pelo cliente.

# CAPÍTULO 5

# 5 SOFTWARE DESENVOLVIDO

Além da implementação mecânica do sistema foi necessário implementar um ambiente gráfico requerido pelo cliente para facilitar a utilização da célula de aparafusamento. Para a implementação do ambiente gráfico foi utilizado o ambiente de desenvolvimento da Microsoft, o Visual Studio 2008 com os *plugins* da Framework Qt que permitem executar o Qt Designer, Figura 103, tornando o desenvolvimento de aplicações em Qt mais rápidas e com um design mais atractivo.

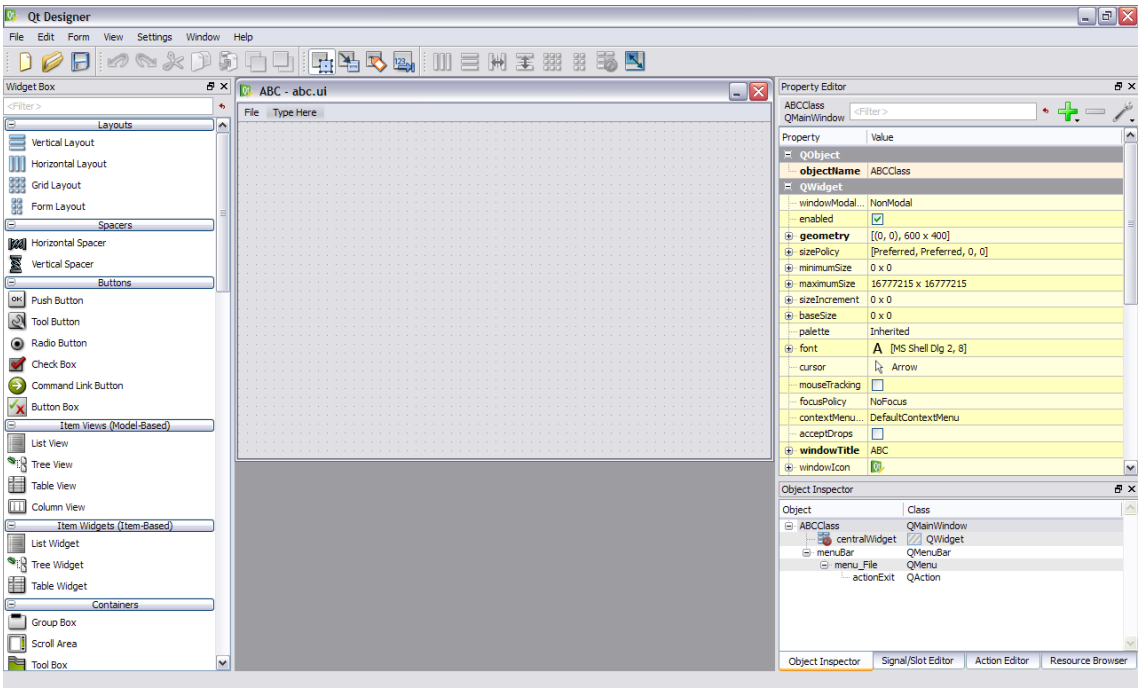



Figura 103 Ambiente gráfico da ferramenta desenvolvimento da Qt Software, o Qt Designer

Como requisitos do cliente era exigido que o programa fosse capaz de seleccionar o programa de aparafusamento, indicar o estado actual do aparafusamento em curso, indicar quais os parafusos bem ou mal aparafusados e permitir guardar em ficheiros de texto as estatísticas de um dia de trabalho para que estes possam ser consultadas posteriormente.

## 5.1 METODOLOGIA DE IMPLEMENTAÇÃO

A implementação da aplicação no sistema embebido da Toradex deve ser capaz de efectuar o controlo da célula de aparafusamento bem como disponibilizar os dados do processo num monitor sem que a aplicação deixe de responder em qualquer ocasião.

O sistema não deve ficar dependente de nenhuma das suas funções, o que poderia causar com que o sistema deixa-se de responder ficando a célula de trabalho sem controlo. Desta forma foram definidos dois níveis na aplicação, o primeiro nível e o mais importante efectua o controlo dos dispositivos da célula de aparafusamento e um segundo nível referente à visualização dos dados no monitor.

O primeiro nível tem prioridade máxima sendo executado sempre ocorrem interrupções nas placas de expansão de entradas ou é necessário enviar dados para a placa de expansão de saídas. O segundo nível é executado quando o sistema está livre, Figura 104.

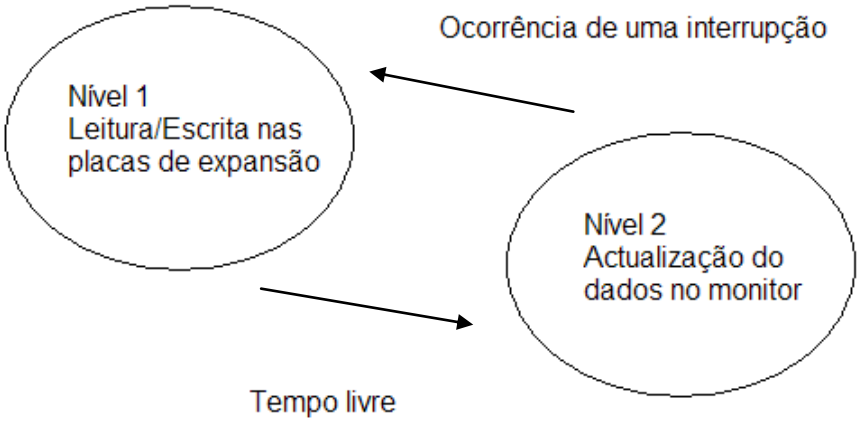



Figura 104 Metodologia de implementação da aplicação

Para a implementação da aplicação na Framework Qt, que é baseada em C++ e sendo esta uma linguagem orientada a objectos (OO) é necessária a implementação do sistema de controlo em uma ou mais classes, em que cada classe é responsável por uma parte da implementação da aplicação, desta forma implementou-se diversas classes, uma para a inicialização da aplicação, uma para o ambiente visual, uma para o controlo do processo, entre outras.

No diagrama seguinte é possível visualizar as diversas classes existentes e como estas interagem entre si, Figura 105.

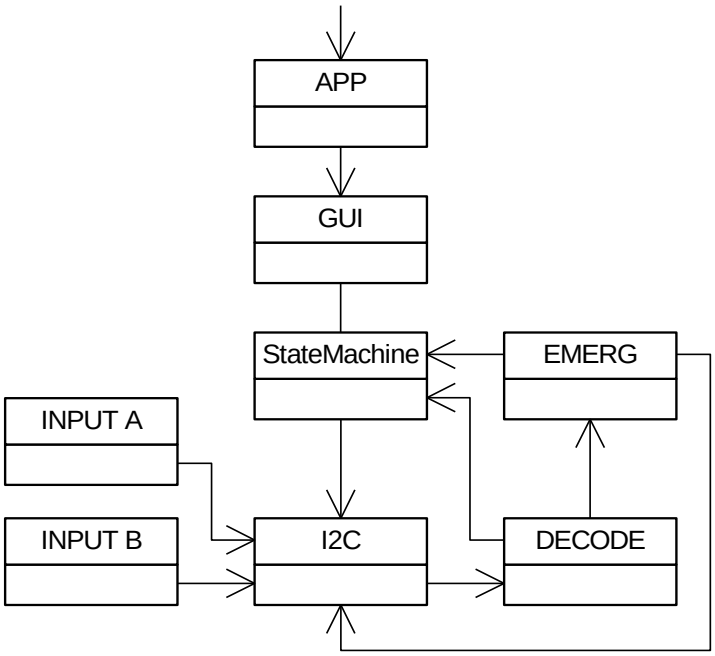



Figura 105 Diagrama de classe utilizada na aplicação desenvolvida

Descrevendo-se as classes da imagem anterior, tem como primeira classe a classe *APP* que inicializa a aplicação e plataforma Qt, de seguida é inicializada a classe *GUI* que tem como objectivo a implementação do ambiente da aplicação e de todas as outras classes, bem como os mecanismos de comunicação e sinalização entre classes.

As classes *INPUT A* e *INPUT B* referem-se às threads implementadas para a inicialização e monitorização das interrupções referentes à placa de expansão de entradas, a classe *I2C* é responsável pelo envio e recepção de dados pelo barramento de $I^2C$, sendo executada dentro de uma *thread* e possuindo registos que sinalizam a necessidade de efectuar uma leitura ou escrita no caso de esta estar ocupada, a classe *DECODE* analisa os dados recebidos pela classe *I2C* interpretando apenas os dados relevantes (sinais de emergência), a classe *EMERG* é uma classe de emergência que obtêm o controlo temporário do sistema de forma a implementar a rotina de emergência, a classe *StateMachine* é responsável por determinar as acções a executar em cada estado da máquina de aparafusamento.

Um aspecto importante é a prioridade das *threads* implementadas, que no arranque da aplicação são inicializadas com o mesmo nível de prioridade para que estas partilhem igualmente o processador, no entanto dependendo dos sinais obtidos pelas placas de expansão de entradas a classe *EMERG* que possuí uma *thread* pode obter controlo total sobre o processador para que esta implemente a rotina de emergência sem ser perturbada, reduzido ao mínimo o tempo de atraso entre a ocorrência da emergência e a acção a ser executada.

Para implementar a comunicação entre os diferentes objectos, utilizou-se um mecanismo disponibilizado na Framework Qt, o denominado *"Signals & Slots"*. Este mecanismo permite implementar funções em cada classe que agem como receptores ou sinalizadores de acções para outras classes, ou seja, um sinalizador (*signal*) de uma classe ao estar ligado a um receptor *(slot)* de outra classe permite que as duas classes interajam entre si, enviando dados como argumentos das funções.

O mecanismo *"Signals & Slots"* é considerado seguro por ser do tipo *type safe*, ou seja, os argumentos de uma função do tipo *signal* tem de ser coincidentes com os argumentos da função do tipo *slot*, reduzindo as hipóteses de ocorrer uma interligação não desejada entre objectos diferentes, contudo, se os argumentos das funções forem iguais não é possível detectar o erro. Verifique-se o seguinte exemplo:

```
public slots:
    void setValue(int value);
signals:
    void valueChanged(int newValue);

QObject::connect(&a, SIGNAL(valueChanged(int)), &b, SLOT(setValue(int)));
```

As duas funções iniciais do exemplo anterior implementam, no primeiro caso o receptor e no segundo o sinalizador, bastando para isso adicionar antes da função a indicação se é do tipo *slots* ou *signals*.

Para estabelecer a comunicação entre os dois objectos é necessário chamar a função *QObject::connect* em que os argumentos são os próprios objectos e as funções *slot* e *signal*. Este mecanismo pode ser implementado em qualquer classe, para isso deve-se incluir a livraria referente à comunicação (QObject) e declará-la no inicio da classe como:

```
#include <QObject>
class ABC
{       Q_OBJECT
        public slots:
                ...
};
```

## 5.2 ALGORITMOS DE CONTROLO

Para implementar o controlo da célula foi desenvolvida uma máquina de estados que permite actuar de acordo com o implementado em cada estado, o funcionamento desta é o descrito no diagrama seguinte, Figura 106:

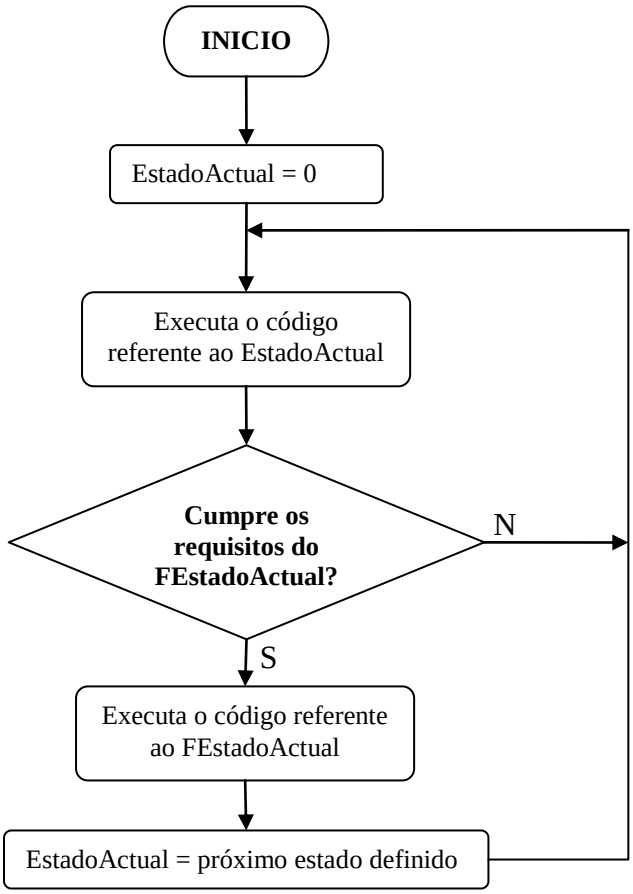



Figura 106 Diagrama de funcionamento da máquina de estados

No inicio a máquina de estado foi definida com o estado inicial como zero executando o código referente ao estado actual, analisando de seguida se as condições de transição ("FEstadoActual") para o próximo estado são cumpridas, se não forem cumpridas vai executar o código referente ao estado actual, caso sejam cumpridas as condições de transição este executa o código de transição e actualiza o estado actual com o próximo estado.

Esta máquina de estado é facilmente implementada através da condição *switch-case* permitindo com o índice do estado actual (EstadoActual), executar o código referente a cada estado da máquina e analisando-se também as condições de transição de estado, como se pode verificar no exemplo seguinte:

```
void EXEC_Programa::run()
{  while(1)
   {        switch(m_STATE)
            {
            case 0:   Estado0();
                      Estado0F();
                      break;
            case 1:   Estado1();
                      Estado1F();
                      break;

                     ...

                  }
         Sleep(1);
   }
}
```

Em cada *case* existem duas funções, uma para executar as acções desse estado e outra para analisar as condições de transição para o próximo estado e é esta segunda função que tem a capacidade de modificar o índice que indica o estado actual.

Esta máquina de estado é executada dentro de uma *thread* com o mesmo nível de prioridade que as *threads* de interrupção, para que possam também ser executadas em simultâneo com a máquina de estado. Todas as outras *threads* existentes apenas são executadas em algumas ocasiões como o envio/recepção de dados, análise dos dados e em caso de emergência da máquina.

No final é colocado um tempo de paragem, o *Sleep(1)*, para que as *threads* de mais baixa prioridade possam ser executadas garantindo um bom funcionamento de todo o sistema.

Para se efectuar a comunicação com as placas de expansão é utilizada a classe I2C, esta classe permite escrever ou ler das placas de expansão mediante as funções *SetIOPin(int device, int pin, bool value)* e *GetIOPin(int device, int op).* A primeira função envia o valor de um pino para uma placa de expansão de saídas e a segunda lê o valor do registo das entradas. No caso de ocorrer uma escrita quando o barramento de $I^2C$ está em uso esta ficará num *buffer* a aguardar até possa ser enviada, no caso de uma leitura a *thread* aguarda pelo fim do envio actual e seguidamente efectua a leitura continuando posteriormente os envios dos dados restantes. O diagrama seguinte exemplifica o algoritmo implementado, Figura 107:

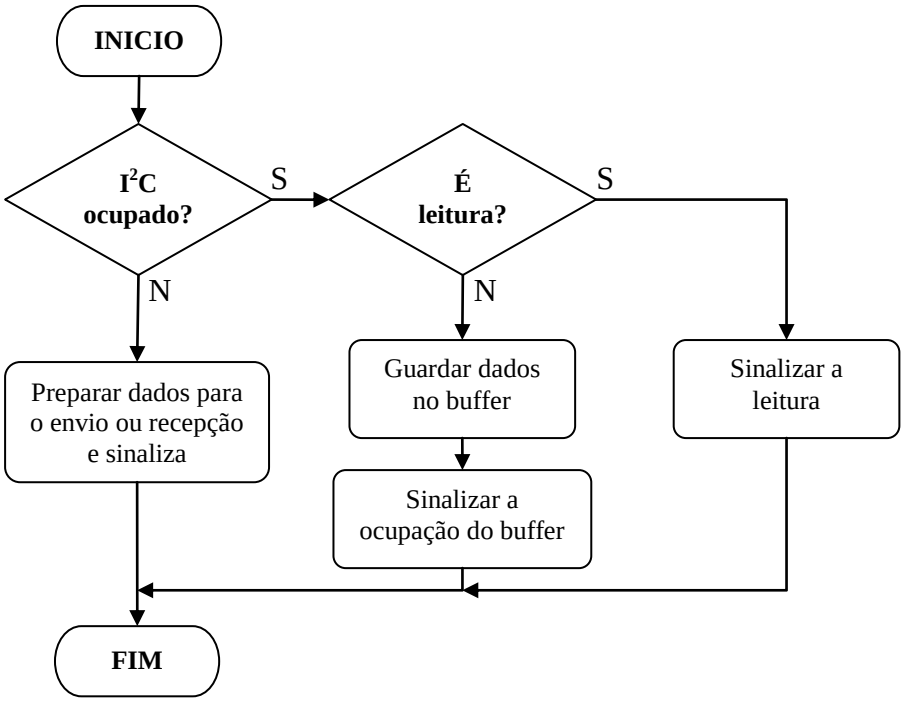



Figura 107 Algoritmo de preparação dos dados para escrita ou leitura por $I^2C$

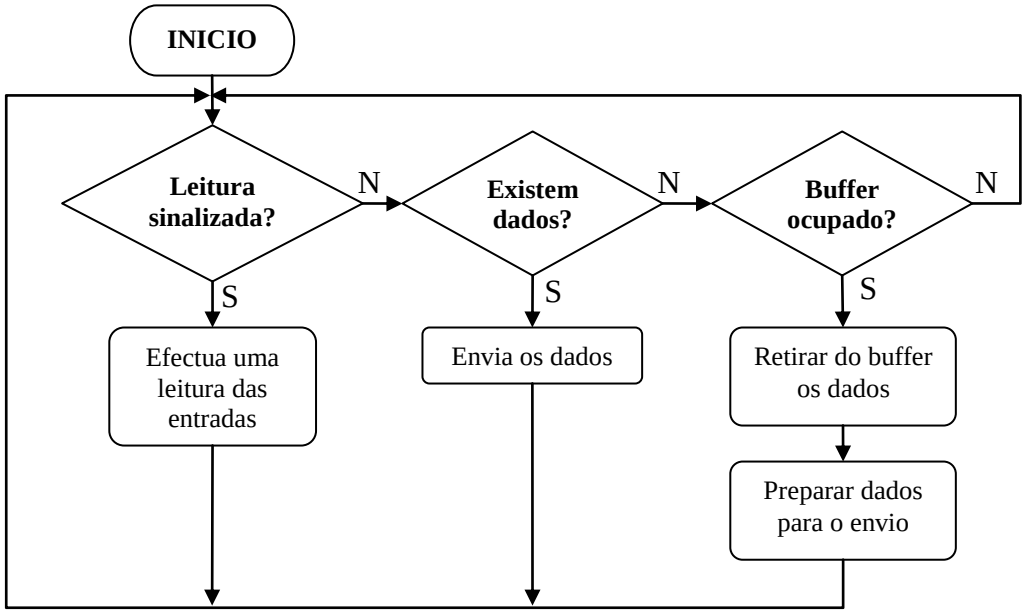



Figura 108 Algoritmo de execução da thread de $I^2C$

O diagrama da Figura 107 representa as acções antes de enviar os dados pelo barramento de $I^2C$, onde se verifica se o barramento está ocupado e que tipo de dados se pretende enviar. No diagrama da Figura 108 está representado o funcionamento da *thread* existente na classe I2C responsável pelo envio ou recepção dos dados pelo barramento de $I^2C$.

## 5.3 AMBIENTE GRÁFICO PARA MONITORIZAÇÃO

Com a máquina de estados desenvolvida é necessário trabalhar igualmente o ambiente gráfico da aplicação para cumprir com alguns requisitos impostos, um interface amigável é um requisito muito importante porque o utilizador deve compreender facilmente todos os botões e parâmetros presentes no monitor.

Tendo em conta um dos requisitos no qual se pretende uma fácil escolha do programa de aparafusamento, foi implementado o seguinte modelo, Figura 109.

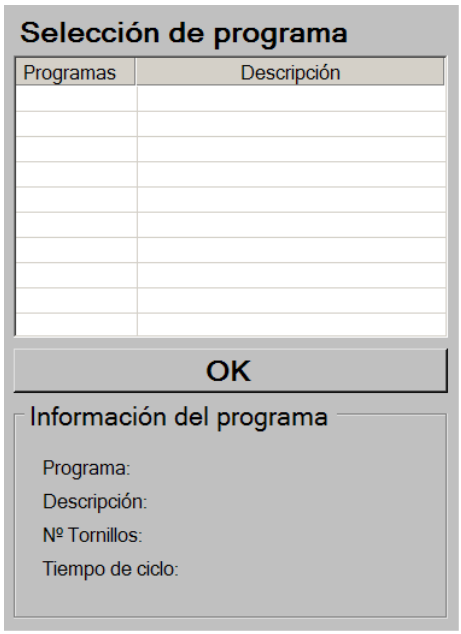



Figura 109 Selecção do programa de aparafusamento

Esta parte da aplicação que permite a escolha do programa, é constituída por um objecto do tipo *QGroupBox* que permite agrupar os dados referentes ao programa seleccionado, um objecto do tipo *QPushButton* que é o botão posicionado a meio da figura anterior que permite confirmar a selecção do programa, um objecto do tipo *QTableView* que permite listar os diversos programas existentes e diversos objectos do tipo *QLabel* que são usados para definir os vários textos presentes na figura.

A escolha de um programa é efectuada seleccionando primeiro o programa da lista disponibilizada, carregando de seguida no botão "OK" e observando se os dados referentes à aplicação escolhida aparecerem na zona de informação, só então o programa foi correctamente seleccionado.

Outro requisito imposto é o acompanhamento do processo de aparafusamento de uma forma gráfica e de fácil compreensão, desta forma o seguinte modelo foi implementado, Figura 110.

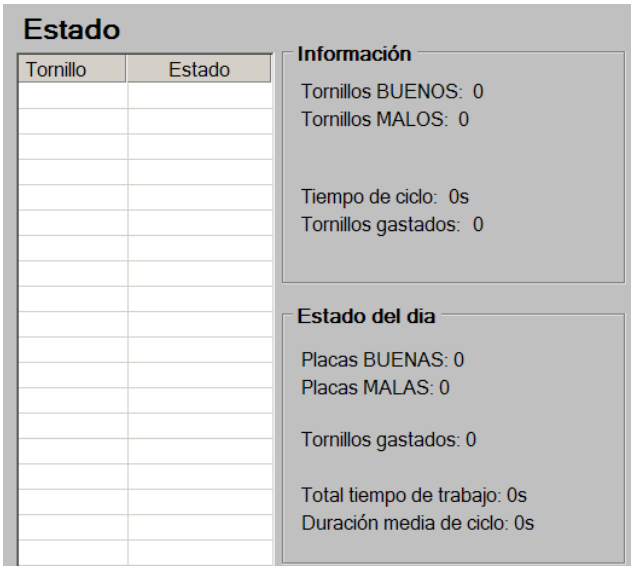



Figura 110 Indicação do estado do aparafusamento e dados estatísticos

Nesta parte da aplicação a monitorização do estado do aparafusamento de cada parafuso é dado pelo objecto do tipo *QTableView* que permite listar o número de parafusos que vão ser aparafusados, colocando em cada linha a cor verde se este for bem aparafusado ou a cor vermelha se mal aparafusado, acompanhando o progresso do aparafusamento.

Além de acompanhar o estado do processo, no final de cada ciclo de aparafusamento os objectos *QGroupBox "Información"* e *"Estado del dia"* são actualizados com a informação do aparafusamento actual e do estado do aparafusamento total diário, respectivamente.

Os dados referentes ao dia são igualmente guardados em ficheiro de texto com o nome do programa de aparafusamento e com a data (dia, mês e ano), caso se mude de programa no mesmo dia os dados no monitor serão apagados e caso exista já um ficheiro desse dia do programa seleccionado, estes serão recuperados e continuará a partir dos dados recolhidos do ficheiro.

Na mudança do dia, ou seja, após a passagem da meia-noite a aplicação guardará os últimos dados para ficheiro e apagará os dados do monitor criando um novo ficheiro para o dia de trabalho, isto é importante porque em muitos casos a máquina de aparafusamento poderá ficar vários dias ligada.

Para a configuração dos programas de aparafusamento é necessário programar os controladores dos eixos com o programa fornecido pela marca deste, contudo é necessário introduzir manualmente a referência dos programas na aplicação. Para facilitar esta configuração foi desenvolvida a seguinte janela, Figura 111.

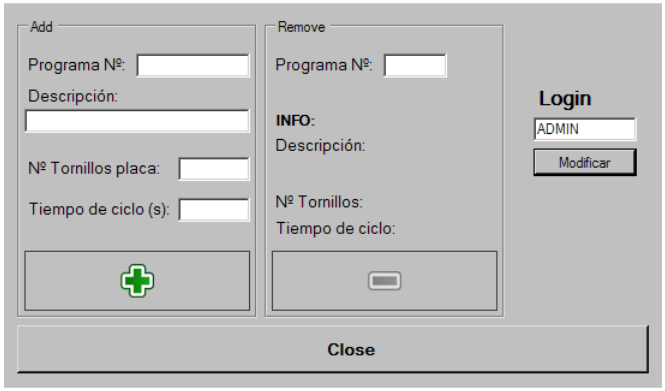



Figura 111 Configuração dos programas de aparafusamento

Esta janela de configuração da aplicação permite adicionar ou remover programas de aparafusamento sendo criada a partir de um objecto do tipo *QWidget*. É constituída também por objectos do tipo *QLineEdit* que permitem a introdução de texto, objectos do tipo *QPushButton* que são os botões visíveis na figura anterior, além de outros como o *QGroupBox* e o *QLabel*.

Quando se inicia a configuração dos programas é necessário introduzir uma palavra-chave já definida anteriormente e que permitirá ter acesso à edição dos programas, podendo depois remover ou adicioná-los. Para adicionar é necessário introduzir obrigatoriamente o número do programa correspondente ao existente nos controladores dos eixos e o número de parafusos associado, os outros parâmetros são opcionais. Para remover basta colocar o número de um programa existente na lista dos programas e confirmar a sua remoção.

A actualização nos ficheiros é efectuada no momento da adição ou da remoção de cada programa, a actualização da lista de programas no ecrã principal é efectuada automaticamente após o fecho da janela de configuração através do mecanismo de comunicação "*signal & slots*" referido anteriormente, sinalizando o fecho da janela e executando a função de leitura do ficheiro com os novos programas.

Com as diferentes partes desenvolvidas de forma a alcançar os requisitos do cliente a aplicação final ficou com o seguinte ambiente gráfico, Figura 112.

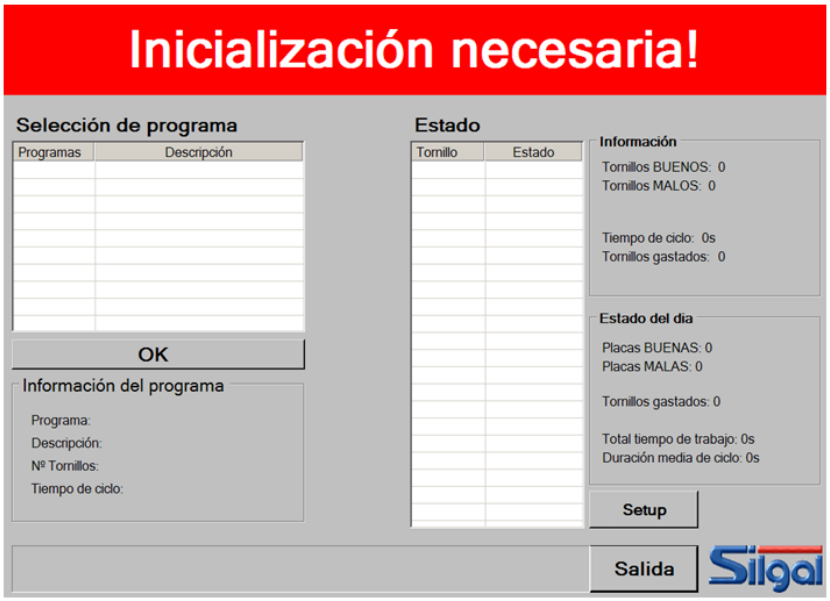



Figura 112 Ambiente gráfico implementado na célula de aparafusamento

## 5.4 CONCLUSÕES

Durante o desenvolvimento da aplicação gráfica para o controlo e monitorização da célula industrial surgiram diversos problemas referentes ao tempo de resposta da aplicação, esta necessitou se ser desenvolvida de forma a não privar os processos de menor prioridade do uso do processador que ocorriam inicialmente, tornando o ambiente gráfico lento e de difícil utilização.

O uso de uma máquina de estados para a implementação do controlo da máquina revelou-se como a melhor solução, contudo poderá revelar-se limitada e de difícil implementação em aplicações mais complexas. Tratando-se de uma máquina com uma execução sequencial de acções a máquina de estados apenas necessitava de separar cada estado em dois, um de execução e outro de análise da transição para o próximo estado, isto para permitir uma análise do código mais simples.

A utilização da Framework Qt é uma mais-valia na implementação de qualquer aplicação gráfica devido aos diversos controlos já implementados e ao mecanismo de comunicação *"Signals & Slots"* que facilitam a implementação de acções entre os objectos das diferentes classes, contudo é necessário respeitar a implementação descrita pela Framework Qt, o que por vezes dificultou e elevou o tempo de implementação das classes da aplicação.

# CAPÍTULO 6

# 6 CONCLUSÕES E TRABALHO FUTURO

Neste capítulo descrever-se-á as conclusões e elações retiradas do projecto desenvolvido, mais especificamente o estudo do sistema embebido da Toradex e da sua aplicação no controlo e monitorização de uma célula automática de aparafusamento.

## 6.1 CONCLUSÕES

A adopção dos sistemas embebidos para aplicações de controlo e monitorização de dados é uma mais-valia para a implementação de sistemas que se querem compactos ou que necessitem de um tempo de resposta curto, ideais para a implementação de uma só aplicação. O sistema embebido em análise é da marca Toradex, escolhido devido às suas características e aos acessórios de expansão disponíveis, contando igualmente com o facto de este ter como sistema operativo o Windows CE.

A utilização do Windows CE como SO permite não só uma utilização simplificada dos recursos do processador PXA270 (com as livrarias disponibilizadas pela Toradex) como também uma fácil ambientação aos programas de desenvolvimento existentes para essa plataforma (Visual Studio 2008).

Com o objectivo de estudar o sistema embebido para a sua implementação no controlo e monitorização de dados efectuou-se uma análise do tipo de código (*Managed* e *Unmanaged*), retirando-se como conclusão que o código *unmanaged* seria a melhor opção para a implementação de uma aplicação. Apesar de existirem opiniões favoráveis ao uso de código do tipo *managed* em aplicações embebidas, o seu uso continua a ser alvo de bastantes críticas e dúvidas.

Assim, como o uso do código do tipo *unmanaged* (C++) se revelou como o que produziu melhores resultados nos testes efectuados iniciou-se uma análise de diferentes tipos de frameworks que permitiriam desenvolver aplicações gráficas mais facilmente, reduzindo o tempo final de desenvolvimento de uma aplicação em C++.

A framework seleccionada foi a da Qt Software (Nokia), a Framework Qt que possui uma versão compatível com o Windows CE e inúmeros controlos já desenvolvidos e comuns às aplicações existentes, com esta framework foi possível desenvolver aplicações gráficas com qualidade garantido interfaces simples para o utilizador.

Com o uso do mecanismo *"Signals & Slots"* a tarefa de interligar as acções dos diversos objectos desenvolvidos tornou-se relativamente simples, contudo o uso deste mecanismo requer uma análise cuidada dos métodos a implementar para efectuar a comunicação, pois estes são do tipo *type safe*, garantindo igualmente ao mesmo tempo alguma segurança na ligação entre o emissor e o receptor.

Para a interligação do sistema embebido com uma célula industrial foi necessário o desenvolvimento de placa de expansão de E/S para que os níveis lógicos da célula (0V a 24V) fossem compatíveis com os níveis lógicos do sistema embebido (0V a 3,3V). Como o sistema embebido possui um módulo de $I^2C$ este foi utilizado para que fosse possível expandir o número de E/S suportadas pelo sistema para 128, isolando também as placas de expansão do sistema embebido. O uso do barramento de $I^2C$ limita a utilização até 8 placas de expansão e a distância do próprio barramento.

O desenvolvimento da célula de aparafusamento automático de PCB's serviu para testar as potencialidades do sistema embebido com as placas de expansão, requerendo a implementação de uma máquina de estados capaz de executar as acções de forma sequencial para garantir o aparafusamento do PCB ao chassi. A implementação da aplicação foi dividida em duas partes, uma mais prioritária onde se executava o código referente ao controlo da célula e outra menos prioritária onde se realizava a actualização do ecrã.

Em síntese, o sistema embebido da Toradex com todas as suas características, com a utilização de hardware para expandir as E/S e com o uso da Framework Qt permite a implementação das mais diversas aplicações, quer de controlo e monitorização de dados ou de outro tipo.

Do ponto de vista do autor, para além da satisfação de concluir o projecto com os requisitos cumpridos, salienta-se o conhecimento adquirido nas várias áreas implicadas na implementação do projecto.

## 6.2 TRABALHO FUTURO

O projecto actual encontra-se a funcionar cumprindo os requisitos propostos, no entanto este pode ser melhorado tornando-o mais flexível, sendo proposto como trabalho futuro os seguintes pontos:

- Utilizar um protocolo de comunicação mais robusto e flexível, como o CANBus de forma a permitir um maior alcance, velocidade, número de dispositivos e melhor segurança nas transmissões.
- Utilizar um processador com maior capacidade de processamento do que o PXA270 para melhorar a resposta do sistema a aplicações mais complexas.
- O desenvolvimento do hardware deve ter em conta uma maior adopção de componentes SMD para diminuir espaço e optar pela utilização de mais algumas E/S comuns para também diminuir o espaço necessário pelos conectores.
- Melhorar a implementação da máquina de estados para se tornar mais flexível, podendo permitir diversas máquinas de estados para execuções diferentes.

# REFERÊNCIAS


[1] Jianmin Duan, F. L. (2008). "Research of Key Technologies in a Windows CE-based Monitoring and Control System". Industrial Electronics and Applications, ICIEA 2008, 494-496.

[2] Ogawa, M., & Henmi, Y. (2006). "Recent Developments on PC+PLC based Control Systems for Beer Brewery Process Automation Applications". SICE-ICASE, 2006. International Joint Conference, 1053-1056.

[3] Jacqueson, Nelly (1999). "Windows CE for Industrial Computing". Real-Time Magazine, 76-80

[4] Wikipédia (Julho de 2008). *Embedded System.* Consultado a 5 de Junho de 2009. Disponível em: http://en.wikipedia.org/wiki/Embedded_system

[5] Toradex (Maio de 2008). *Orchid Datasheet*. Consultado a 1 Junho de 2009. Disponível em: http://www.toradex.com/@api/deki/files/511/=Orchid_V1.0_Datasheet_Rev1.3.pdf

[6] Antratek (2009). *Atmel AVR32 tools & boards*. Consultado a 1 de Julho de 2009. Disponível em: http://www.antratek.com/AVR32.html

[7] BeagleBoard (2009). *BeagleBoard Hardware*. Consultado a 1 de Julho de 2009. Disponível em: http://beagleboard.org/hardware

[8] Gumstix (2009). *Computer-on-Module*. Consultado a 1 de Julho de 2009. Disponível em: http://www.gumstix.com/store/catalog/motherboards.php

[9] Gumstix (2009). *Expansion Boards*. Consultado a 1 de Julho de 2009. Disponível em: http://www.gumstix.com/store/catalog/expansion.php

[10] Armadeus (2008). *APF27*. Consultado a 1 de Julho de 2009. Disponível em: http://www.armadeus.com/english/products-processor_boards-apf27.html

[11] Armadeus (2008). *APF9328 DevFull*. Consultado a 1 de Julho de 2009. Disponível em: http://www.armadeus.com/english/products-development_boards-apf9328_devfull.html

[12] Hall, Mike (2005). "Windows CE 5.0 for real-time systems". Embedded Computing Design

[13] GuideBookGallery (2006). Imagem do Sistema Operativo Windows CE. Consultado a 1 de Julho de 2009. Disponivel em: http://www.guidebookgallery.org/guis/windowsce/screenshots

[14] Wikipédia (Julho de 2009). *VxWorks*. Consultado a 1 de Julho de 2009. Disponível em: http://en.wikipedia.org/wiki/VxWorks

[15] Wikipédia (Julho de 2009). *MontaVista*. Consultado a 1 de Julho de 2009. Disponível em: http://en.wikipedia.org/wiki/MontaVista_Software

[16] MontaVista (2009). *Real-Time Linux*. Consultado a 3 de Julho de 2009. Disponível em: http://www.mvista.com/real_time_linux.php

[17] Toradex (2009). *Colibri Modules*. Consultado a 3 de Julho de 2009. Disponível em: http://www.toradex.com/En/Products/Colibri_XScale_Computer_Modules_Overview_PXA255_PXA270_PXA270M_PXA300_PXA310_PXA320_ARM

[18] Toradex (Maio de 2008). *Orchid Datasheet*. Consultado a 3 Julho de 2009. Disponível em: http://www.toradex.com/@api/deki/files/511/=Orchid_V1.0_Datasheet_Rev1.3.pdf

[19] Toradex (Março de 2007). *Sharp WQVGA TFT Datasheet*. Consultado a 6 Julho de 2009. Disponível em: http://www.toradex.com/@api/deki/files/684/=WQVGA_V01_00_Datasheet2007-03-20-lowres.pdf

[20] DevHood (2001). .*Net Framework*. Consultado a 7 de Julho de 2009. Disponível em: http://www.devhood.com/training_modules/dist-a/Intro.NET/intro.net.htm

[21] Microsoft MSDN (2009). .*NET Framework Conceptual Overview*. Consultado a 8 de Julho de 2009. Disponível em: http://msdn.microsoft.com/en-us/library/zw4w595w.aspx

[22] Wikipédia (2009). *Serial Peripheral Interface Bus.* Consultado a 20 de Junho de 2009. Disponível em: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

[23] Wikipédia (2009). *$I^2C$*. Consultado a 10 de Junho de 2009. Disponível em: http://en.wikipedia.org/wiki/I%C2%B2C

[24] Microchip (2007). *MCP23017*. Consultado a 11 de Junho de 2009. Disponível em: http://ww1.microchip.com/downloads/en/DeviceDoc/21952b.pdf

[25] Optonic (2004). *PS2502-4*. Consultado a 20 de Junho de 2009. Disponível em: http://www.optoinc.com/datasheets/PS2502-4.pdf

[26] Cosmo (2002). *KP1040*. Consultado a 20 de Junho de 2009. Disponível em: http://www.milliard.com.hk/image/COSMO/KP1040.pdf

[27] Intel (Abril de 2004). *Intel® PXA27x Processor Family*. Consultado a 2 de Junho de 2009. Disponível em: http://www.balloonboard.org/hardware/300/ds/PXA270-dev-manual.pdf

[28] Hall, Mike; Maillet, Steve (2004). "Cats, Toast, and Interrupts on Windows CE .NET". Microsoft's MSDN

[29] GlobalSpec (2009). *High Rigidity Uniaxial Electric Actuator*. Consultado a 1 de Julho de 2009. Disponível em: http://www.globalspec.com/FeaturedProducts/Detail/High_Rigidity_Uniaxial_Electric_Actuator/15628/0?deframe=1

[30] AEB-Robotics (2008). *MOTOE56LR*. Consultado a 2 de Julho de 2009. Disponível em: http://www.aeb-robotics.com/products/mcs-series-1/motoe56lr

[31] AEB-Robotics (2006). *MOTOE56LR*. Consultado a 2 de Julho de 2009. Disponível em: http://www.aeb-robotics.com/pdf_download/download/manuali/man_rbt5_eng.pdf

[32] RS Amidata. *Motor, 4rpm,12Vdc con Engranaje Reduct.* Consultado a 3 de Julho de 2009. Disponível em: http://pt.rs-online.com/web/search/searchBrowseAction.html?method=getProduct&R=0440313

[33] SMC (2009). *Linear Actuators*. Consultado a 4 de Julho de 2009. Disponível em: http://www.smcaus.com.au/pages/product_selection.html#

[34] Schmersal (2009). *AZ 16zi with connector*. Consultado a 6 de Julho de 2009. Disponível em: http://products.schmersal.com/585/546/00170/range.html?lang=en

[35] Moeller (2009). *SL Signal Towers.* Consultado a 9 de Julho de 2009. Disponível em: http://www.moeller.net/en/products_solutions/motor_applications/command/signaltowers.jsp

# BIBLIOGRAFIA


1. Blanchette, J., & Summerfield, M. (2006). *C++ GUI Programming with Qt 4.* Prentice Hall.

2. Boling, D. (2003). *Programming Microsoft® Windows® CE .NET, Third Edition.* Microsoft.

3. Burdick, R. (1999). *Essential windows CE application programming.* John Wiley & Sons.

4. *I2C-bus specification and user manual.* (2007). Philips.

# ANEXOS

# ANEXO I - COMPILAÇÃO DA *FRAMEWORK* QT PARA WINDOWS CE

Com a escolha da *framework* Qt da Nokia para o desenvolvimento do ambiente gráfico da aplicação é necessário compilar devidamente o código fonte fornecido no site (www.qtsoftware.com) para que este possa gerar as bibliotecas necessárias para que a aplicação consiga ser executada no sistema operativo Windows, Figura 113.

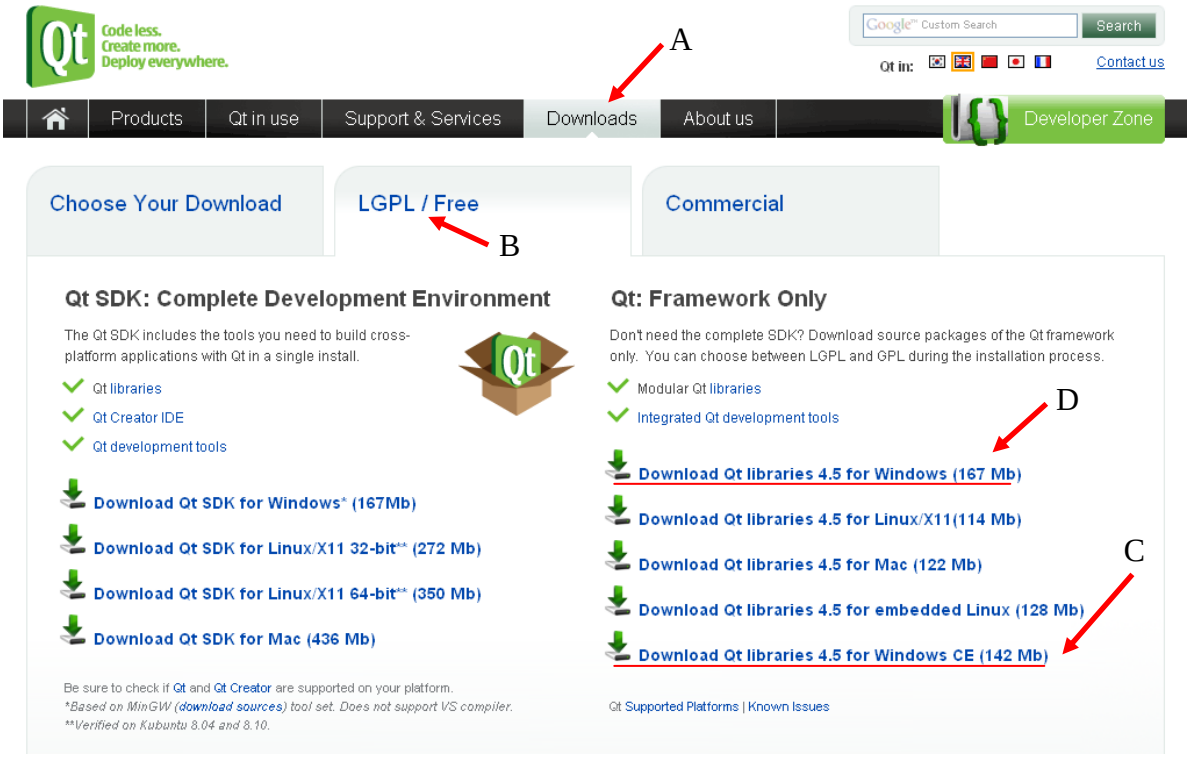



Figura 113 Site da Qt Software, secção de downloads.

Com o *browser* preferido, navega-se até ao site Qt Software (www.qtsoftware.com) onde se selecciona "*Downloads*" (A), seguido de "*LGPL/Free*" (B) que indica a versão livre, onde se selecciona as livrarias para o sistema operativo Windows CE (C), é necessário igualmente efectuar o download das livrarias do Qt para a plataforma Windows (D), estas são necessárias para incluir o Qt no Visual Studio 2008.

Após o download das bibliotecas é necessário efectuar igualmente o download do *plugin* para o Visual Studio 2008, que pode ser encontrado no final da página web, Figura 114.


Figura 114 Plugin para o Visual Studio.

Após o download das livrarias, Figura 115, é necessário no caso da livraria para o sistema operativo Windows CE, extrair o seu conteúdo para uma pasta com um caminho não muito longo (ex: C:\Qt\qt-4.5.1).

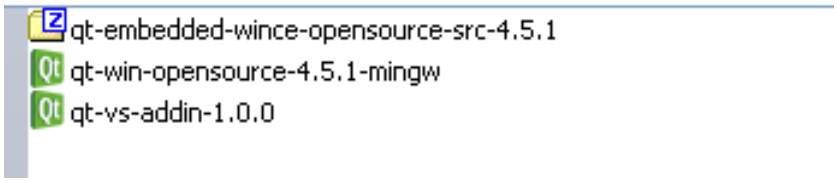



Figura 115 Software necessário para instalar a Framework Qt.

Os vários ficheiros extraídos de *qt-embedded-wince-opensource-src-4.5.1.zip* não estão compilados, não sendo possível usá-los no Windows CE. Para que seja possível a criação das livrarias para o Windows CE é necessário compilar os ficheiros, o que implica que exista uma instalação no Visual Studio 2008, neste caso.

As instruções para a compilação podem ser encontradas num ficheiro HTML (C:\Qt\qt-4.5.1\doc\htm\index.html), Figura 116.

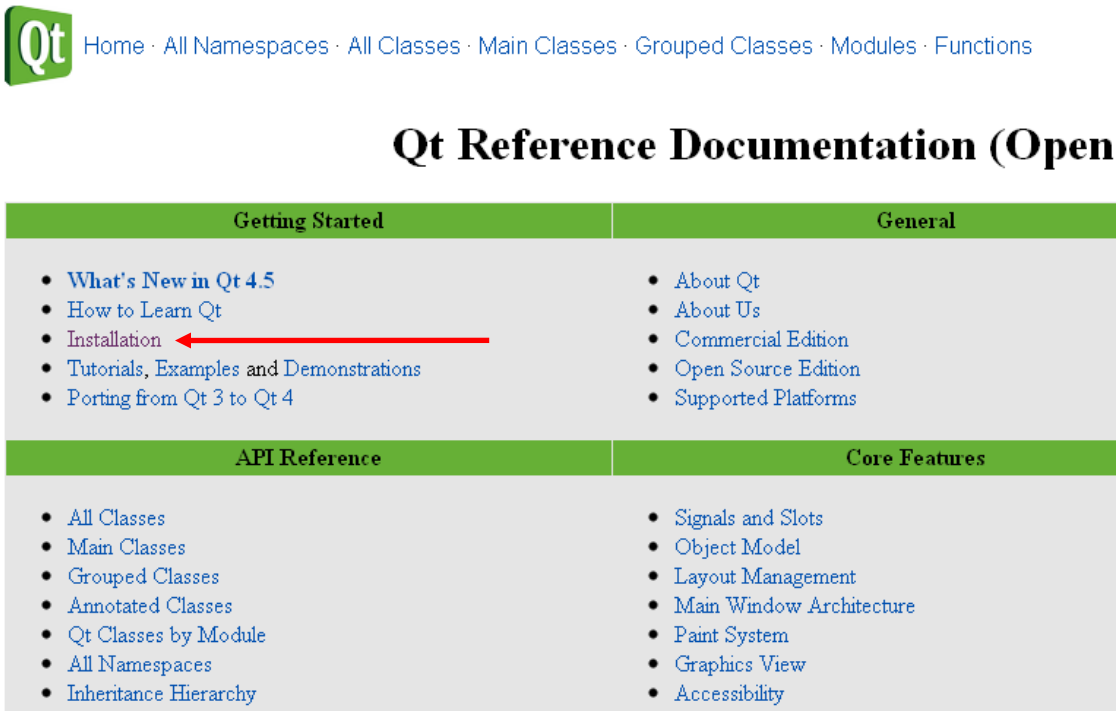



Figura 116 Instruções de instalação na documentação disponibilizada.

Contudo esta descrição da instalação é um pouco ambígua em alguns passos, para colmatar este ponto de seguida demonstra-se os passos a seguir para a compilação dos ficheiros extraídos para o sistema operativo Windows CE.

O primeiro passo para efectuar uma compilação correcta dos ficheiros extraídos é a definição de uma variável de ambiente com o nome *path*, para isto, no Windows XP segue-se o seguinte caminho:

***Meu Computador → Ver informação de sistema → Avançadas → Variáveis de ambiente***

Procurando nas "Variáveis do sistema" a existência de uma variável *path*, Figura 117. Se esta não existir deve então ser criada com o seguinte valor: C:\Qt\qt-4.5.1\bin. Caso esta já exista deve ser acrescentado o valor indicado anteriormente.

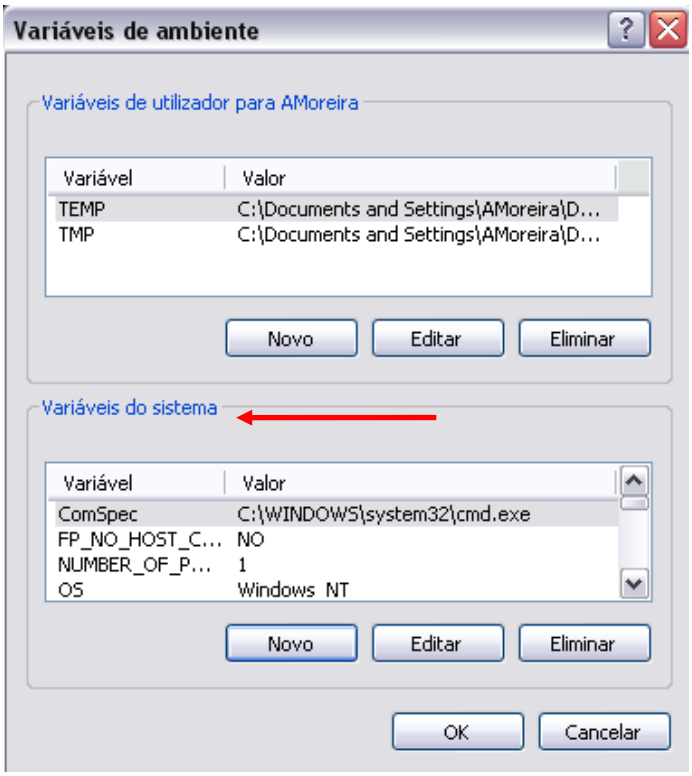



Figura 117 Modificação das variáveis de ambiente

Com a criação/actualização da variável *path* o sistema reconhece agora algumas ferramentas necessárias para a compilação.

O passo seguinte permite definir as variáveis necessárias para que as ferramentas do Visual Studio 2008 sejam reconhecidas, isto implica executar a consola do Visual Studio a partir do menu Iniciar do Windows XP, seguindo o seguinte caminho:

***Iniciar → Todos os programas → Microsoft Visual Studio 2008 → Visual Studio Tools →Visual Studio 2008 Command Prompt***

Iniciando deste modo a consola do Visual Studio 2008, Figura 118.

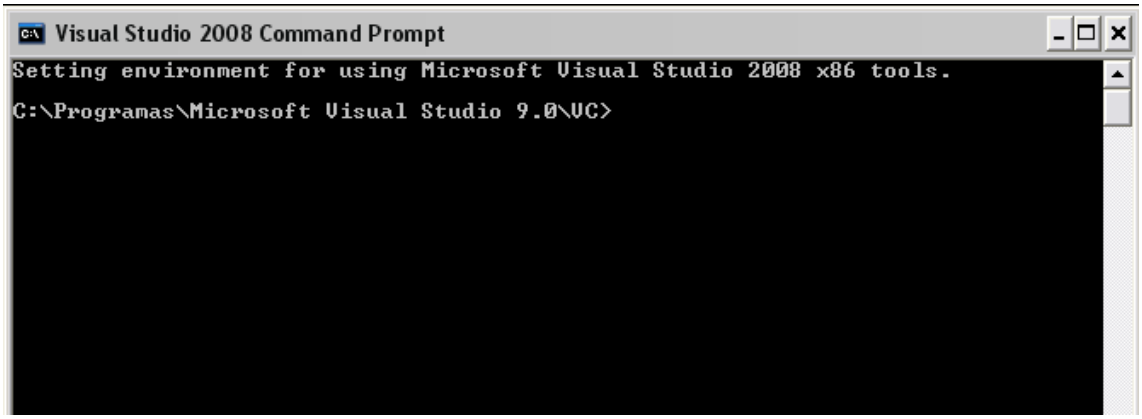



Figura 118 Inicio da consola do Visual Studio 2008

Com a consola aberta é necessário executar o seguinte ficheiro: "*vcvars32.bat*", que permite adicionar as variáveis de sistema necessárias, encontrando-se na pasta: C:\Programas\Microsoft Visual Studio 9.0\Common7\Tools\.

Após a execução deste passo é necessário ir para a pasta C:\Qt\qt-4.5.1\, onde se vai executar o ficheiro de configuração da compilação indicando a plataforma de compilação e a plataforma de destino, Figura 119:

***configure -debug-and-release -shared -platform win32-msvc2008-xplatform wince50standard-armv4i-msvc2008***

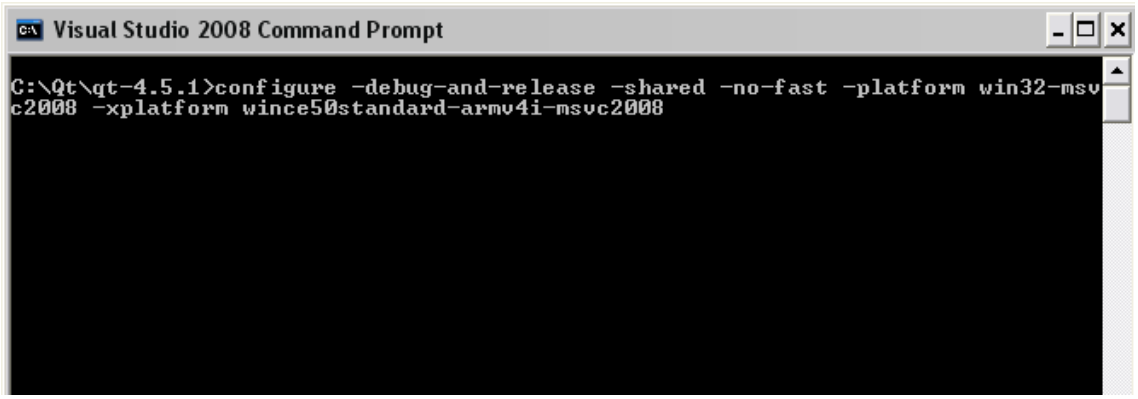



Figura 119 Primeira linha de código

Com a execução do comando anterior, que demora alguns minutos, o sistema ficará configurado para começar a compilação da *Framework* Qt.

Para finalizar a configuração é necessário a actualização de algumas variáveis, apenas possível se o SDK (Standard Software Development Kit) do Windows CE 5.0 estiver instalado no Windows XP, caso não esteja é necessário instalá-lo.

Caso todas as ferramentas envolvidas (Visual Studio 2008 e Windows CE 5.0 SDK) tenham sido instaladas com os caminhos de instalação por defeito, as linhas de comando a executar serão as seguintes, Figura 120:

***set INCLUDE=C:\Programas\Microsoft Visual Studio 9.0\VC\ce\include;***
***C:\Programas\Windows CE Tools\wce500\STANDARDSDK_500\Include\Armv4i***

***set LIB=C:\Programas\Microsoft Visual Studio 9.0\VC\ce\lib\armv4i;***
***C:\Programas\Windows CE Tools\wce500\STANDARDSDK_500\Lib\ARMV4I***

***set PATH=C:\Programas\Microsoft Visual Studio 9.0\VC\ce\bin\x86_arm;%PATH%***

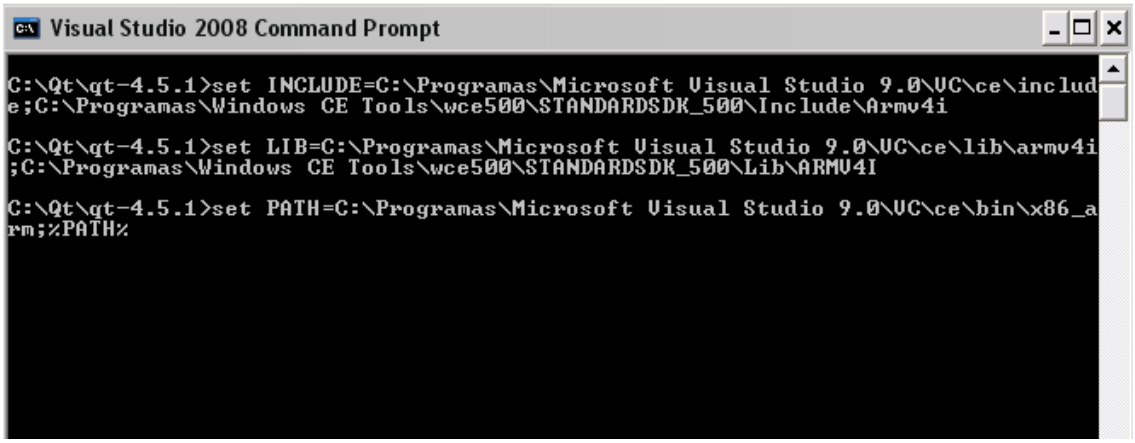



Figura 120 Configuração dos diversos PATHs

Executando de seguida o seguinte comando:

***setcepaths wince50standard-armv4i-msvc2008***

Por fim, como ultimo passo, executa-se o comando ***nmake*** que vai iniciar a compilação da Framework Qt para o Windows CE com as definições escolhidas, este processo pode demorar algumas horas. Se todo o processo correu sem problemas, a pasta C:\Qt\qt-4.5.1\lib\ deverá ter vários ficheiros do tipo. dll que são necessários na plataforma que executa o Windows CE, caso contrário deve-se recomeçar o processo de compilação novamente.

Com a compilação bem sucedida das livrarias para o Windows CE é agora necessário instalar as livrarias do Qt para o Windows XP, cujo processo é simplificado pelo instalador incluído no executável "qt-win-opensource-4.5.1-mingw.exe", depois desta instalação é necessário instalar o *plugin* do Qt para o Visual Studio 2008 através do executável "qt-vs-addin-1.0.0.exe".

Após a instalação destes, fica a faltar uma pequena configuração de forma que o Visual Studio 2008 consiga executar o "*Designer.exe*" e compilar com as livrarias da *Framework* Qt.

Esta configuração é efectuada no próprio Visual Studio 2008 através do seguinte passo: ***Qt (barra superior)*** → ***Qt Options*** como se pode verificar na Figura 121.

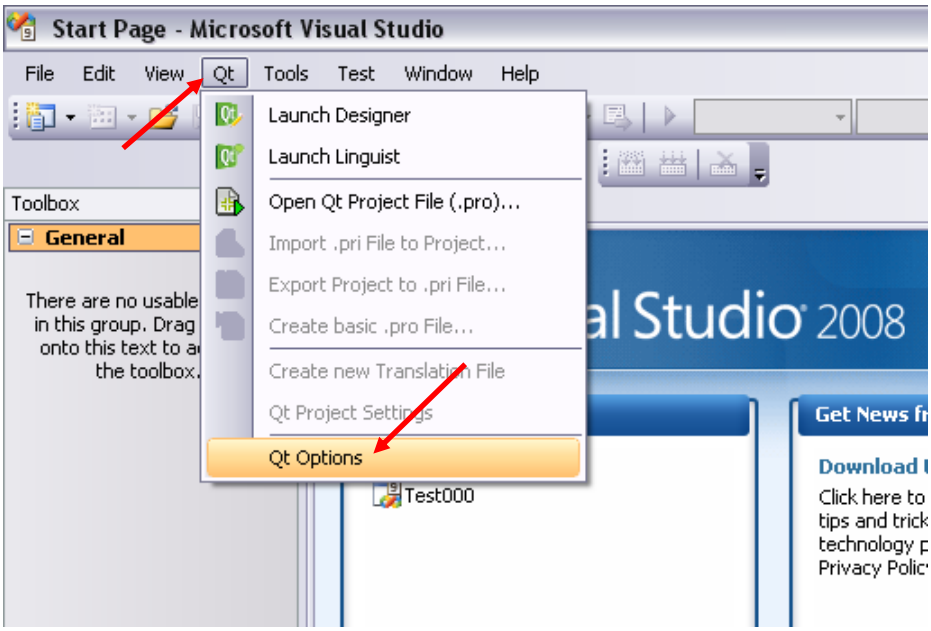



Figura 121 Configuração da Framework Qt no Visual Studio.

O menu de configuração aparece, Figura 122, onde se deve indicar o caminho para cada uma das livrarias instaladas (Windows XP e Windows CE), neste caso e seguindo os passos indicados anteriormente para a compilação é possível verificar as duas livrarias detectadas na figura.

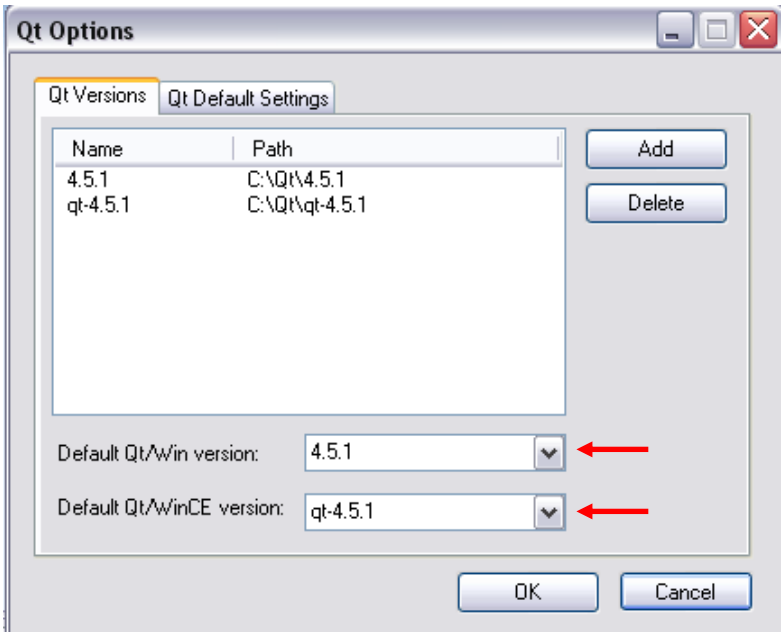



Figura 122 Indicação do caminho das livrarias da Framework Qt

Deste modo, o Visual Studio 2008 está configurado para compilar aplicações baseadas na *Framework* Qt para a plataforma com o Windows CE.

# ANEXO II - CIRCUITO FINAL DA PLACA DE EXPANSÃO DE SAÍDAS

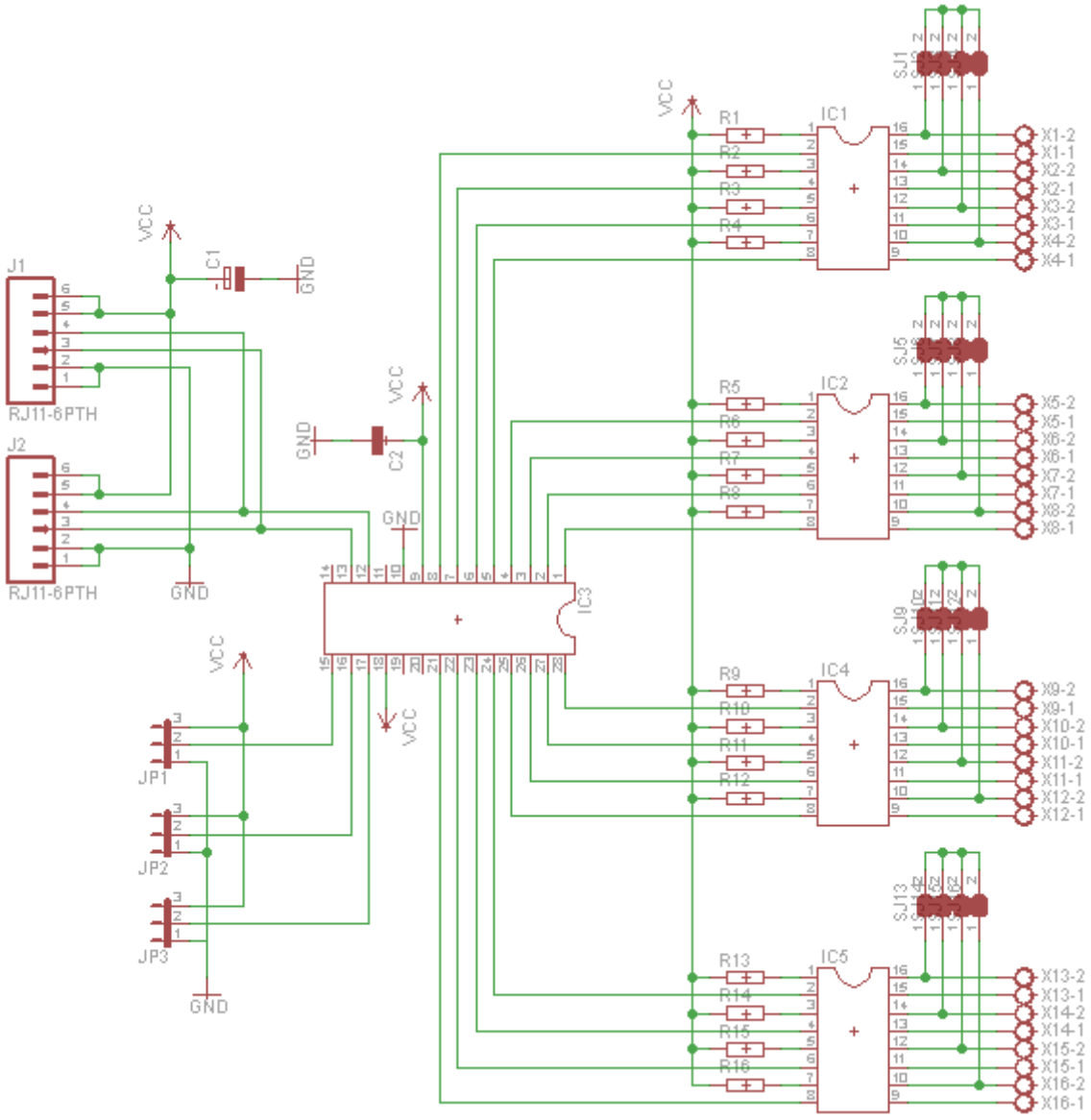



Figura 123 Circuito da placa de expansão de saídas

# ANEXO III - CIRCUITO FINAL DA PLACA DE EXPANSÃO DE ENTRADAS

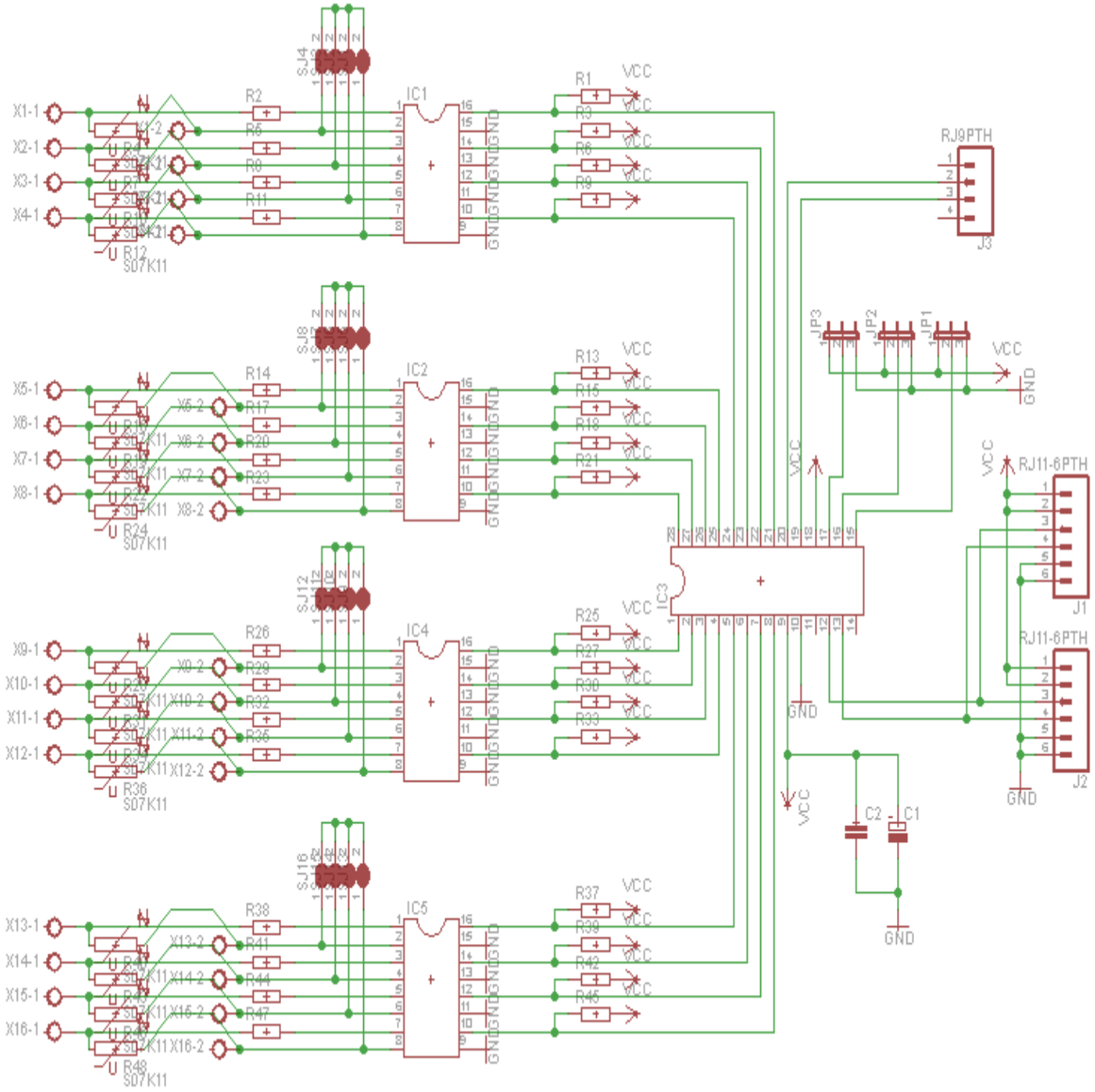



Figura 124 Circuito da placa de expansão de entradas

# ANEXO IV – DESENHOS DAS PLACAS DE EXPANSÃO DE E/S

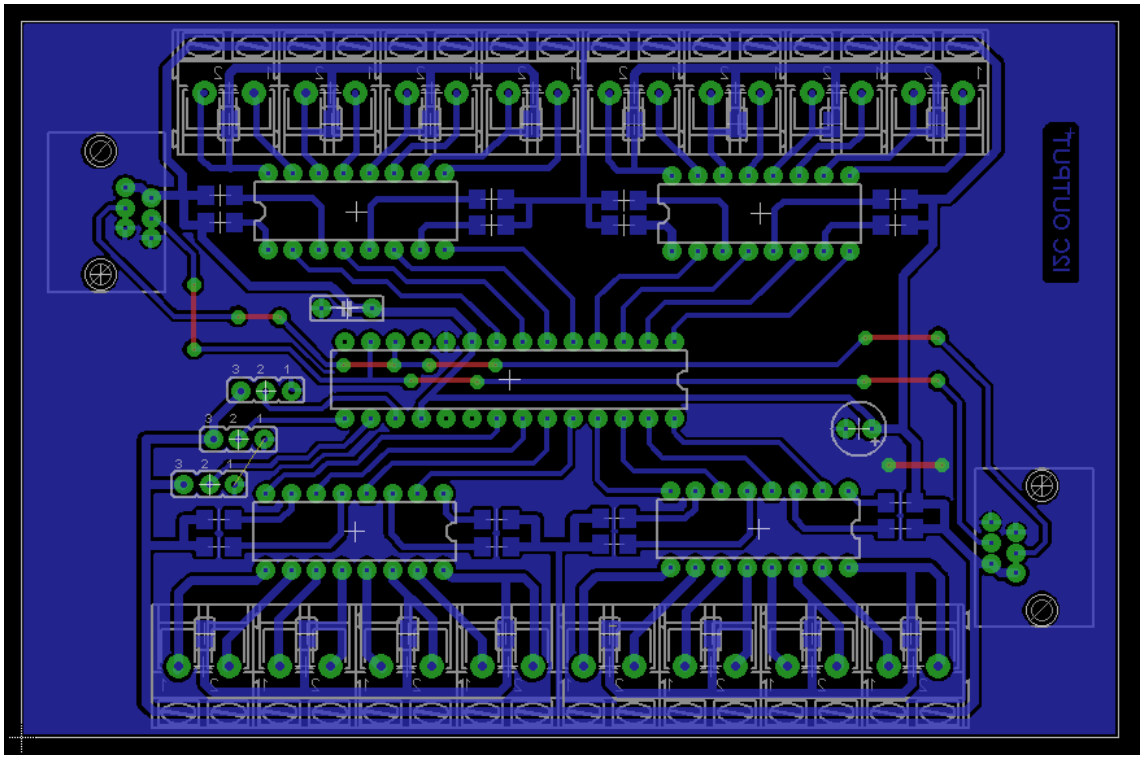


Figura 125 Desenho da placa de expansão de saídas

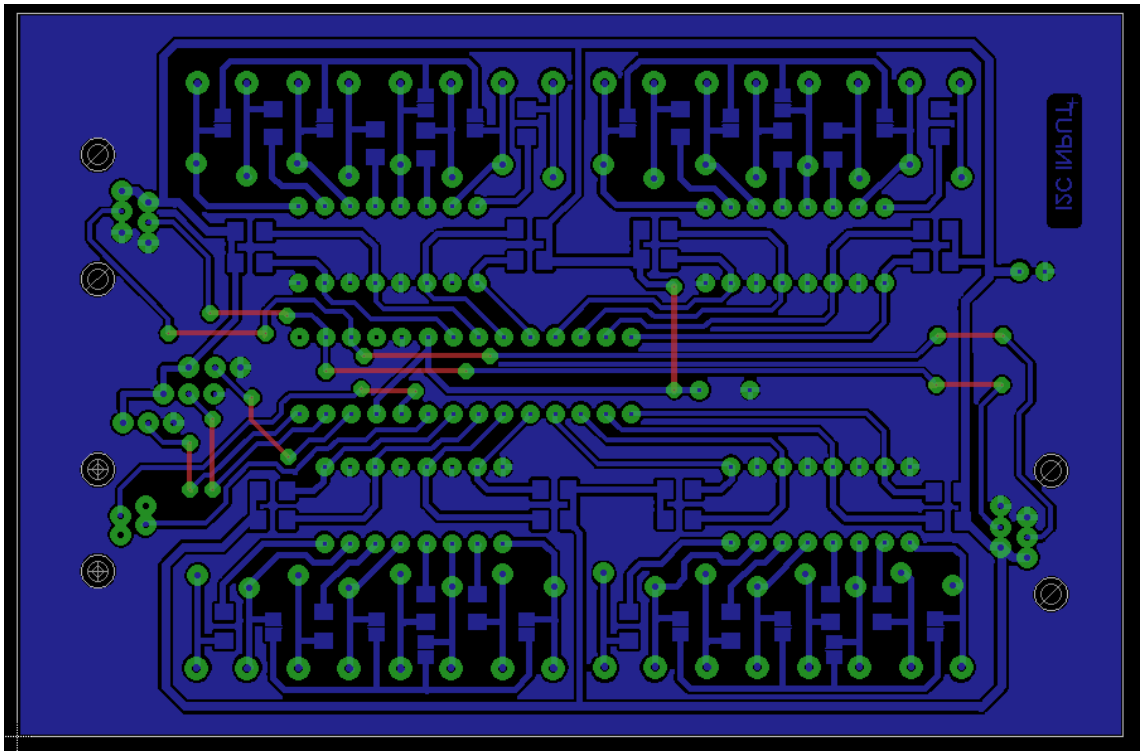


Figura 126 Desenho da placa de expansão de entradas