

Laboratório de Informática e Computadores
Semestre de Verão de 2008-2009

Trabalho prático

Autores:

31401 - Nuno Cancelo
32900 - José Lourenço
33595 - Nuno Sousa

Indíce

Enunciado.....	3
Introdução.....	3
Objectivo.....	3
Descrição Funcional.....	4
ARQUITECTURA DO SISTEMA.....	5
Controlo do Mostrador.....	5
Leitor de Teclado.....	6
Descodificador de Teclado.....	7
FIFO.....	7
Controlo Central.....	8
Desenvolvimento da Solução.....	9
Prólogo.....	9
Área de Hardware:.....	10
Implementação do Acesso ao Kit Didáctico e Controlo do display LCD.....	10
Implementação do Controlo do Teclado.....	14
Implementação do Fifo.....	15
Implementação da solução (em hardware).....	19
Área de Software:.....	20
Implementação do acesso ao Kit Didáctico e com o display LCD.....	20
Implementação da comunicação com o Teclado e com o Controlo do Mostrador.....	22
Conclusão.....	28
Bibliografia.....	29
Agradecimentos.....	30

Enunciado

Introdução

O presente trabalho serve de pretexto para que o aluno estude os ciclos de leitura e de escrita numa memória RAM, a descodificação de um teclado organizado em matriz espacial, a comunicação série, a sincronização entre sistemas, a construção estruturada de software, entre outros conceitos. Embora a estrutura proposta para resolução do projecto que em seguida se enuncia, possa estar um pouco afastada de uma solução real, tem a virtude de transmitir ao aluno uma série de problemáticas na área da síntese e teste de circuitos digitais, bem como na construção de programas em linguagem JAVA que interagem com sinais de uma estrutura hardware.

Objectivo

Tendo por base o sistema didáctico SD_USB_PORT, pretende-se implementar um Sistema para Controlo de Acessos (SCA) a instalações. O sistema tem o diagrama de blocos apresentado na Figura 1, sendo constituído por um teclado de 16 teclas, um mostrador LCD (*Liquid Cristal Display*) de duas linhas de 16 caracteres, e um trinco electromecânico para abertura da porta. O SCA tem dois modos de funcionamento (Operação e Manutenção) determinados pelo comutador M. Quando em modo de operação o dialogo é estabelecido com o utilizador através do teclado e do LCD. Em modo manutenção o dialogo com o gestor é totalmente realizado no PC.

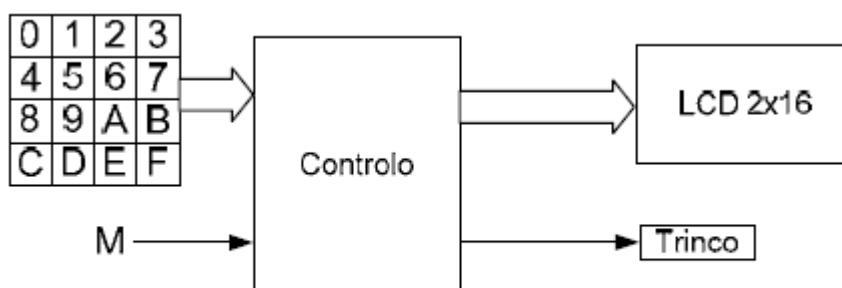


Figura 1 – Diagrama de blocos geral do SCA

Descrição funcional dos vários componentes que constituem o sistema SCA:

- **Teclado:** constituído por 16 teclas dispostas em matriz espacial. As 10 teclas numéricas de 0 a 9 servem para inserir o número do utilizador e o PIN (*Personal Identifier Number*). As restantes teclas de A a F são usadas para funções de controlo e edição
- **LCD:** no LCD são apresentadas mensagens que conduzam o utilizador ao longo das várias funções efectuadas através do teclado (abertura da porta, alteração de PIN, etc.).
- **Trinco electromecânico:** O trinco electromecânico realiza a abertura da porta de acesso às instalações. Para tal é necessário que seja activado entre 2 e 5 segundos.

Descrição Funcional

O sistema possui os Números, Nomes e PINs dos utilizadores que têm acesso às instalações protegidas. Esta informação estará armazenada num ficheiro de texto (com um utilizador por linha) que é carregado no início do programa e reescrito no final do programa.

O sistema armazena até 1000 utilizadores, que são inseridos e suprimidos através do teclado do PC pelo gestor do sistema. Para acesso às instalações, o utilizador, quando estiver presente no LCD a mensagem “Insert Number” deverá inserir os três dígitos correspondentes ao número de utilizador seguido da tecla **E**, e aguardar que o sistema lhe peça o PIN, por exibição no LCD das seguintes mensagens: nome do utilizador na primeira linha, e “Insert PIN ---” na segunda linha. Na inserção do PIN, por cada tecla premida é trocado um carácter ‘-’ por ‘*’.

O PIN é constituído por três dígitos numéricos. A inserção do PIN termina premindo a tecla **E** (*Enter*) iniciando-se de seguida o processo de autenticação. Terminado o processo de autenticação é exibida a mensagem ‘OK’ ou ‘Invalid PIN’. Durante a inserção do número de utilizador ou do PIN, poderá utilizar a tecla **C** (*Cancel*). Ao premir a tecla **C** o sistema tem o seguinte comportamento:

caso o LCD contenha dígitos limpa todos os dígitos, se não contiver dígitos aborta o processo em curso.

Sobre o sistema podem-se realizar as seguintes acções em modo Manutenção:

- **Inserção de utilizador** - Tem como objectivo inserir um novo utilizador no sistema. O sistema atribui o primeiro número disponível, e espera que seja introduzido pelo gestor do sistema o nome e o PIN do utilizador. O nome tem no máximo 16 caracteres.

- **Remoção de utilizador** - Tem como objectivo remover um utilizador do sistema. O sistema espera que o gestor do sistema introduza o número de utilizador e pede confirmação depois de apresentar o nome.

- **Inserir mensagem** - Permite associar uma mensagem de informação dirigida a um utilizador específico. No processo de autenticação de acesso às instalações, caso exista uma mensagem associada ao utilizador, esta é exibida no LCD, tendo o utilizador que premir a tecla **C** ou **E**. No caso de premir a tecla **E**, o trinco é accionado e a mensagem removida do sistema, caso prima a tecla **C** a acção de abertura é cancelada e a mensagem permanece no sistema.

- **Terminar o programa** - A acção que permite terminar o programa. O programa termina após a confirmação do utilizador e a reescrita do ficheiro com a informação dos utilizadores.

Em modo de Operação para além da abertura da porta o utilizador pode alterar o PIN.

- **Alterar PIN** - Acção realizada através do teclado de utilizador e que permite alterar o PIN. O novo PIN é pedido ao utilizador se no processo de autenticação de acesso às instalações o utilizador premir após o PIN a tecla **A** (Alterar) em vez da tecla **E**.

Durante a execução das acções realizadas através do teclado do PC não podem ser realizadas acções no teclado do utilizador e no LCD deve constar a mensagem “Out off Service”.

Tratamento de excepções:

- Se durante o decurso de uma operação realizada no teclado do utilizador não for premida uma tecla no intervalo de 5 segundos a operação é abortada.

- Se durante a inserção de um campo numérico for premida uma tecla não numérica, esta

deve ser ignorada.

ARQUITECTURA DO SISTEMA

O SCA, descrito na Figura 2, será realizado recorrendo a uma solução *hardware/software*. O **Leitor de Teclado** e o **Controlo do Mostrador** deverão ser implementados em *hardware* e o controlo central será implementado em *software* a executar num PC. O **Leitor de Teclado** descodifica as teclas, armazena-as até ao limite de 7 e disponibiliza o código das teclas ao **Controlo Central**. O **Controlo Central** processa as teclas, envia para o **Controlo do Mostrador** informação contendo os dados a apresentar no LCD e ordem de accionamento do Trinco. O envio da informação do **Controlo Central** para o **Controlo do Mostrador** é realizado em série.

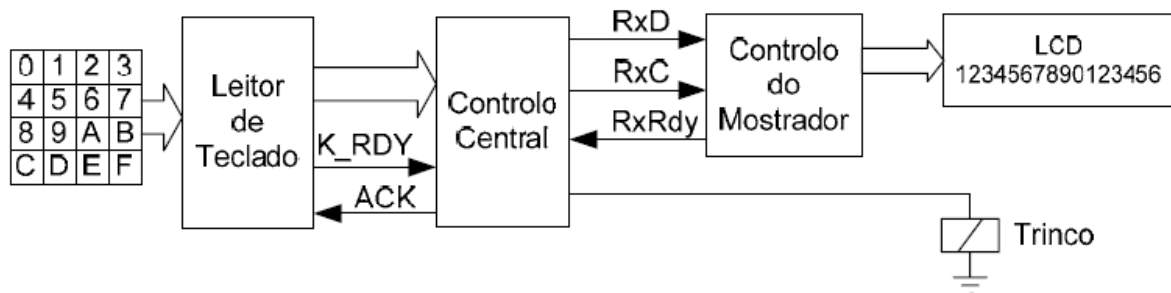


Figura 2 – Diagrama de blocos do SCA

Controlo do Mostrador

O módulo Controlo do Mostrador tem o diagrama de blocos mostrado na Figura 3. Este módulo deverá realizar a recepção em série de informação enviada pelo Controlo Central, e envia-la para o LCD segundo a especificação do fabricante.

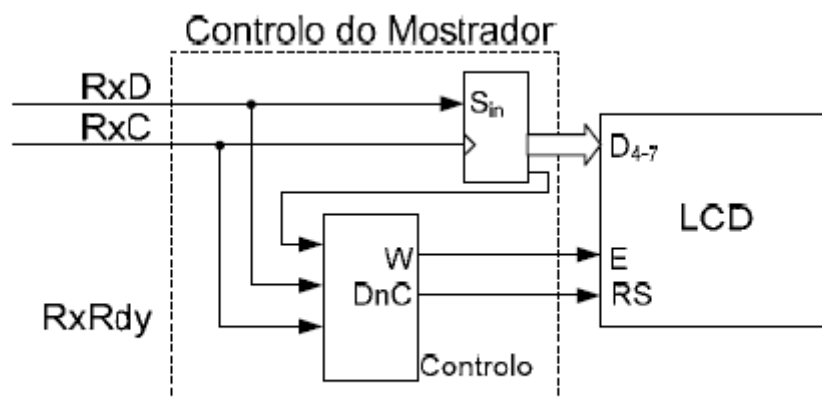


Figura 3 – Arquitectura do Controlo do Mostrador

O Controlo do Mostrador recebe em série uma trama constituída por um bit de controlo e quatro de dados. Os quatro bits de dados são armazenados para posterior envio ao LCD. A mensagem contém como primeiro bit de informação, o bit **DnC** que indica se os quatro bits de

dados são de Controlo ou Dados.

A comunicação série é processada através de três linhas: dados **RxD**, *clock* **RxC** e **RxRdy** (sinal de protocolo) para indicar que o sistema de Controlo do Mostrador está disponível para a recepção de uma trama.

A comunicação realiza-se segundo o protocolo ilustrado na Figura 4. Note que o sinal de sincronismo **RxC** é fornecido pelo Controlo Central.

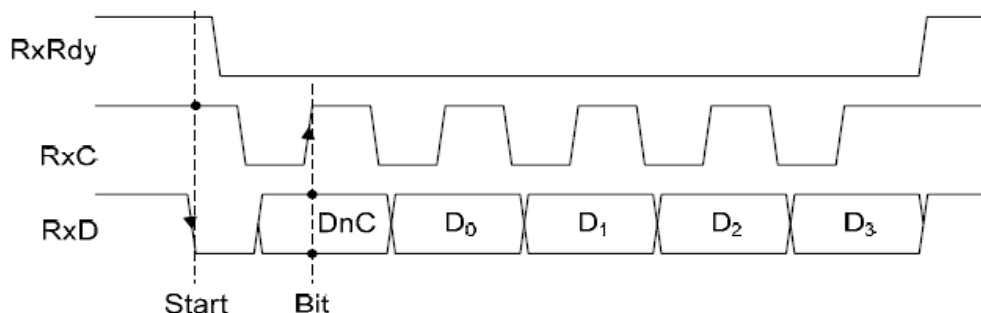


Figura 4 – Diagrama temporal da comunicação série

O protocolo considera como condição de sincronismo e início da comunicação (*start*), uma transição descendente na linha de **RxD** quando a linha **RxC** se encontra a “1”. Os bits da trama são recolhidos pelo **Receptor** nas transições ascendentes do sinal **RxC**. As três linhas usadas nas comunicações apresentam ainda os seguintes comportamentos:

- O **Receptor** só indica que está disponível para a recepção de uma nova trama (**RxRdy** activo) após ter processado a trama anterior
- No final da transmissão de uma trama, o **Controlo Central** deverá colocar as linhas **RxD** e **RxC** ao nível lógico “1”
- Se a meio da comunicação existir uma situação de *start* o **Receptor** reinicia a recepção.

Leitor de Teclado

O Leitor de Teclado é constituído pelo decodificador de teclado e pelo módulo FIFO (*First In First Out*) conforme o esquema bloco mostrado na Figura 5.

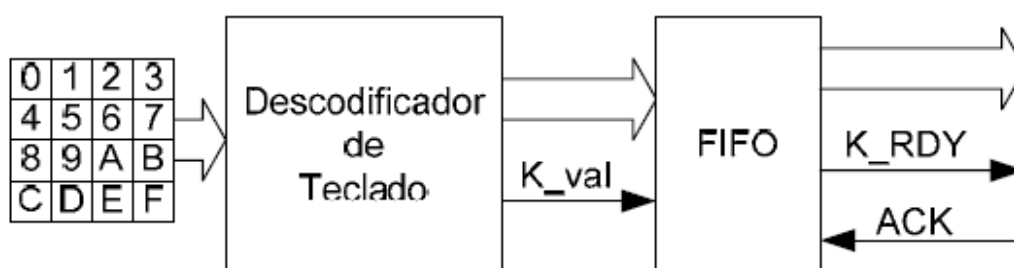


Figura 5 – Leitor do teclado

Descodificador de Teclado

Este módulo realiza a descodificação de um teclado matricial 4x4, segundo a solução apresentada na Figura 6.

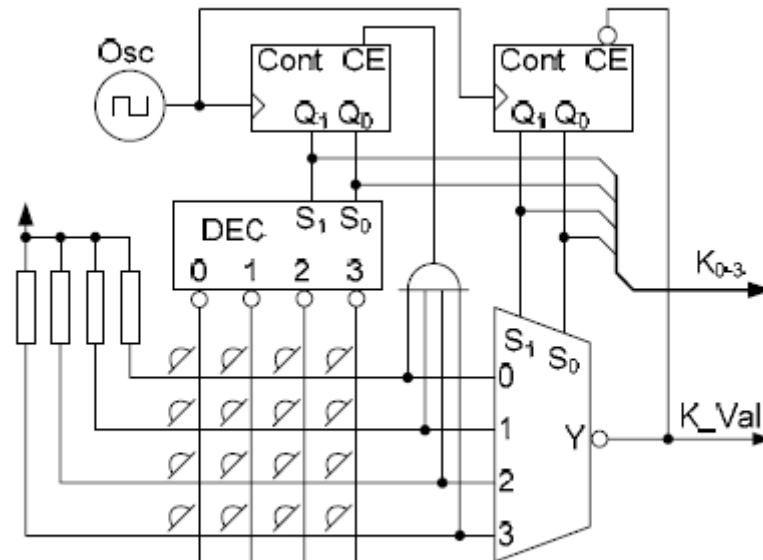


Figura 6 – Descodificador de teclado

FIFO

O FIFO é constituído por uma RAM de 8x4 onde vão sendo armazenadas as teclas que forem sendo premidas. O código das teclas armazenadas, ficam disponíveis em 4 bits para serem consumidos pelo sistema SCA. A ordem pela qual as teclas foram premidas não pode ser alterada aquando da entrega ao sistema SCA.

Para além dos 4 *bits* de codificação da tecla (K_{0-3}), existirão dois sinais K_RDY e ACK para controlo de fluxo. O sinal (K_RDY) é activado pelo Leitor de Teclado quando existe um código de tecla disponível em K_{0-3} , e o sinal ACK é activado pelo controlo central para indicar que já consumiu o código da tecla presente em K_{0-3} . Estes sinais têm o diagrama temporal apresentado na Figura 7.

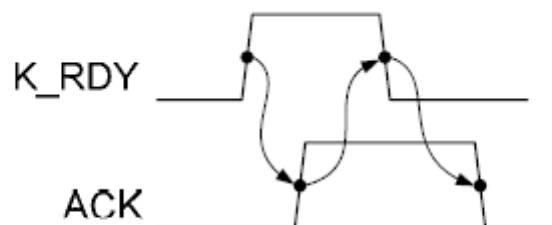


Figura 7 – Diagrama temporal de ACK e K_RDY

Controlo Central

Este módulo é responsável pela gestão do sistema SCA. Será implementado num PC que interage com o *hardware* através do módulo SD_USB_PORT.

Desenvolvimento da Solução

Prólogo

No decurso da implementação foram analisados os datasheets dos componentes utilizados e foram garantidos os respectivos protocolos de utilização indicados pelos fabricantes.

A análise de cada problema foi ponderado e o estudo da implementação foi exaustivo, sendo a execução somente efectuada após verificar que a solução para aquele problema seria de facto, nuns casos eficiente noutros eficaz.

No decorrer deste relatório será demonstrada a solução optada, sendo referido o tipo de raciocínio que nos levou aquela solução sem haver a necessidade de demonstrar graficamente o problema, o que poderia tornar o relatório longo, monótono e confuso.

O desenvolvimento da solução para o problema apresentado foi separado em duas áreas e elaborado em três fases cada:

Área de Hardware:

1. Implementação do Acesso ao Kit Didáctico e Controlo do display LCD.
2. Implementação do Controlo do Teclado
3. Implementação do Fifo

Área de Software:

1. Implementação da captura de sinais do Kit Didáctico e display LCD.
2. Implementação da comunicação com o Teclado e com o Controlo do Mostrador
3. Implementação do interface de operação e manutenção do Controlo de Acessos (Controlo Central)

Área de Hardware:

Implementação do Acesso ao Kit Didáctico e Controlo do display LCD.

Acesso ao Kit Didáctico

O acesso ao Kit Didáctico é efectuado pelo porto USB, que garante a comunicação entre o Kit e o computador (sendo necessário a inclusão de bibliotecas de funções previamente instaladas). O Kit têm disponíveis um porto de Input (a 8 bits) que recebe os dados vindos dos componentes programados (controlo de LCD, controlo de Keyboard, controlo do FIFO) e um porto de Output (a 8 bits) que garante a comunicação entre o Kit e Controlo de LCD (respeitando o protocolo estabelecido) e o FIFO.

Esta implementação será utilizado principalmente pela componente de software.

Controlo do display LCD

A análise desta fase iniciou-se no estudo do diagrama temporal do protocolo de comunicação em série que é utilizado para comunicar com o display LCD. Este diagrama representado na Figura 4, deu origem a um primeiro ASM chart bastante simplista do ponto de vista de análise mas bastante complexo do ponto de vista de alterações futuras, uma vez que estaríamos a ter vários estados de comunicação, tantos quantos os bits que fossem transmitidos. O que torna a solução restrita e imutável.

Assim sendo, partimos para uma segunda análise em que teríamos um “Contador de Bits” que iria contar os bits necessários para serem enviados e guardar os mesmos bits num *shift-register* e que mediante um sinal iria proceder ao envio dos mesmos. Esta solução já permite uma alteração do protocolo, quanto ao numero de bits transmitidos, sem que seja necessário alterar toda a programação.

Mas mesmo assim verificámos uma situação, que poderia originar problemas, e que repetia parte do ASM. Esta situação, referida pelo enunciado com *detecção de start*, foi contornada pela alteração da nossa máquina de estados em duas:

- Detecção de start
- Conversor serie para paralelo.

Assim ficamos com solução orientada por objectos, em que foram separados os problemas e o seu funcionamento é garantido pela sua relação simbiótica.

Mesmo com esta solução deparamos-nos com o problema de estarmos a analisar sinais no mesmo estado, ou seja, uma das condições para uma máquina funcionar estaria a ser analisada no mesmo estado em que essa condição seria lançada. Para ultrapassar esta questão, criamos um desfaseamento de clock de uma das máquina para que as condições fossem analisadas correctamente.

Depois de toda esta análise, que nos tomou uma boa parte do tempo, conseguimos apresentar uma solução modular e eficiente, que nos permite alterar o protocolo no futuro sem danificar a robustez da solução.

Esta solução é apresentada através da Figura 8.

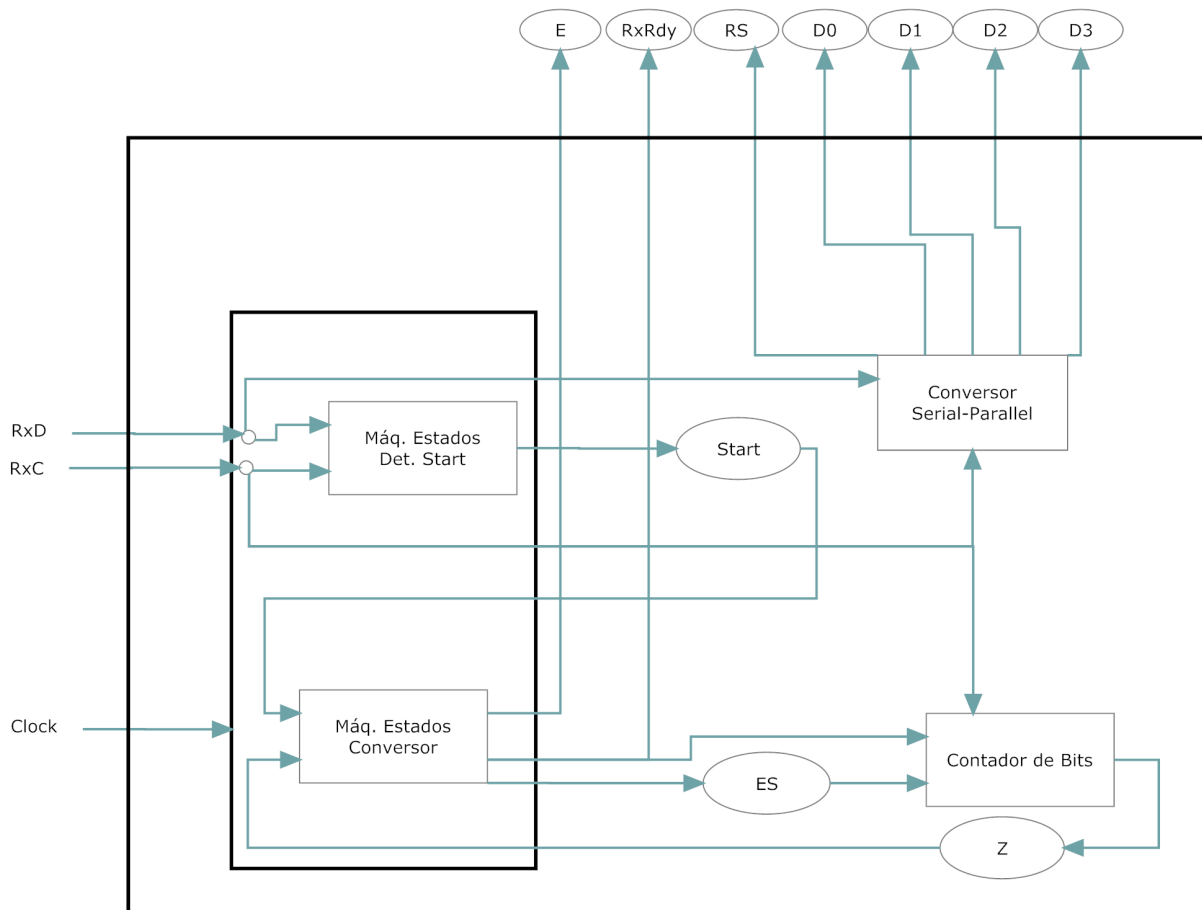


Figura 8: Controlo do Mostrador

Como se pode constatar, tendo os sinais de entrada RxD (que contem os dados a serem enviados) e RxC (que controlará o processamento do sinal RxD), daremos origem a uma série de sinais de dados para o display LCD (RS, R0-R3, E) e uma sinal de controlo (RxRdy) que notificará o Controlo Central que os dados foram processados.

A implementação desta solução pode-se verificar em forma de diagrama nas figuras 9 até 12.

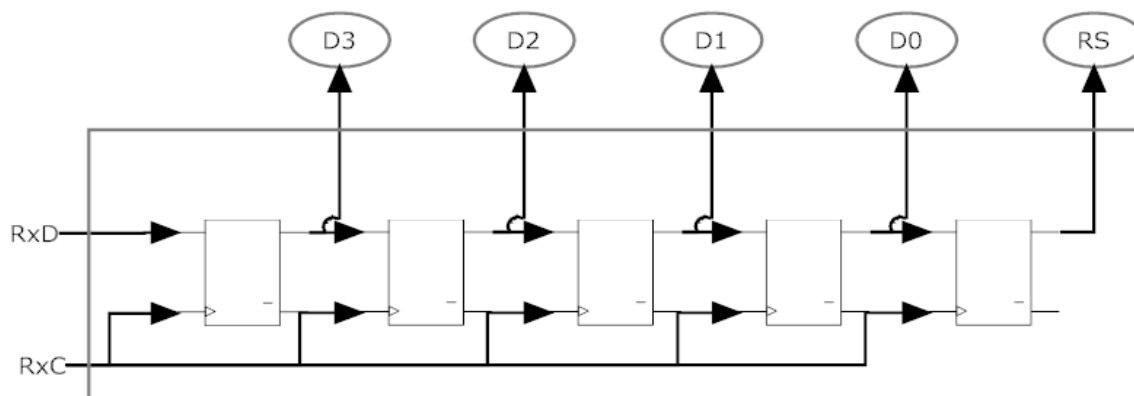


Figura 9: Conversor Serial-Parallel 5 Bits

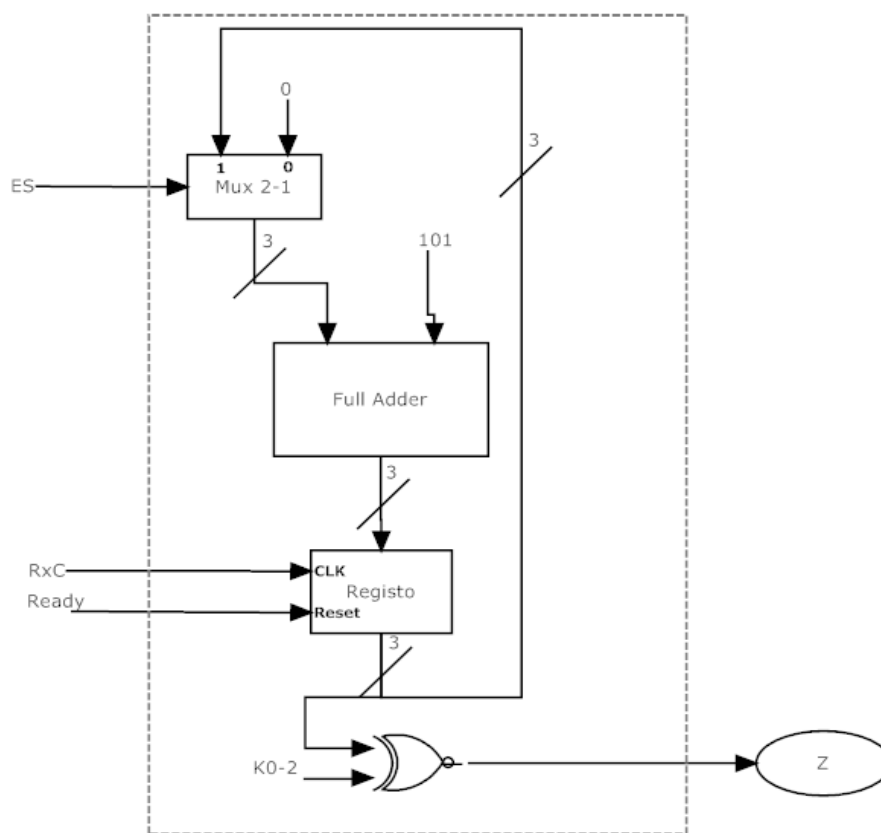


Figura 10: Contador de Bits

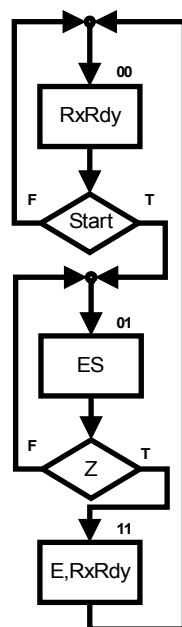


Figura 11:
 Conversor
 Serial-Paralel

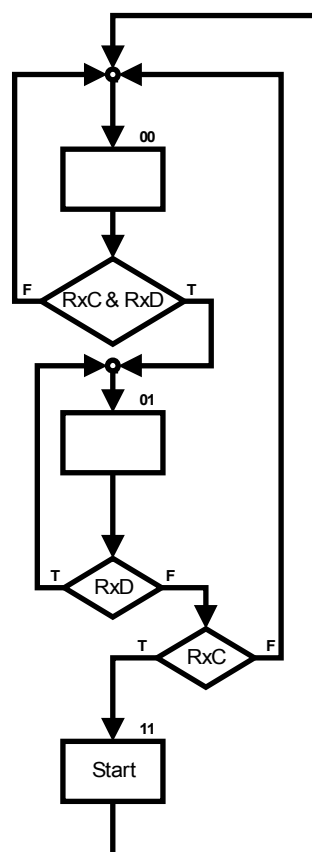


Figura 12:
 Detecção
 De
 Start

Implementação do Controlo do Teclado

Para esta fase, a implementação segue o esquema da figura 6, uma vez que a utilização de dois contadores é mais eficiente do que a utilização de um único para contador para contar as colunas e as linhas.

As resistências de pull-up, garantem o valor '1' em todas as teclas e quando uma tecla é pressionada a linha onde esta se encontra assume o valor lógico '0'.

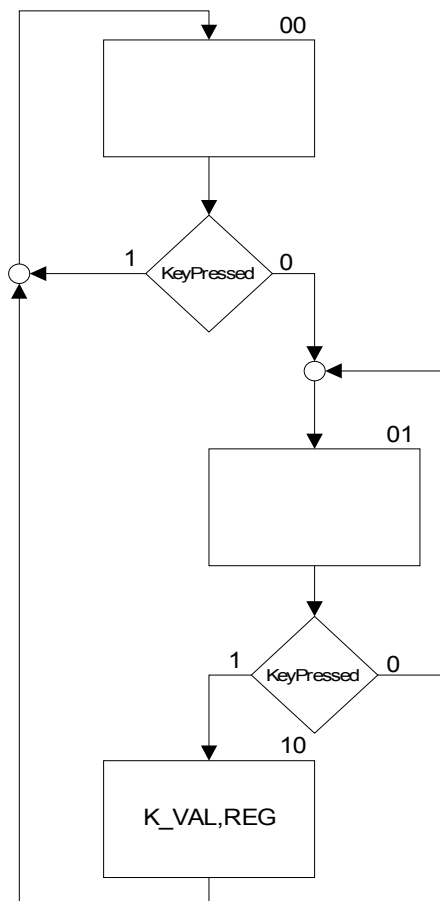
Atendendo à especificação do fabricante, referente à corrente admitida nas saídas da PAL, é necessário calcular o valor mínimo das resistências para que a especificação seja cumprida. Assim sendo, aplicando a Lei de Ohm, podemos chegar ao seguinte valor:

$$R = \frac{5V}{(4 \cdot 10^{-3})} = 1,25 K \Omega$$

Este valor será utilizado nas resistências de pull-up.

Neste momento sentimos a necessidade de criar uma máquina de estados para evitar que o sinal de aviso que há uma tecla seja continuamente enviado, evitando por um lado ter teclas repetidas enchendo depressa o buffer do fifo, e por outro lado permitindo que a tecla pressionada seja capturada mesmo que fique constantemente pressionada mas sendo somente recolhida uma vez.

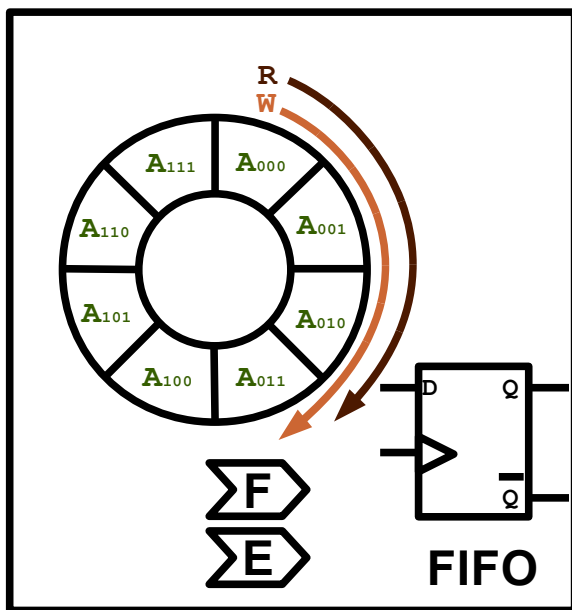
O seguinte diagrama, corresponde aos nossos desejos:



Implementação do Fifo

Esta secção do trabalho foi a mais complexa do projecto, uma vez que foi necessário muita análise para que toda a sua implementação usufrui-se toda a funcionalidade utilizando somente uma única ATF750. Optamos por esta solução, porque nos foi sugerido que era possível usar uma única e nós quisemos confirmar.

De início foi-nos necessário perceber o que era um FIFO ao pormenor e consultando o sítio na Internet da Wikipédia ([http://en.wikipedia.org/wiki/FIFO_\(computing\)](http://en.wikipedia.org/wiki/FIFO_(computing))) chegámos ao seguinte diagrama:



Flags:

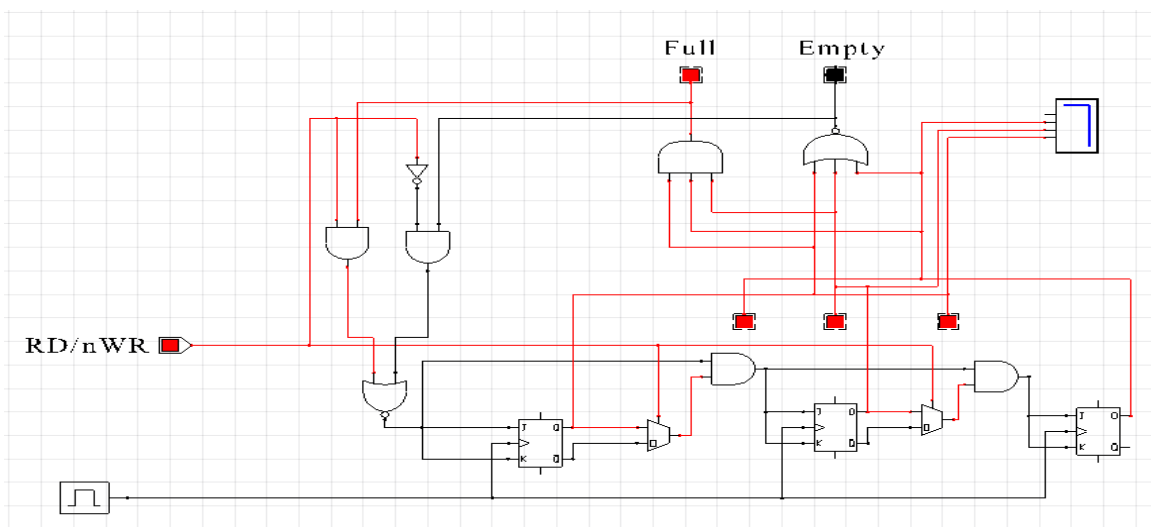
F = Queue is Full on Write

E = Queue is Empty on Read

BR = Bus Request

BG = Bus Granted

Nesta análise verificámos que haveria uma serie de flags que poderiam ser implementadas (como o almost full, almost empty) assim como saber a quantidade de elementos no Fifo a qualquer momento. Para tal, de forma a ter uma implementação alternativa caso não conseguimos chegar à nossa solução favorita, esboçamos o seguinte esquema:



Desde o início alcançamos obstáculos com o número de pinos e pinnodes que poderíamos utilizar para a nossa solução. Uma das primeiras simplificações que executámos foi com o sinal RNW. Este sinal, ao analisar o datasheets da RAM verificámos que utilizando lógica interna entra em modo read quando está a '1' e em modo escrita a '0'. Deste modo, simplificando a nossa solução fomos libertando sinais não necessários e originando outros com múltiplos propósitos.

Foi necessário criar tris-states na PAL de modo a ter os dados consistentes para a RAM e para aceder à memória foram criados dois contadores que tem como objectivo servirem de endereços de leitura e de escrita, que vão sendo incrementados com o sinal de WRCLK (para a escrita) e BG (para a leitura).

Neste momento foi propício o estudo prévio do Fifo, pois para evitar que os contadores se sobrepossem, estragando assim os dados já guardados, foram criadas duas flags:

- Flag Full: Que significa que os endereços são iguais e que a última operação foi uma escrita
- Flag Empty: Que significa que os endereços são iguais e que a última operação foi uma leitura

Neste momento concluímos que é necessário criar uma flag que representa qual foi a última operação realizada e para tal criámos um registo WR que quando a '1' a operação foi uma escrita e quando a '0' a operação foi uma leitura. Mais uma vez necessitámos analisar o nosso sistema para obter um sinal que servisse de entrada a este registo, uma vez que neste momento a quantidade de pinos disponíveis era nula. Assim concluímos que o sinal !RADDR era o sinal ideal uma vez que era utilizado na leitura e ignorado na escrita, e quando os dados estivessem estáveis lançávamos o sinal REGOP para activar o registo.

Estamos então presentes numa altura que garantimos os tempos de acesso, os endereços estáveis e os dados (para escrita e de leitura).

Surge então uma situação que não contávamos e que quase nos levou (tão perto de alcançar o nosso propósito) a desistir desta solução. O nosso integrado estava completamente lotado e nós não tínhamos forma de obter a tecla para enviar para o sistema.

A solução surge em forma de brincadeira, mas numa segunda análise verificámos que era a solução perfeita. A nossa solução passa por ter um registo fora da PAL que irá receber os dados originários da RAM e que será guardado segundo o sinal de REG (activado no acto de leitura) e um sinal de KEY_RDY que indica ao sistema que há tecla disponível. No retorno, o sistema lança um sinal ACK que tem duas funções:

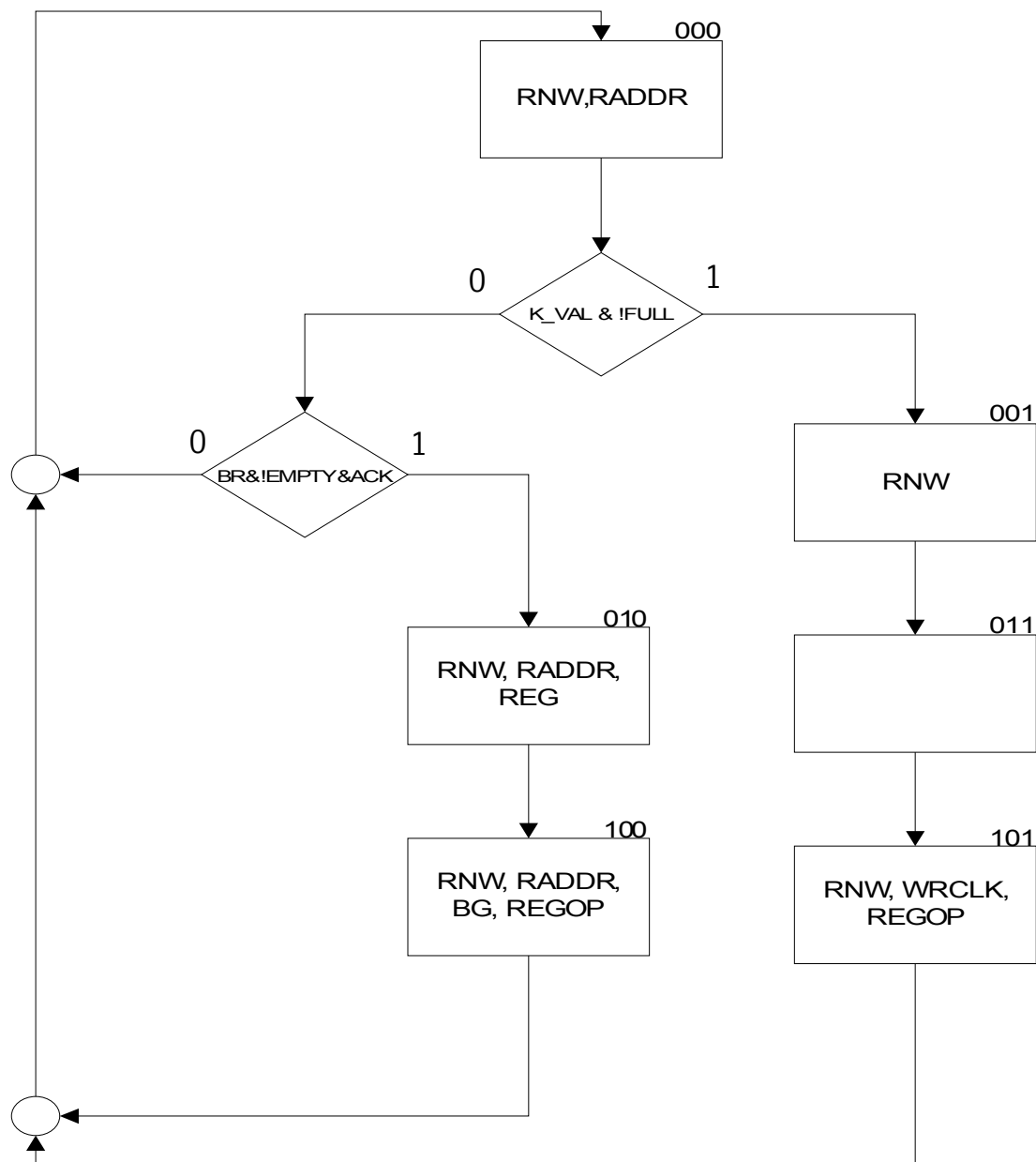
- Faz reset ao registo
- Notifica o Fifo que a tecla foi processada.

Desta forma, esta solução permite ter (de um certo ponto de vista, na pior das hipóteses) 10 teclas:

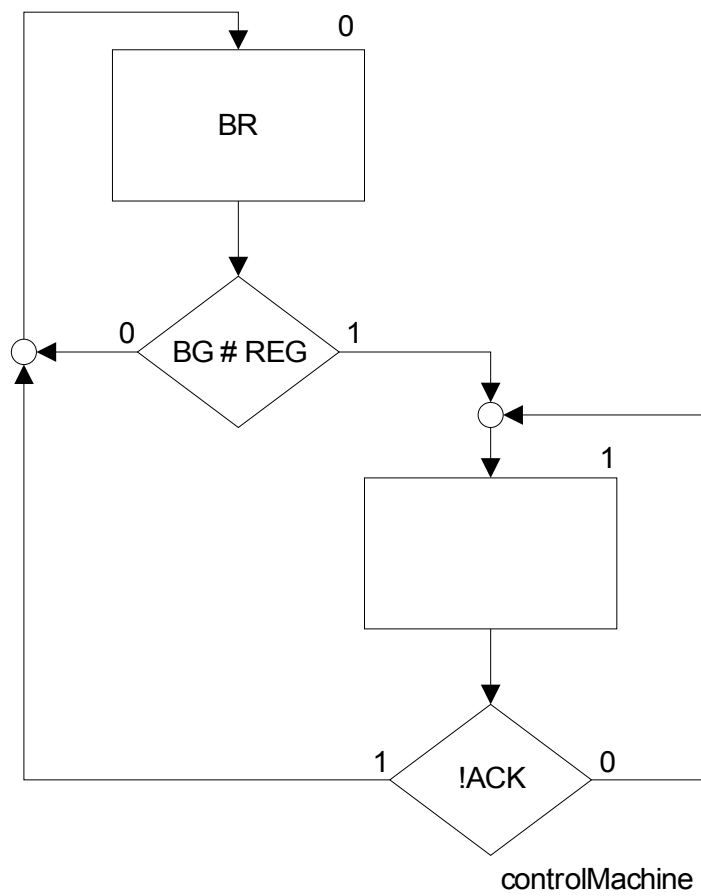
- 1 tecla pressionada no teclado matricial
- 8 teclas no FIFO
- 1 tecla no registo pronta a ler pelo sistema.

Esta solução permite que seja guardada e disponibilizada uma tecla sempre que haja necessidade sem que haja indisponibilidade, no entanto dadas as características da nossa solução não permite alterações para guardar mais teclas.

Para controlar todas as funcionalidades foram criadas duas máquinas de estados, uma que controla os dados e outra que controla o pedido do BUS.



mainMachine



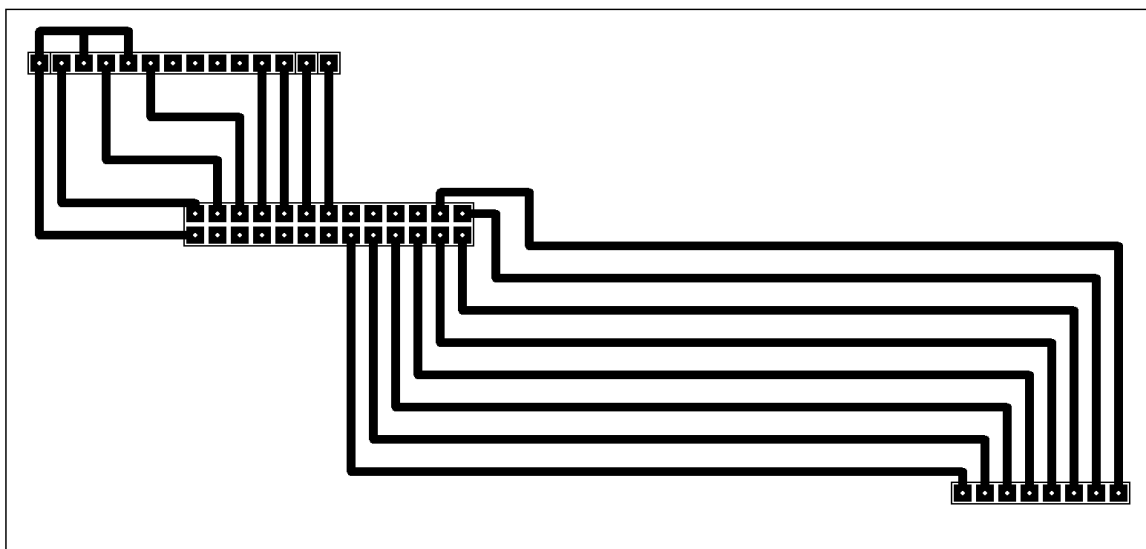
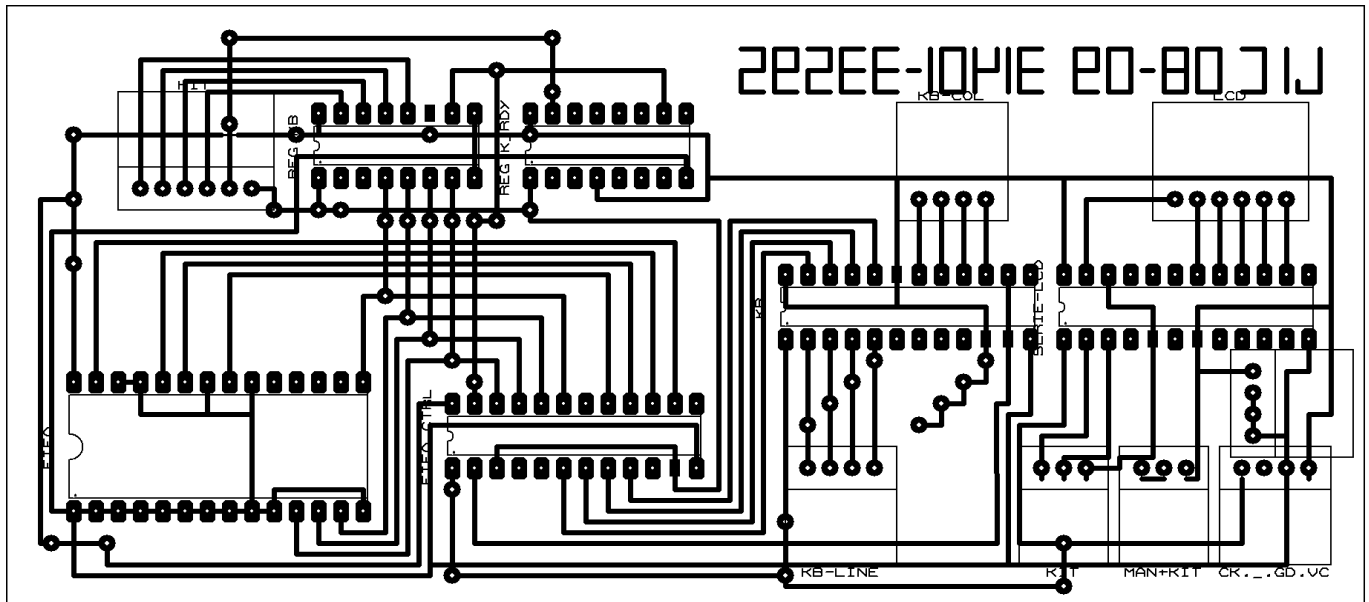
Implementação da solução (em hardware)

Até iniciarmos a secção que diz respeito ao FIFO, estávamos motivados em fazer a implementação em breadboards, mas após conclusão do FIFO essa ideia foi eliminada, isto porque só a implementação do FIFO iria requerer muitas ligações entre componentes e antevendo o tempo que poderíamos desperdiçar na implementação do trabalho, tendo em conta que ainda faltava implementar os restantes componentes, optámos por utilizar placas de circuito impresso para execução deste projecto.

Deste modo, o nosso trabalho foi reduzido para 15 ligações:

- 7 ligações para o Input Port
- 4 ligações para o Output Port
- 3 ligações para o Caixa Didáctica

A nossa solução fica assim disposta



Área de Software:

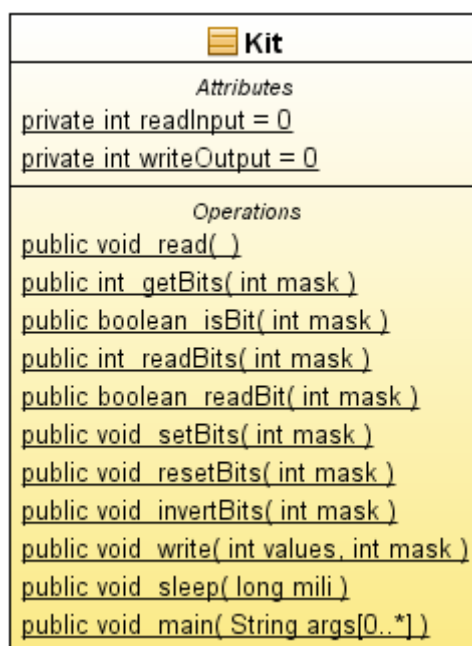
Implementação do acesso ao Kit Didáctico e com o display LCD

Acesso ao Kit Didáctico

Para comunicação com o Kit didáctico, foi necessário adicionar à biblioteca do *Eclipse* o driver USBPort fornecido na página (oficial) da Cadeira de LIC. Este driver fornecia as classes *isel.leic.utils* e *isel.leic.usbio* que fornecem métodos de acesso de leitura e escrita nos portos Input/Output do Kit didáctico.

Para iniciar o projecto foi necessário alterar as classes fornecidas para implementar os métodos solicitados. A primeira classe alterada foi a classe *Kit.java*, que serve de modelo de acesso directo aos portos (utilizando os métodos do driver) e também com métodos de 'análise' dos portos.

O diagrama UML seguinte, demonstra os métodos implementados:



Esta classe será a ponte de comunicação entre o Kit e o Controlo Central.

Acesso ao Display LCD


Para a implementação desta classe, decidimos objectivar o problema separando as suas funções. Implementamos duas classes, uma que implementa o protocolo de comunicação com o Display LCD recebendo os dados de forma genérica e envia em serie para o hardware Controlo do Display, e outra classe que é responsável pela inicialização e comunicação com o Display, mas que utilizará a classe anterior como forma de comunicação.

Esta solução permite-nos alterar no futuro o protocolo de comunicação para outra solução

ou mesmo alterar a comunicação com o Display, sem estarem dependentes um do outro.

Como se pode verificar no UML, foram implementados na classe LcdAccess somente os métodos de inicialização e comandos de funcionamento do Display, enquanto na classe LcdProtocol foram implementados os métodos responsáveis para respeitar o protocolo de comunicação, assim como é visível a relação entre as classes.

 LcdAccess
<p><i>Attributes</i></p> <p><u>private boolean isCentered = false</u></p> <p><u>private boolean isVisible = false</u></p> <p><u>private boolean isBlinking = false</u></p> <p><u>private boolean needRS = false</u></p> <p><u>private int cursorPosition = 0</u></p>
<p><i>Operations</i></p> <p><u>public void entryModeSet()</u></p> <p><u>public void displayControlOn()</u></p> <p><u>public void displayControlOff()</u></p> <p><u>public void blinkOff()</u></p> <p><u>public void init()</u></p> <p><u>private void sleep(int i)</u></p> <p><u>public void clear()</u></p> <p><u>public void clearLine(int line)</u></p> <p><u>public void posCursor(int line, int col)</u></p> <p><u>public void setCursor(boolean visible, boolean blinking)</u></p> <p><u>public void write(char c)</u></p> <p><u>public void write(String txt)</u></p> <p><u>public void writeLine(int line, String txt)</u></p> <p><u>public void setCenter(boolean value)</u></p> <p><u>public int getPos(String s, int b)</u></p> <p><u>private void procValue(int value)</u></p> <p><u>private void writeBits(int value)</u></p> <p><u>public void main(String args[0..*])</u></p>

 LcdProtocol
<p><i>Attributes</i></p> <p><u>private int bits2Shift = 4</u></p>
<p><i>Operations</i></p> <p><u>private void initProtocol()</u></p> <p><u>private void setStart()</u></p> <p><u>private boolean isReady()</u></p> <p><u>private void resetProtocol()</u></p> <p><u>private void delay()</u></p> <p><u>private void fullDelay()</u></p> <p><u>public void sendBits(int rs, int value)</u></p> <p><u>private void sendBit(int value)</u></p> <p><u>public void main(String args[0..*])</u></p>

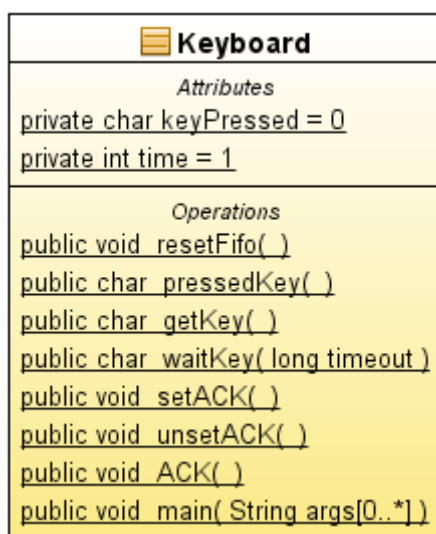
Implementação da comunicação com o Teclado e com o Controlo do Mostrador

Comunicação com o Teclado

A classe responsável pela comunicação com o teclado, irá utilizar a classe Kit como plataforma de recepção da tecla premida e foram implementados os métodos de necessários para obter univocamente a tecla pressionada.

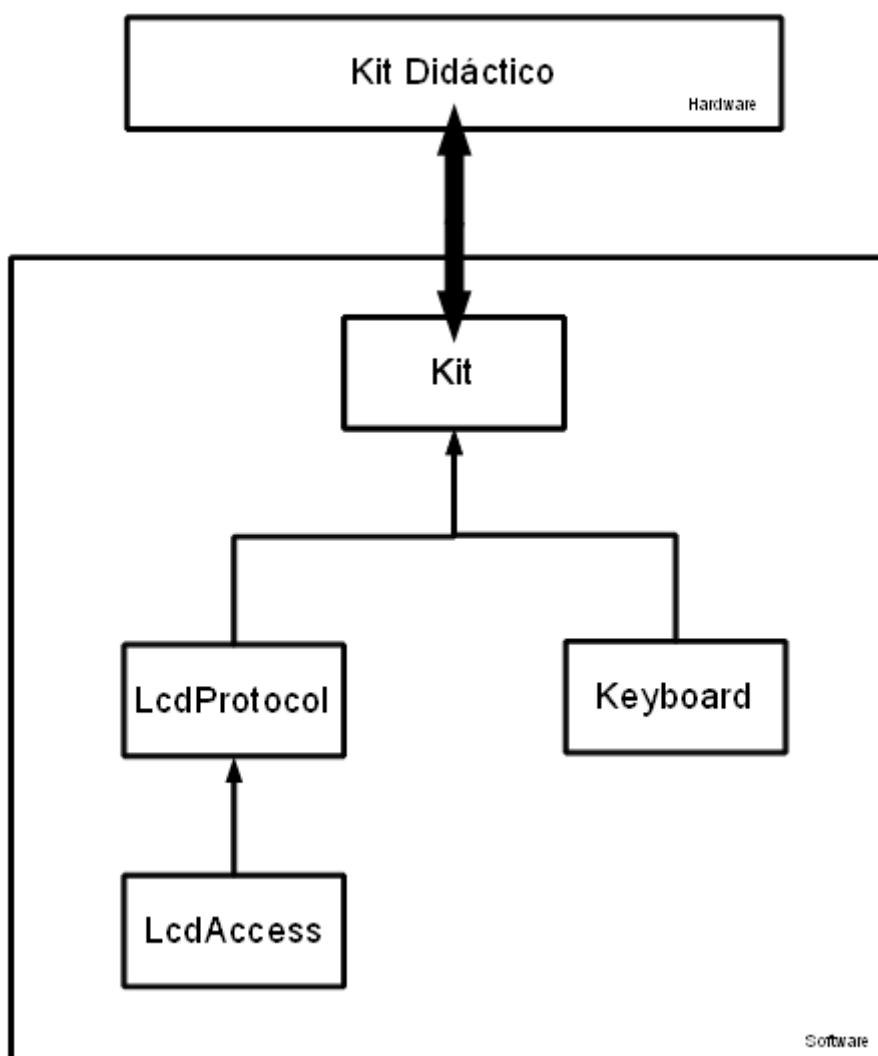
Com a implementação do FIFO (em hardware) houve a necessidade de acrescentar mais dois métodos para garantir o sinal ACK que indicará ao FIFO que já foi processada a tecla guardada.

A nossa solução, segue o seguinte UML:



Comunicação com o Teclado e com o Controlo do Mostrador

Neste momento, estamos em condições de mostrar como toda esta separação de problemas torna o programa modelar, de fácil acesso e de fácil implementação.

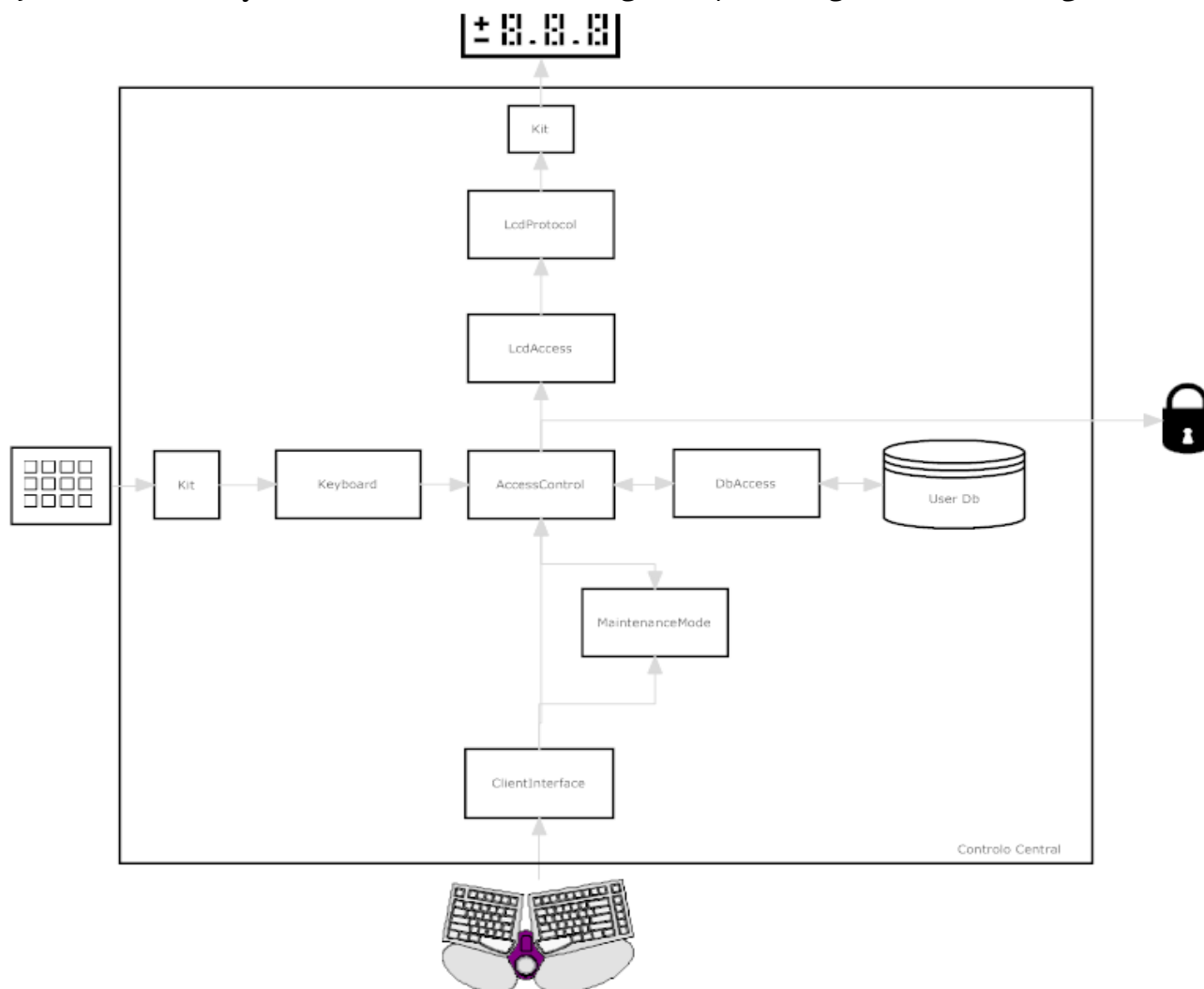


Implementação do interface de operação e manutenção do Controlo de Acessos (Controlo Central)

Nesta altura estamos aptos para prosseguir para uma interface de Controlo Central, porém surgem-nos algumas questões. Para cumprir com os requisitos indicados, vemos a necessidade de resolver os seguintes problemas:

- Garantir comunicação com o kit num único local, para evitar acessos múltiplos e repetição de código
- Criação e manutenção da nossa base de dados, assim como o seu conteúdo
- Criação de uma interface gráfica (modo consola) que permitisse a interacção com o utilizador
- Garantir a Segurança no modo de acesso á área de Manutenção.

Para ultrapassar estas situação, decidimos manter o pensamento “Um Problema, Um Objecto, Uma Solução” e estruturamos um diagrama para diagnosticar “*The Big Picture*”.



Podemos verificar pelo diagrama, que vamos necessitar de mais umas classes para garantir a característica modelar da solução.

Começamos por criar a classe User, que serve para identificar cada Utilizador. Claro está que esta classe só por si, serve para caracterizar um utilizador fornecendo métodos de atribuição e de obtenção de dados do mesmo. Vamos precisar de uma estrutura de dados para iterar por outros utilizadores e garantir a estabilidade e consistência da mesma.

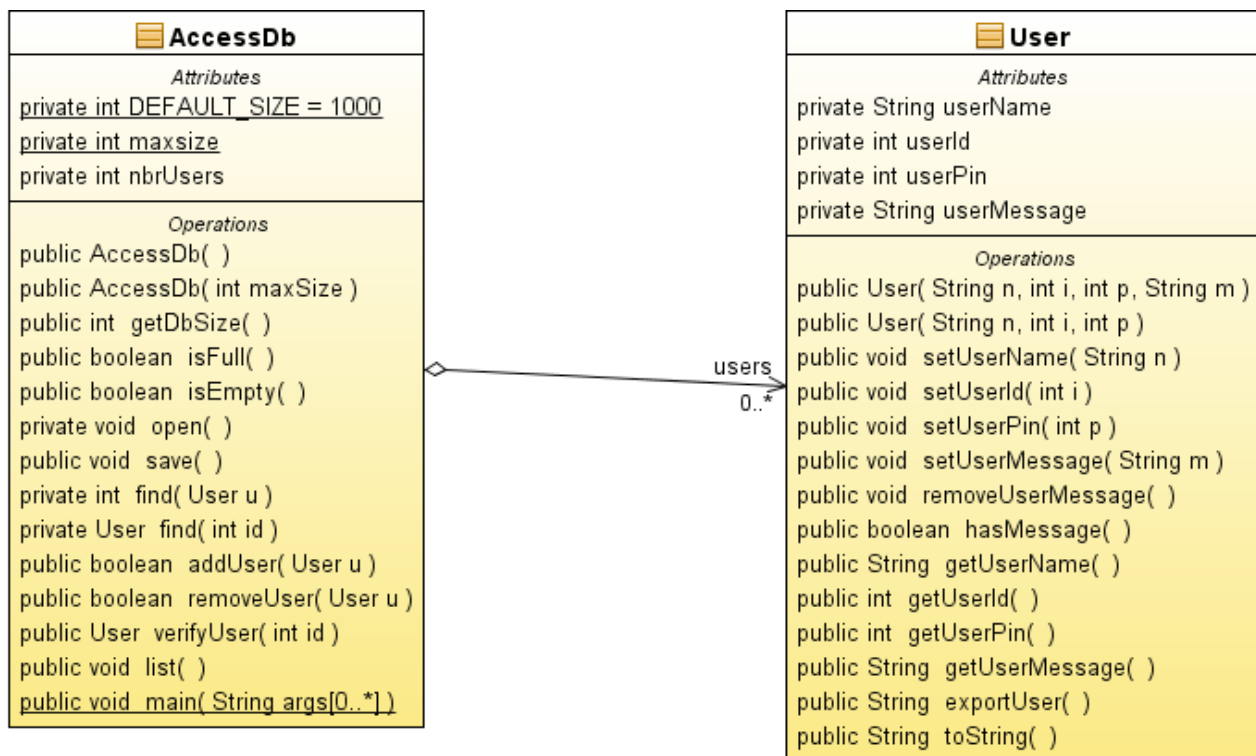
Para tal criámos a classe AccessDb, que será o objecto responsável por garantir que a unicidade dos elementos, assim como fornecendo métodos para aceder aos dados. Esta classe tem no seu descritivo de funções os métodos de acesso, adição e remoção de dados.

Toda a estrutura de dados é obtida/guardada num ficheiro de texto, porém é fácil a alteração para suportar uma outra estrutura de dados.

Optámos por uma estrutura LinkedList que nos permite modular a solução sem grande custo para o computador.

Outra opção que tomámos foi não indexar o numero de utilizador à posição da estrutura. Esta opção garante a modularidade ao administrador para poder indexar o utilizador ao numero de empregado (p. ex.).


A nossa solução segue o seguinte UML:




Ao ultrapassar o obstáculo da estrutura de dados, sentimos a necessidade de ter classes que controlarão a interactividade entre os utilizadores/administradores e o nosso sistema através da introdução dos dados via teclado matricial (os utilizadores) ou através de um computador (os administradores).

Esta interacção é garantida pela classe `AccessControl` que irá controlar os dados recebidos com os dados consistentes na Estrutura de Dados e dará acesso (ou não) à área reservada. Outra classe necessária é a manutenção da Base de dados, controlada por `MaintenanceMode`.

Seguem em seguida os UML:

 AccessControl
<i>Attributes</i> private char controlChar private boolean aShow = false private boolean cancelFlag = true
<i>Operations</i> public AccessControl() private String greetings() private void writeMessage(String txt) private void write(char c) private void clearNbr(String s, int max, int i) private void cancel(int s) private void clear() private void setCenter(boolean b) private void writeLine(int line, String text) private void posCursor(int line, String text, int max) private int getNbr(int i) private boolean verifyPin(User user, int pin) private void openDoor() private void closeDoor() private int newPin() private int ask4NewPin() private int confirm4NewPin() private boolean changePIN(User u) public void close() public boolean operationAccess() private void giveAccess() private void sleep(long time) public int ask4User() public void resetShow() public int ask4Pin(String name) public int ask4Pin(String name, String message) private int ask4PinLcd() public AccessDb getDB() public void main(String args[0..*])

 MaintenanceMode
<i>Attributes</i> private String adminPassword private boolean au = false
<i>Operations</i> public MaintenanceMode(AccessDb ourDB) private void setPassword(String pw) private void getPassword() private void changePwd() public boolean isLocked() private boolean verify(char pw[0..*]) private boolean verify(char pw1[0..*], char pw2[0..*]) private boolean auth() private void maintenaceMainMenu() private int askUserId(String m) private String askUserName(String m) private void showMessage(String m) private int askUserPin(String m) private String askUserMessage(String m) private String confirmation(String s) private boolean option(String o) public void addUserMenu() public void removeUserMenu() public void addMessageUser() public void searchUserMenu() public void listUsersMenu() public void clearScreen() public void exit() public void mainMenu() public void main(String args[0..*])

Ao longo deste projecto um dos inconvenientes desde cedo foram os valores das variáveis que tínhamos em todas as classes, variáveis as quais eram utilizadas por mais do que uma classe. Para resolver este problema criámos uma classe para servir de “armazém” para as variáveis, sendo estas Constantes para todas as classes utilizadas.

Esta solução permitiu-nos ganhar tempo no desenvolvimento, uma vez que as constantes passavam desde já a estarem centralizadas, e a alteração do seu valor seria somente efectuada numa única classe.

LicConstants	
Attributes	
public int RxRDY_MASK = 0x80	
public int KEY_VAL_MASK = 0x40	
public int KEY_LOCK_MASK = 0x10	
public int KEY_DAT_MASK = 0x0F	
public int ACK_MASK = 0x10	
public int DOOR_MASK = 0x20	
public int RxC_MASK = 0x02	
public int RxD_MASK = 0x01	
public String ASK4USER = "Insert Number ___"	
public String ASK4PIN = "Insert PIN ---"	
public String ASK4NEWPIN = "New PIN ---"	
public String CONFIRMNEWPIN = "Confirm PIN ---"	
public String OK_MESSAGE = "OK"	
public String ERROR_PIN_MESSAGE = "Invalid PIN"	
public String SALUTATION[0..*] = {"Good Morning", "Good Afternoon", "Good Evening"}	
public String DOOROPEN = "Door is Open!"	
public String CLOSEDOR = "System Locked!"	
public String INVALIDUSER = "Invalid User!"	
public String TIMEOUT = "Timeout!"	
public String INVALID_KEY = "Pin Mismatch!"	
public String MAINTENANCE_MESSAGE = "Out off Service"	
public int PINPOS = 11	
public int MAXDIGIT = 3	
public int IDNBR = 3	
public int CHECKUSER = 0	
public int CHECKPIN = 1	
public int CHECKNEWPIN = 2	
public int CONFIRMPIN = 3	
public String ERROR_USER_MESSAGE = "Invalid User ID"	
public int CLOCK_MASK = 1	
public int HALFCLOCK_MASK = CLOCK_MASK / 2	
public int SHIFT_BITS_MASK = 1	
public char KEYS[0..*] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'}	
public long WAIT = 200	
public int RS_MASK = 0x40	
public int ENABLE_MASK = 0x80	
public int DATA_MASK = 0x0F	
public int NIBBLE_SHIFT_MASK = 4	
public int CLEAR_MASK = 0x01	
public int ENTRY_SET_MASK = 0x06	
public int RETURN_HOME_MASK = 0x02	
public int CURSOR_ON_MASK = 0x0A	
public int CURSOR_OFF_MASK = 0x0B	
public int DISPLAY_ON_MASK = 0x0C	
public int DISPLAY_OFF_MASK = 0x08	
public int BLINK_ON_MASK = 0x09	
public int BLINK_OFF_MASK = 0x08	
public int SHIFT_MASK = 0x04	
public int ADDR_COUNTER_MASK = 0x80	
public int DISPLAY_SIZE_MASK = 16	
public int VALUE_MASK = 0x0F	
public int MAXTIMERISE = 350	
public int MAXTIMEFALL = 400	
public int SLEEP = 1	
public int STARTMASK = 0xFF	
public int EIGHTBITS = 0xFF	
public String configFile = "user.netrc"	
public int MAINSLEEP = 100	
public String FILEPATH = "users.txt"	
Operations	

Conclusão

É de comum acordo que este projecto deu-nos muito prazer em realizar. Foi magnifico implementar os conhecimentos obtidos de cadeiras anteriores (LSD e PG) e executar um projecto que no início não sabíamos como começar.

Ao longo do semestre foram imensos os desafios lançados, por nós aceites e alcançados. De todo o projecto a implementação da PAL que controla o LCD e a PAL do FIFO foram as que mais tempo nos tomaram, chegando a um ponto que achávamos que não iríamos alcançar os objectivos.

No que diz respeito ao software, este foi “uma brisa num dia de praia”, cuja a excepção foi a implementação do classe que comunica com o LCD, cuja dificuldade foi ultrapassada pela chamada de atenção de uma sequência de inicialização que estava no datasheet que nós não reparámos.

Outro pormenor que nos deu maior alegria, foi utilizar placas de circuito impresso para a implementação em hardware, que para além de simplificar a implementação tornou o trabalho mais apresentável.

Bibliografia

- <http://www.wikipedia.com>
- <http://www.alldatasheet.com>
- <http://www.deetc.isel.ipl.pt/sistemasdigitais/LIC>
- <http://http://www.cypress.com>

Reportório SVN de apoio à cadeira:

svn checkout <http://lic-sv-08-09.googlecode.com/svn/trunk/> lic-sv-08-09-read-only

Agradecimentos

Agradecemos ao Prof. Pedro Sampaio, pelo suporte que nos deu ao longo do semestre e de todos os desafios que nos lançou.