ISEL | DEETC

Sistemas Embedidos I

Trabalho Práctico

Semestre de Inverno de 2010/2011

Autores:
30505 – Telmo Pinto
31401 – Nuno Cancelo
33595 – Nuno Sousa

Indície

Introdução

Este relatório irá acompanhar a execução do trabalho de Sistemas Embebidos I que irá ser realizado ao longo do semestre.

O objectivo desta unidade curricular passa por desenvolver uma aplicação para funcionar no "target" LPC-H2106 (LPC) com micro-controlador LPC2106 da NXP com arquitectura ARM.

Para comunicação entre o target e o host (irá ser utilizado um PC com Linux) será efectuada através da interface USB-JTAG (JTAG) do fabricante DLP Design.

Ao longo do semestre serão entregues novas actividades a cumprir para nos ambientarmos ao ambiente de desenvolvimento e à programação do micro-controlador. O trabalho final sairá da junção destas tarefas.

Este relatório será actualizado ao longo do semestre consoante forem sendo terminadas as várias actividades e nele serão reflectidas as opções tomadas em cada fase do projecto.

1ª actividade – Ambiente de trabalho

Ligações e hardware utilizado

Nesta actividade o objectivo é montar o módulo LPC e o módulo JTAG numa *breadboard*, efectuar as ligações necessárias ao funcionamento do LPC, verificar o correcto funcionamento do módulo e o desenvolvimento de um pequeno programa para ambientação à programação em Assembly ARM, compilação e debug do mesmo.

A ligação entre o host (PC) e o target (módulo LPC) funcionará do seguinte modo:

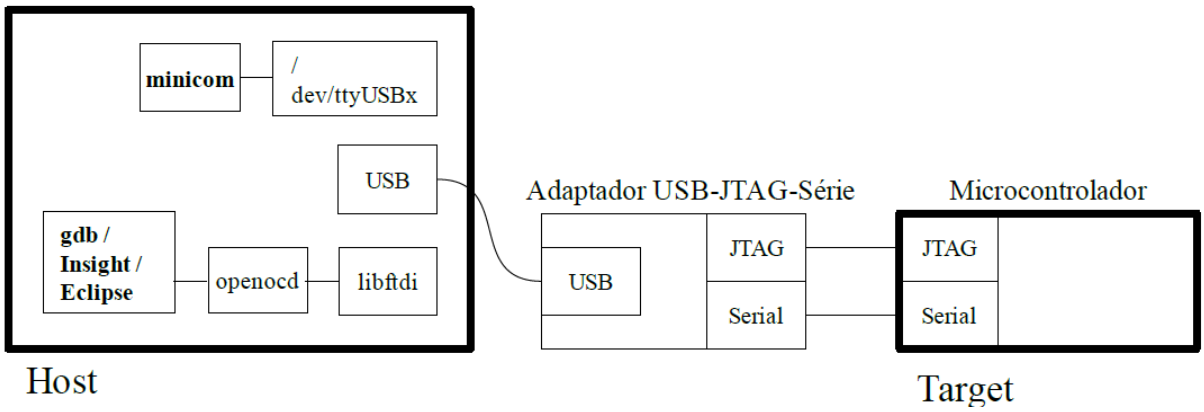

Figura 1: Ligação do Host (PC) ao Target (LPC)

A ligação dos componentes na placa foi efectuada sem dificuldades seguindo o esquema eléctrico fornecido. Por experiência de unidades curriculares anteriores as montagens em *breadboard* funcionam muito bem quando são necessárias alterações às ligações mas podem ao mesmo tempo levar a ter vários problemas de maus contactos. Por essa razão iremos utilizar um placa de circuito impresso pré-perforada que permitirá ter as ligações sem maus contactos e, ao mesmo tempo, permite a evolução do esquema de ligações nas actividades seguintes.

O esquema eléctrico inicial é o seguinte:

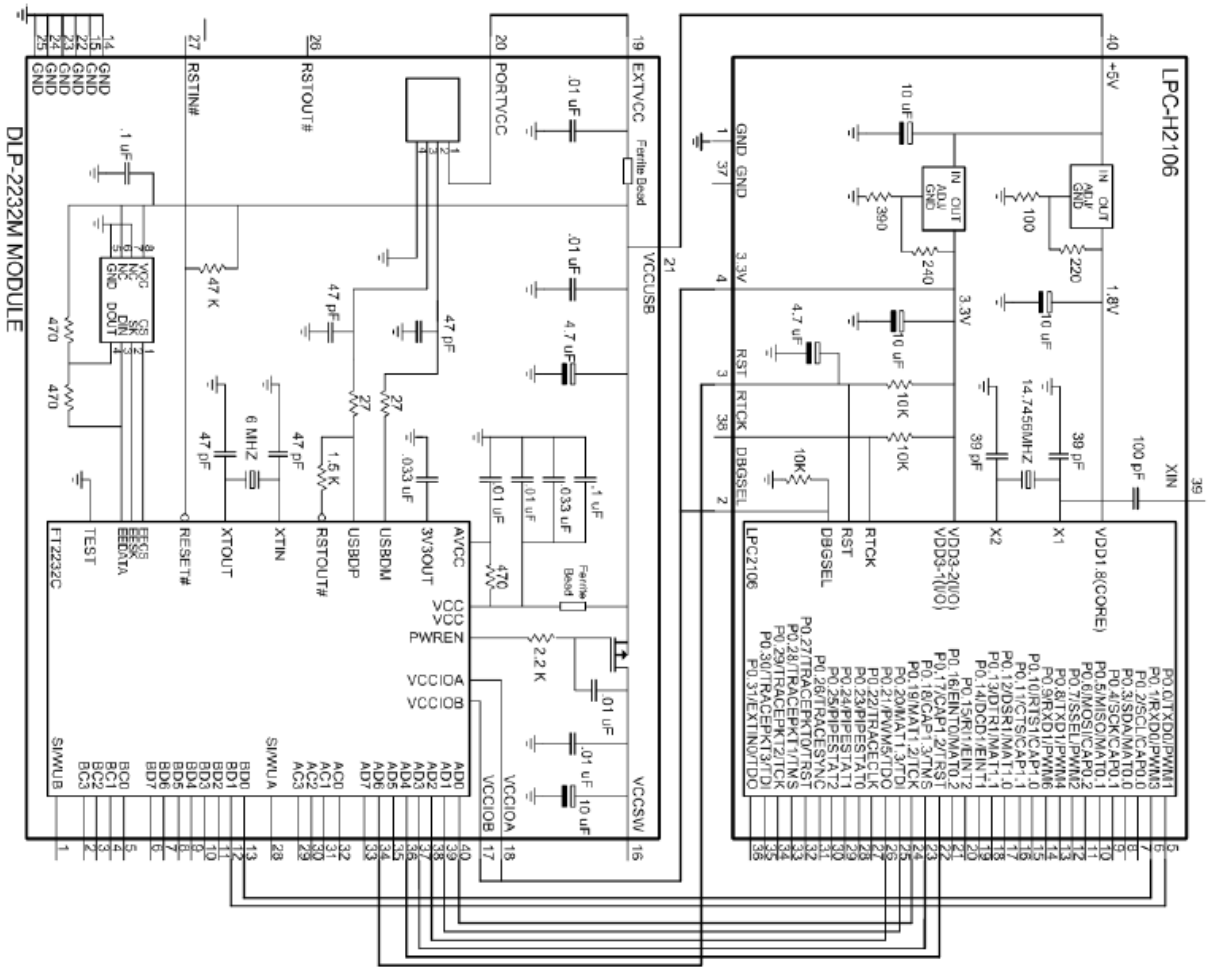

Figura 2: Esquema Eléctrico

Configuração do host

Na configuração do host encontrámos alguns problemas, primeiro pela placa USB-JTAG não estar a ser detectada por um dos hosts utilizados e depois na comunicação com o módulo usando o **openocd** devido a estarmos a utilizar uma versão mais antiga do mesmo. Depois de actualizarmos para a versão 0.40 a comunicação com os módulos já funcionou perfeitamente.

As notas a reter relativamente à compilação de programas para utilizar no LPC passam por ter de se utilizar um ficheiro de script na *linkagem* do programa para que este fique posicionado nas posições de memória correctas para utilização no módulo e para que as instruções de código fiquem alinhadas em memória uma vez estas são sempre codificadas em 32 bits e que o módulo LPC necessita estejam alinhadas.

Questões

- Qual a dimensão de memória ocupada pelo programa?

  Cada instrução ocupa 4 bytes pelo que a memória ocupada pelo programa será 4 x <nº instruções>. Neste caso sendo 11 instruções serão 44 bytes.

- Justifique o valor -20 que aparece na instrução **ldr r0, [pc, #-20]** resultante da escrita de **ldr r0, valor1** no ficheiro fonte.

  O "-20" vem da distância entre o PC que se encontra na instrução ldr e a localização da variável valor1. Neste caso, estando "5 linhas de código" antes da posição do PC e sendo 4 bytes por instrução/linha: 4 x 5 = 20.

- Porque é que quase todas as instruções têm um código com o valor hexadecimal 'E' no dígito de maior peso? E porque é que há uma que tem 'C'?

  Os 4 bits de maior peso da codificação das instruções correspondem aos códigos de condição de execução das instruções. Todas as instruções podem ser executadas condicionalmente e as condições são testadas antes da execução da instrução. Neste código temos apenas uma instrução condicional (bgt – branch greater than) em que a codificação do Greater Than (GT) é 1100, ou seja, 'C'. No caso das instruções incondicionais (AL – always) a codificação é 1110, ou seja 'E'.

- Porque é que na execução do comando **x /10i 0x4000000** não apareceu a última instrução do programa?

  Porque a opção /10i apenas mostra apenas as primeiras 10 posições de memória a partir do endereço indicado. Neste caso ocupamos 11 posições de memória pelo que a última instrução não aparece.

- Modifique o programa para que este determine o menor dos dois números. Verifique.

  O programa para determinar o maior de dois números será igual com excepção do salto condicional que passa de **bgt** para **ble**. Para comodidade de leitura mudámos as labels de maior para menor. Código encontra-se na pasta exe1.

- Faça um programa para ordenar uma sequência de valores.

  Para executar este programa usámos as funções criadas nas questões anteriores para verificar o maior de dois números e fazer as trocas necessárias à ordenação.

2ª actividade – LED e teclado

Ligação do teclado

Para efectuar as ligações para ler do teclado existem várias hipóteses tanto na escolha dos pinos a usar como na implementação do código para ler as teclas do mesmo.

Na escolha dos pinos a usar, descartámos imediatamente a utilização dos sinais P0.14 (por ser utilizado para efectuar reset ao LPC) e os sinais P0.2 e P0.3 por serem *open colector*.

Assim, para manter os sinais seguidos por facilidade no uso das máscaras durante a implementação optámos por usar os sinais P0.4 a P0.7 (GPIO 0.2 e 0.3)para as colunas do teclado e os P0.8 a P0.11 (GPIO 0.4 e 0.5) para as linhas. Estes sinais correspondem aos pinos 9 a 12 e 13 a 16 do módulo LPC.

O LED irá ser ligado ao pino 17 do módulo correspondente ao sinal P0.12.

Juntamente com a ligação do teclado será necessário colocar resistências para efectuar um *pull-up* ou *pull-down* do sinal de modo a ser detectado pelo controlador. Após análise do mesmo e discussão do assunto na aula percebemos que efectuar *pull-up* com resistências de grande valor o seria mais aconselhável de modo a que a corrente nos portos do controlador fosse baixa evitando o aquecimento do mesmo.

Devido ao algoritmo escolhido para ler o valor do teclado (ver mais adiante) foi necessário colocar resistências de *pull-up* em todos os pinos do teclado. O valor encontrado para estas, após verificar que o valor máximo típico nos pinos do controlador é de cerca de 50 mA, foi de 66kOhm. Utilizámos resistências de 50kOhm que permitem compensar alguma resistência das próprias ligações da *breadboard* e garantir ao mesmo tempo o bom funcionamento do LPC.

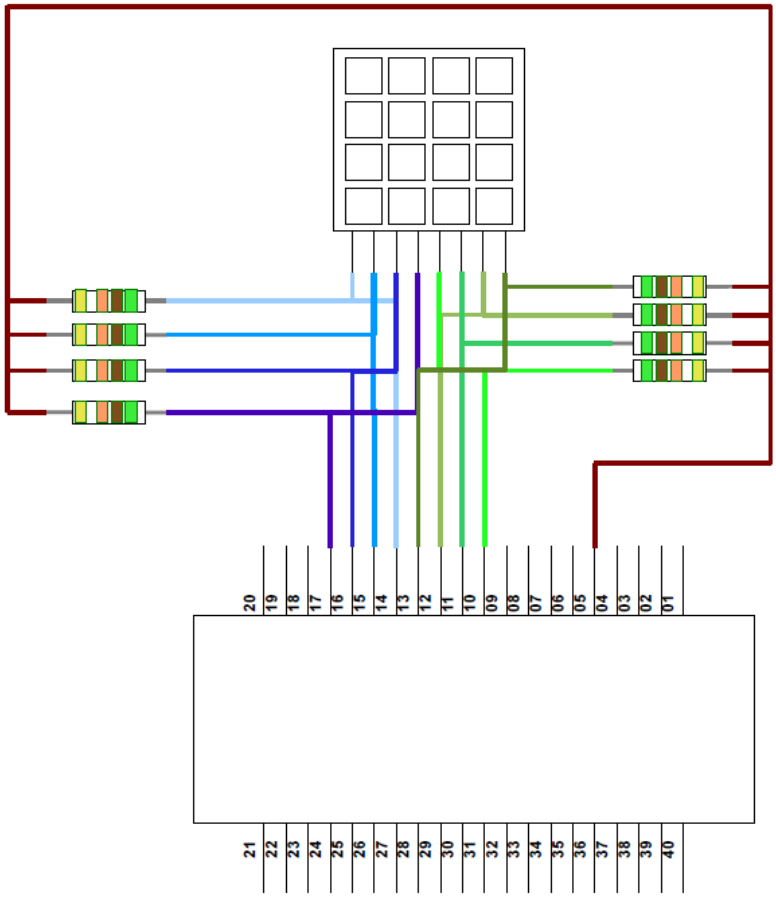

Illustration 1: Ligação do teclado

Activação e configuração portas IO do LPC

Os pinos do LPC são partilhados por diversas funções (UART, Timer, ligação ao módulo JTAG, GPIO, etc) pelo que temos de escolher que qual a função que pretendemos e configurar no registo adequado.

Ao colocar 0x00000000 no endereço 0xE002C000 configuramos os portos P0.0 a P0.15 como GPIO.

Os endereço 0xE0028008 (IODIR), 0xE0028000 (IOPIN), 0xE0028004 (IOSET) e 0xE002800C (IOCLR) são utilizados para definir os pinos de *input* e *output*, ler o estado dos pinos, colocar os pinos a 1 e colocar os pinos a 0 respectivamente.

Algoritmo de leitura do teclado

Na escolha do algoritmo a utilizar a solução que mais rapidamente surgiu foi a de implementar em software a leitura de um teclado por hardware utilizado em unidades anteriores: colocar 1 do lado das colunas e criar um contador para colocar um sinal 0 em cada uma das colunas por ordem e ir lendo as linhas (com

um segundo contador) por ordem até encontrar uma tecla pressionada.

Uma vez que podemos facilmente verificar se existe tecla pressionada (e qual o valor da mesma) escrevendo nas colunas em simultâneo e lendo todas as linhas também em simultâneo e, uma vez que todos os portos de IO do módulo podem ser definidos programaticamente para serem Input ou Output conforme for necessário optámos por uma abordagem diferente.

Uma vez que optámos por *pull-ups* em todos os pinos do teclado, escolhemos as colunas como *output* e as linhas como *input* e forcámos 0 em todas as colunas. Ao ler as linhas, se o resultado for 1111 não existe tecla pressionada. Se uma tecla pressionada uma das linhas terá valor 0. Aí, trocamos os pinos de *input* com os de *output* e colocamos as linhas a 0. Lendo o valor das colunas surgirá um dos quatro valores a 0. Neste momento já temos a linha e a coluna a que pertence a tecla pressionada e podemos proceder à descodificação da mesma.

Com este algoritmo o que obtemos é um *bitmap* com a tecla que foi pressionada. Isto permite rapidamente verificar se é uma tecla nova ou se a mesma foi mantida pressionada guardando o *bitmap* e comparando com o novo *bitmap* lido. Poderá permitir, também, dar duplas funções às teclas consoante o tempo que elas forem pressionadas. Ao pressionar uma tecla devolve o valor da mesma, mas se for pressionada durante *x* tempo poderá devolver um bitmap diferente que corresponde a uma acção também ela diferente.

Esta vantagem torna mais fácil evitar o *bounce* típico da leitura do teclado quando este alimenta um *buffer* de teclas pressionadas. Se o *bounce* não for tido em conta o *buffer* irá encher todo com a mesma tecla o que obriga a esperar que a tecla seja libertada para ser enviada para processamento. No nosso caso, ao guardarmos o *bitmap* da tecla, retornamos imediatamente o valor e ao entrar novamente no programa de leitura do teclado verificamos se a tecla continua pressionada e só retornamos novo valor após não termos tecla nenhuma e uma nova (ou a mesma) ser pressionada novamente.

Uma vez que o programa está a funcionar de modo sequencial, quando as tarefas a executar com o valor da tecla demoram algum tempo (como é o caso do led a piscar) se libertarmos a tecla e voltarmos a pressionar este assume que a mesma não foi libertada pelo que fica a aguardar nova tecla. Quando o programa de leitura funcionar em modo independente do resto do código (tendo um buffer a receber as teclas pressionadas) já irá funcionar correctamente.

Uma alteração a efectuar posteriormente será guardar a tecla anterior na memória em vez de utilizar um registo para que esta não se perca e possa ser consultada em qualquer altura e por qualquer função que necessite.

Inicialização da área de suporte a excepções

Criámos um ficheiro init.S que inicializa o espaço reservado para o tratamento de excepções. Mais tarde, conforme seja necessário, será actualizado com o tratamento das mesma, ou seja, a instrução ***B .*** será substituída pelo código de tratamento da excepção ou o salto para o código que faz esse tratamento. Neste momento, este código permite que, no caso de ser lançada uma excepção, saber qual foi a excepção lançada uma vez que o código irá ficar parado no local de tratamento da mesma.

```
/* LPC2606 INITIALIZATION */

/* STARTUP SECTION - VECTORS INITIALIZATION*/
.global _START
.text

        LDR PC, RESET_ADDR
        LDR PC, UNDEF_INST_ADDR
        LDR PC, SOFT_INTER_ADDR
        LDR PC, INST_MEM_FAULT_ADDR
        LDR PC, DATA_MEM_FAULT_ADDR
        LDR PC, CHECKSUM
        LDR PC, IRQ_ADDR
        LDR PC, FIQ_ADDR

        RESET_ADDR:                     .word START
        UNDEF_INST_ADDR:        .word UNDEF_INST
        SOFT_INTER_ADDR:        .word SOFT_INTER
        INST_MEM_FAULT_ADDR:    .word INST_MEM_FAULT
        DATA_MEM_FAULT_ADDR:    .word DATA_MEM_FAULT
        CHECKSUM:                               .word 0xB8A06F58
        IRQ_ADDR:                               .word IRQ
        FIQ_ADDR:                               .word FIQ


/* START */
_START:
START:
                B MAIN
                B .

/***********************/
/* EXCEPTIONS HANDLING */
/***********************/

UNDEF_INST:
                B .
SOFT_INTER:
                B .
INST_MEM_FAULT:
                B .
DATA_MEM_FAULT:
                B .
IRQ:
                B .
FIQ:
                B .
```

Code 1: Inicialização da área de Vectores (tratamento de excepções)

Gravar o programa na ROM

Para colocar o programa na ROM em vez da RAM foi alterado o script *ldscript* para indicar a posição da memória onde deverá ser gravado o mesmo.

```
ENTRY(_start)

MEMORY
{
        ram : ORIGIN = 0x40000000, LENGTH = 0x10000
        rom : ORIGIN = 0x0, LENGTH = 0x20000
}

SECTIONS
{
        .text : {
        __text_start__ = ABSOLUTE(.);
        *(.text*) *(.glue_7) *(.glue_7t);
        __text_end__ = ABSOLUTE(.);
        } > rom
}
```

Code 2: ldScript para gravação do programa na ROM

Após efectuar o reset no P0.14 o programa ficou a funcionar directamente da ROM, sendo necessário neste momento ligar apenas à porta USB para receber a alimentação.

Conclusão

Bibliografia

Vários PDF's disponibilizados no Moodel