Sistemas Embebidos I

Semestre de Inverno de 2010/2011

Ambiente de trabalho

1. Introdução

No desenvolvimento de aplicações para sistemas embebidos há duas máquinas envolvidas: o *host* e o *target*. O *host* é normalmente uma estação de trabalho genérica – Windows, Linux ou outro, com ligação à Internet e capacidade de memória e de processamento para instalação e execução das ferramentas de desenvolvimento. Quanto ao *target* a diversidade é muita, pode ir de pequenos microcontroladores a computadores com capacidades iguais ou superiores às do *host*.

O ambiente de trabalho depende portanto do tipo de *host* e do tipo de *target*. Em SE1 vais ser usado, como *target,* a placa LPC-H2106 da Olimex, com microcontrolador LPC2106 da NXP com arquitetura ARM, e como *host* computador PC com Linux.

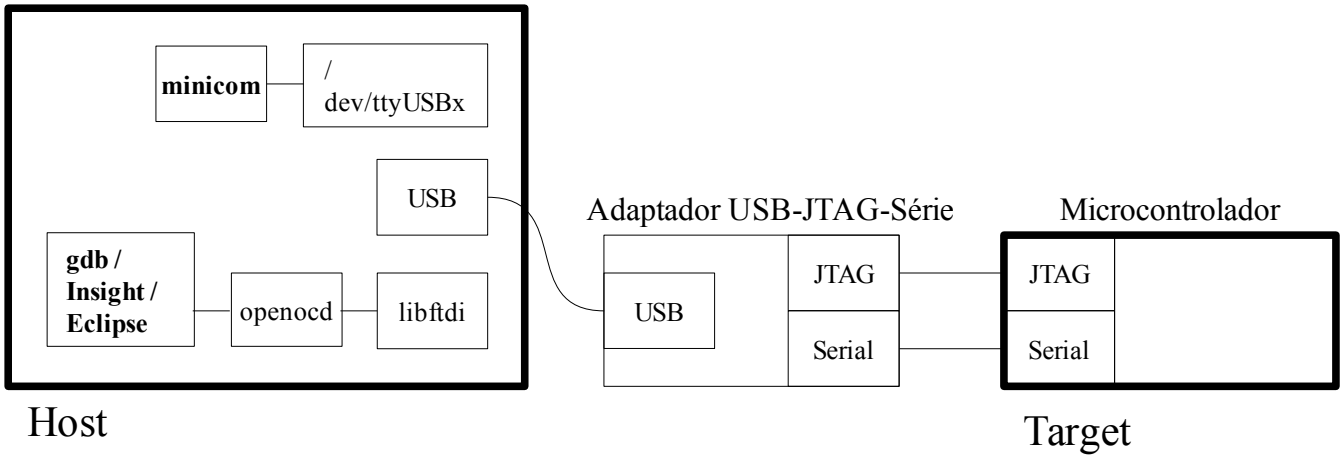
Para a geração dos programas são propostas as ferramentas da GNU para a arquitetura ARM.

A depuração dos programas é realizada a partir de um *debugger*. Este programa permite transferir o programa em teste na memória do *target,* controlar a sua execução e observar resultados intermédios. O *debugger* proposto é o Insight.

O acesso ao microcontrolador é feito através da interface JTAG. Por esta interface é possível, entre outras coisas, transferir dados de/para a memória do *target* e controlar a execução do processador.

O OpenOCD é um projecto *open source* que visa a utilização de dispositivos de baixo custo no acesso à interface JTAG, actualmente disponível em muitos microprocessadores.

O programa **openocd** encarrega-se de adaptar o protocolo GDB às especificidades do *hardware* utilizado, realizando no *target* as operações pretendidas pelo *debugger.*



Para servir como *host* está disponível uma imagem para máquina virtual VMware com as seguintes ferramentas instaladas:

- **ferramentas GNU**
  - **arm-eabi-gcc** – tradutor de linguagem C para código máquina;

- **arm-eabi-ld** – ligador de módulos e localizador de programas;
- **arm-eabi-as** – tradutor de linguagem assembly para código máquina;
- **arm-eabi-gdb** – *debugger* com interface de linha de comandos;
- **arm-eabi-insight** – *debugger* com interface gráfica;
- **arm-eabi-objdump** - utilitário para inspeção de programas;

- **openocd** – adaptador de protocolo GDB para JTAG-EmbeddedICE;
  - **openocd_lpc2106.cfg** – ficheiro de configuração para o microcontrolador LPC206;
- **gedit ou geany** – editores de texto para programação;
- **gtkterm ou minicom** – simuladores de terminal.

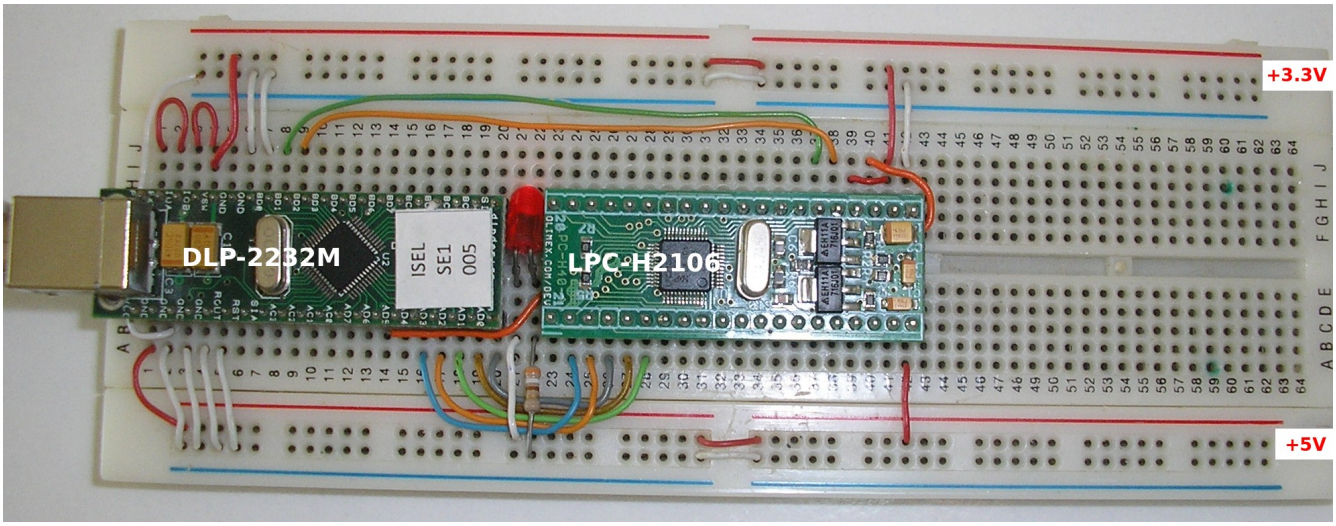
2. Hardware

A ligação do *host* à placa LPC-H2106 é feita através da interface USB-JTAG do fabricante DLP Design, com a referência DLP-2223M.

Esta placa além da interface JTAG suporta também um canal de comunicação série assíncrona que pode ser instanciado como um canal série do *host*.

O *hardware* a utilizar vai ser montado em *breadboard*. Nesta fase monta-se a placa do microcontrolador e a placa de interface USB-JTAG e ligam-se as alimentações, a interface JTAG e o canal série. Em fases posteriores serão acrescentados outros componentes.

Recomenda-se a disposição apresentada na imagem seguinte e a utilização de ligações curtas. O esquema de ligações encontra-se no fim.



Verificação da interface JTAG

Esta verificação é feita com o programa openocd. Este programa actua como um servidor, aceita ligações de programas clientes como o GDB, Insight ou Telnet e executa as operações indicadas por

estes.

Este programa é configurável em relação ao *hardware* do *target* com que vai interagir. No nosso caso é configurado para operar com o microcontrolador LPC2106, da NXP, através da interface USB-JTAG FT2232C da FTDI. A configuração é dada através do ficheiro **`openocd_lpc2106.cfg`** que deve ser colocado na diretoria **`~/se1/code/openocd`**.

Depois de concluída a montagem, ligar o cabo USB ao host e executar o seguinte comando numa janela de comandos, tendo **`~/se1/code/openocd`** como a diretoria corrente.

```
$ openocd -f openocd_lpc2106.cfg
```

Se tudo estiver a funcionar bem será apresentada a seguinte mensagem:

```
Open On-Chip Debugger 0.4.0 (2010-09-08-18:46)
Licensed under GNU GPL v2
For bug reports, read
     http://openocd.berlios.de/doc/doxygen/bugs.html
1000 kHz
trst_and_srst srst_pulls_trst srst_gates_jtag trst_push_pull srst_open_drain
jtag_nsrst_delay: 300
jtag_ntrst_delay: 300
Info : clock speed 1000 kHz
Info : JTAG tap: lpc2106.cpu tap/device found: 0x4f1f0f0f (mfg: 0x787, part: 0xf1f0,
ver: 0x4)
Info : Embedded ICE version 4
Info : lpc2106.cpu: hardware has 2 breakpoint/watchpoint units
Info : JTAG tap: lpc2106.cpu tap/device found: 0x4f1f0f0f (mfg: 0x787, part: 0xf1f0,
ver: 0x4)
Warn : srst pulls trst - can not reset into halted mode. Issuing halt after reset.
target state: halted
target halted in Thumb state due to debug-request, current mode: Supervisor
cpsr: 0xa00000f3 pc: 0x7fffe24c
requesting target halt and executing a soft reset
target state: halted
target halted in ARM state due to debug-request, current mode: Supervisor
cpsr: 0xa00000d3 pc: 0x00000000
dcc downloads are enabled
fast memory access is enabled
```

Verificação do canal série

Estabeleça uma ligação provisória entre os pinos 5 e 6 da placa LPC-H2106, para além das ligações já efectuadas.

Execute o programa simulador de terminal e configure-o para operar sem protocolo. Este programa envia pelo canal série os códigos das teclas premidas no teclado do computador. A ligação anterior faz com que os caracteres enviados sejam retornados para o computador e por consequência afixados no ecrã. Ao remover a ligação verificará que os caracteres digitados deixam de aparecer.

3. diretorias de trabalho

Os programas realizados no âmbito da disciplina de SE1 devem ser guardados numa sub-árvore com a estrutura indicada abaixo. A entrega de trabalhos em formato electrónico – relatórios e código, deve também ser feita segundo esta estrutura.

```
se1
 |
 +-- code
 |    +-- app
 |    |    +-- exe1
 |    |    +-- exe2
 |    |    +-- ...
 |    +-- mylib
 |    +-- openocd
 +-- doc
```

Ao primeiro nível, sob a diretoria **code,** devem ser armazenados os programas e sob a diretoria **doc** os relatórios de trabalhos e outros documentos.

Na diretoria **app,** são criadas sub-diretorias, uma para cada aplicação, onde são guardados os seus dados específicos: ficheiros fonte, ficheiros executáveis, ficheiros intermédios, etc.

Na diretoria **mylib** são guardados ficheiros com declarações (***.h**) e código utilizados em várias aplicações, funcionando como biblioteca privada.

Na diretoria **openocd** são guardados os ficheiros de configuração do openocd.

4. Exercício

4.1 Ficheiro Fonte

Editar o programa seguinte que determina o maior de dois valores numéricos. Pode usar o editor **gedit** ou **Geany**. Grave o ficheiro na diretoria **`exe1`** com o nome **`program.s`**.

```
valor1:   .word    20
valor2:   .word    21
maior:    .word    0

     .global   _start
_start:
     ldr  r0, valor1
     ldr  r1, valor2
     cmp  r0, r1
     bgt  1f
     str  r1, maior
     b    2f
1:
     str  r0, maior
2:
     b    .
```

4.2 Compilação

Para transformar as instruções de mnemónicas legíveis para o código máquina respectivo da arquitetura ARM deve usar o *assembler* **`as`** da GNU.

```
$ arm-eabi-as program.s -o program.o
```

Utilize agora o utilitário **`objdump`** para observar o resultado da compilação.

```
$ arm-eabi-objdump -S program.o
```

A execução deste utilitário produz o seguinte resultado:

```
program.o:     file format elf32-littlearm

Disassembly of section .text:

00000000 <valor1>:
   0:    00000014  .word    0x00000014

00000004 <valor2>:
   4:    00000015  .word    0x00000015

00000008 <maior>:
   8:    00000000  .word    0x00000000

0000000c <_start>:
   c:    e51f0014  ldr  r0, [pc, #-20]; 0 <valor1>
  10:    e51f1014  ldr  r1, [pc, #-20]; 4 <valor2>
  14:    e1500001  cmp  r0, r1
  18:    ca000001  bgt  24 <_start+0x18>
  1c:    e50f101c  str  r1, [pc, #-28]; 8 <maior>
  20:    ea000000  b    28 <_start+0x1c>
  24:    e50f0024  str  r0, [pc, #-36]; 8 <maior>
  28:    eafffffe  b    28 <_start+0x1c>
```

Note que na segunda coluna podem ser observados os códigos máquina das instruções do programa.

4.3 Localização em memória

Este processo permite localizar o programa em endereços de memória válidos para o *target*. A definição destes endereços é explicitada num ficheiro de definições para a ferramenta **`ld`** da GNU (ficheiro **`ldscript`**), que se transcreve de seguida.

```
ENTRY(_start)

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

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

    .data ALIGN(4) : {
    __data_start = ABSOLUTE(.);
    *(.data*);
    __data_end__ = ABSOLUTE(.) ;
    } > ram
```

```
}
```

Por ora não se preocupe com a sintaxe, um pouco intrincada, deste ficheiro. Note apenas nas directivas **ENTRY** e **MEMORY**. Com a primeira define-se o endereço da primeira instrução a executar. Com a segunda define-se o espaço de memória disponível para programas. Começa no endereço **0x40000000**, que corresponde ao endereço base da memória do LPC2106, e tem **0x10000** (64 Kbyte) de dimensão.

Voltando ao exemplo, execute agora o comando seguinte para localizar o programa em memória.

```
$ arm-eabi-ld -T ldscript program.o -o program
```

O resultado da localização pode ser observado invocando o utilitário **objdump** sobre **program.**

```
$ arm-eabi-objdump -S program
```

```
program:     file format elf32-littlearm

Disassembly of section .text:

40000000 <valor1>:
40000000:00000014  .word     0x00000014

40000004 <valor2>:
40000004:00000015  .word     0x00000015

40000008 <maior>:
40000008:00000000  .word     0x00000000

4000000c <_start>:
4000000c:e51f0014  ldr  r0, [pc, #-20]; 40000000 <__text_start__>
40000010:e51f1014  ldr  r1, [pc, #-20]; 40000004 <valor2>
40000014:e1500001  cmp  r0, r1
40000018:ca000001  bgt  40000024 <_start+0x18>
4000001c:e50f101c  str  r1, [pc, #-28]; 40000008 <maior>
40000020:ea000000  b    40000028 <_start+0x1c>
40000024:e50f0024  str  r0, [pc, #-36]; 40000008 <maior>
40000028:eafffffe  b    40000028 <_start+0x1c>
```

A diferença para o passo anterior consiste na alteração dos endereços de memória, que passaram do endereço base **0x00000000** para o endereço base **0x40000000**.

4.4 Teste

Para testar o programa vamos começar por usar o *debugger* GDB da GNU. Durante a disciplina usar-se-há o Insight que automatiza algumas operações e tem interface gráfica de utilizador. O uso do GDB neste exercício visa tornar mais explicito o que acontece quando se faz *debugging* remoto.

```
$ arm-eabi-gdb program
```

```
GNU gdb 6.8
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
```

```
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-eabi"...
(no debugging symbols found)
(gdb)
```

Começa-se por conetar o *debugger* ao *target*. Nesta altura o openocd deve estar a executar tal como descrito anteriormente. O openocd aceita ligação TCP/IP no porto 3333.

```
(gdb) target remote localhost:3333
```

```
Remote debugging using localhost:3333
0x00000000 in ?? ()
```

De seguida, carrega-se o programa na memória do *target* com o comando **`load`**. Chama-se a atenção para o facto de que toda a informação necessária para esta operação está guardada no próprio ficheiro executável (**`program`**). Este ficheiro contém, entre outra informação: *i)* o código das instruções; *ii)* a indicação do endereço em que o programa deve ser carregado; *iii)* o endereço da primeira instrução a executar.

```
(gdb) load
```

```
Loading section .text, size 0x2c lma 0x40000000
Start address 0x4000000c, load size 44
Transfer rate: 3 KB/sec, 44 bytes/write.
```

Podemos agora observar o programa carregado em memória. O comando seguinte permite visualizar 10 posições de memória a partir do endereço definido, sob a forma de instruções Assembly **.**

```
(gdb) x /10i 0x40000000
```

```
0x40000000 <valor1>:    andeq     r0, r0, r4, lsl r0
0x40000004 <valor2>:    andeq     r0, r0, r5, lsl r0
0x40000008 <maior>:     andeq     r0, r0, r0
0x4000000c <_start>:    ldr  r0, [pc, #-20] ; 0x40000000 <valor1>
0x40000010 <_start+4>:  ldr  r1, [pc, #-20] ; 0x40000004 <valor2>
0x40000014 <_start+8>:  cmp  r0, r1
0x40000018 <_start+12>: bgt  0x40000024 <_start+24>
0x4000001c <_start+16>: str  r1, [pc, #-28] ; 0x40000008 <maior>
0x40000020 <_start+20>: b    0x40000028 <_start+28>
0x40000024 <_start+24>: str  r0, [pc, #-36] ; 0x40000008 <maior>
```

O endereço da primeira instrução, correspondente à *label* **`_start`** indicado através da directiva **`ENTRY`**, aquando da localização, foi também carregado no registo **`pc`**.

Visualizar os registos:

```
(gdb) i r
```

```
r0             0xffffffff   4294967295
r1             0xffffffff   4294967295
r2             0xffffffff   4294967295
r3             0xffffffff   4294967295
```

```
r4             0xffffffff   4294967295
r5             0xffffffff   4294967295
r6             0xffffffff   4294967295
r7             0xffffffff   4294967295
r8             0xffffffff   4294967295
r9             0xffffffff   4294967295
r10            0xffffffff   4294967295
r11            0xffffffff   4294967295
r12            0xffffffff   4294967295
sp             0xffffffff   0xffffffff
lr             0xffffffff   4294967295
pc             0x4000000c   0x4000000c <_start>
fps            0x0          0
cpsr           0xa00000d3   2684354771
```

Visualizar as variáveis:

```
(gdb) x /3wx 0x40000000
```

```
0x40000000 <valor1>:    0x00000014     0x00000015     0x00000000
```

Para executar instrução a instrução usar o comando **stepi**.

```
(gdb) stepi
```

```
0x40000010 in _start ()
```

Com **stepi** foi executada a instrução **ldr r0, [pc, #-20]** e o **pc** avançou para a próxima instrução - **0x40000010**.

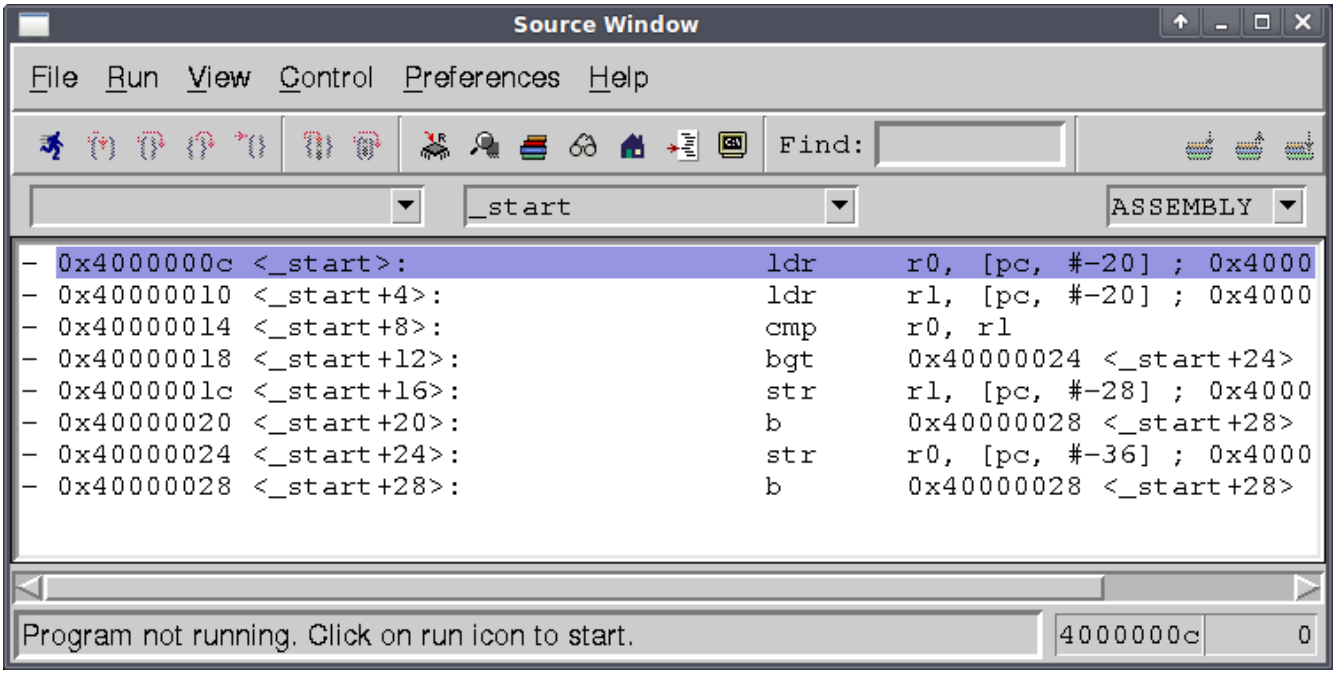
```
(gdb) x /i $pc
```

```
0x40000010 <_start+4>:  ldr  r1, [pc, #-20] ; 0x40000004 <valor2>
```

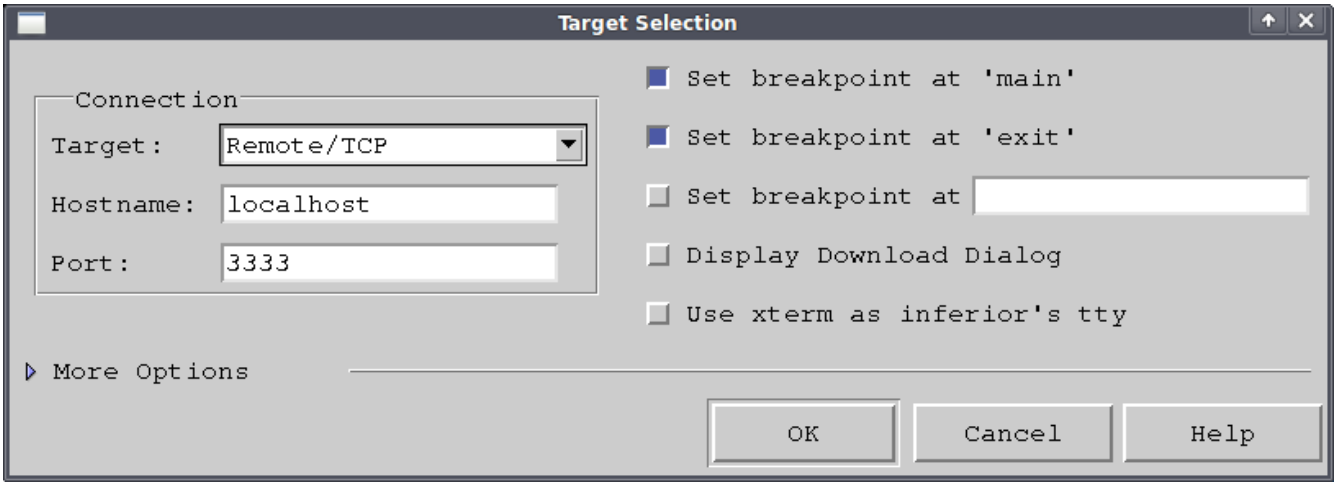
Deve prosseguir com a execução do programa e ir observando o resultado da execução das instruções, nos registos do processador (**r i**) e na memória (**x /3wx 0x40000000**).

Para testar programas de maior dimensão, e escritos em linguagem C, vai usar-se nesta unidade curricular o **Insight**. Este *debugger* comunica com o *target* segundo o protocolo do **GDB** e tem inteface gráfica de utilizador. Este programa foi desenvolvido em *open source* por pessoas das companhias Red Hat e Cygnus (http://sourceware.org/insight/).

```
$ arm-eabi-insight program
```



Da primeira vez que o Insight é executado é necessário definir a ligação ao *target* em File/Target Settings.



Ao carregar no ícone Run, o Insight transfere o programa para a memória do *target*, insere um *breakpoint* na label _start, inicia a execução e pára nesse *breakpoint*. A partir desse momento o Insight entra no estado de execução e passa a aceitar comandos de controlo do processador. Este estado é assinalado pela barra verde horizontal assinalando a próxima instrução a executar.

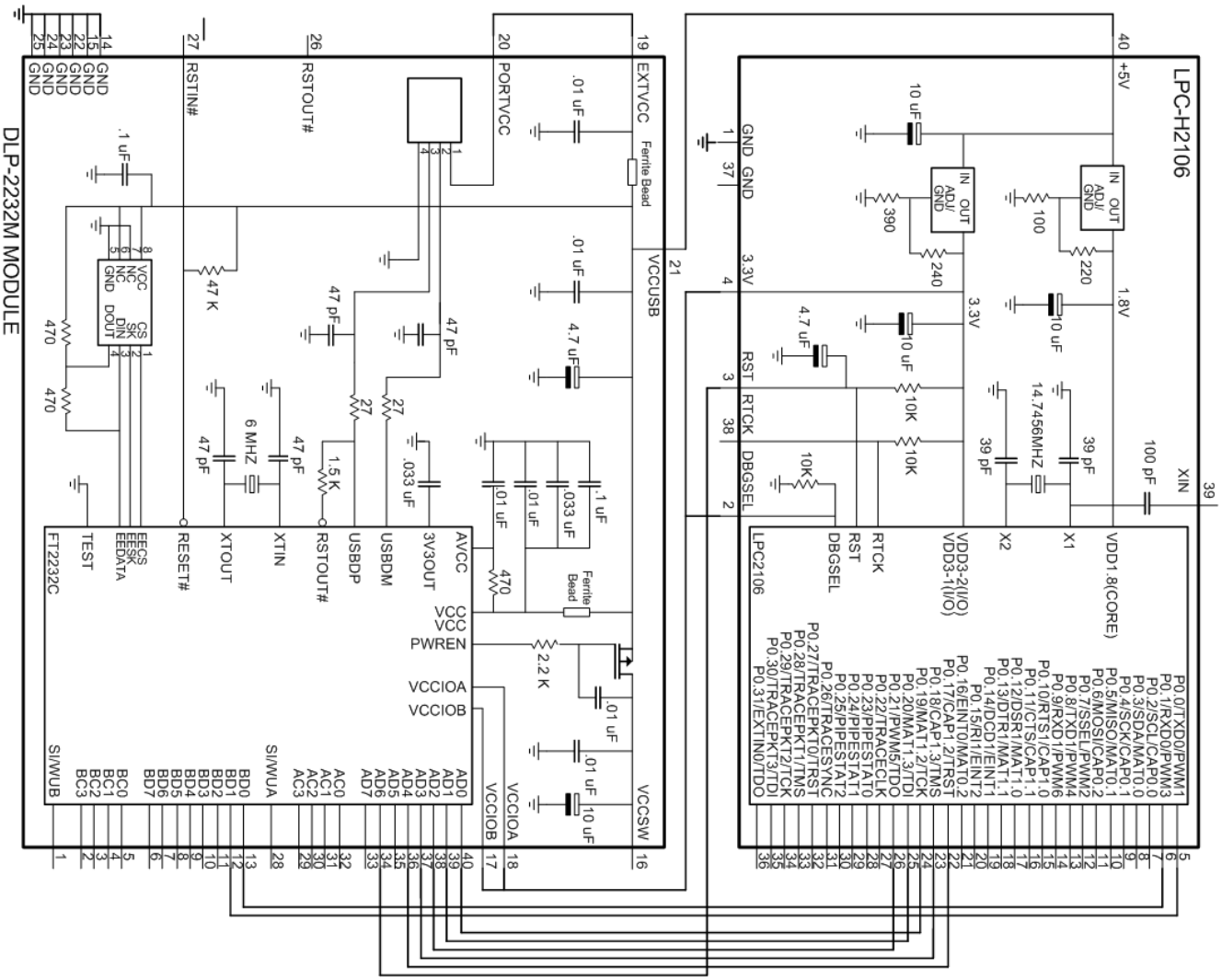
É comum aparecer a seguinte pergunta:

Responda **No**

4.5 Questões

- Qual a dimensão de memória ocupada pelo programa?
- Justifique o valor -20 que aparece na instrução **ldr r0, [pc, #-20]** resultante da escrita de **ldr r0, valor1** no ficheiro fonte.
- 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'?
- Porque é que na execução do comando x /10i 0x40000000 não apareceu a última instrução do programa?
- Modifique o programa para que este determine o menor dos dois números. Verifique.
- Faça um programa para ordenar uma sequência de valores.

4.6 Esquema


LPC-H2106
DLP-2232M MODULE
LPC2106
FT2232C
+5V
1.8V
3.3V
GND
XIN
X1
X2
RTCK
RST
DBGSEL
VDD1.8(CORE)
VDD3-2(I/O)
VDD3-1(I/O)
EXTVCC
VCCUSB
VCCSW
PORTVCC
RSTIN#
RSTOUT#
RESET#
XTIN
XTOUT
USBDP
USBDM
RSTOUT#
3V3OUT
AVCC
PWREN
VCCIOA
VCCIOB
TEST
EECS
EESK
EEDATA
SI/WUA
SI/WUB
14.7456MHZ
6 MHZ
Ferrite Bead