UNIVERSIDADE FEDERAL
DE SANTA CATARINA
Campus Araranguá

1ª Tarefa de Sistemas Digitais Embarcados

Nome do aluno: _____ **Matrícula:** ☐☐☐☐☐☐☐☐ **Data:** 14/03/2016

Disciplina: ARA7560 **Professor:** Fábio Rodrigues de la Rocha **Turma(s):** 08655

1 Introdução

O microcontrolador ARM cortex M3 utilizado nos kits de desenvolvimento é fabricado pela NXP sob o código LPC1768, é um microcontrolador de 64KiB de RAM e 512KiB de Flash. Nesta disciplina desenvolveremos código em principalmente em .C/C++ e utilizaremos o compilador gcc para esta tarefa. O gcc possui diversas ferramentas para realizar cada uma das etapas até gravar o código no microcontrolador. Ao final, utilizaremos uma outra ferramenta opensource para realizar a gravação do código no kit de desenvolvimento.

Essa tarefa usa o Gnu/Linux (distribuição Ubuntu 14.04).

1.1 Instalação do compilador

Digite na linha de comando:

```

sudo apt-get install gcc-arm-none-eabi
```

O apt-get instalará o compilador no diretório `/usr/bin/X11/`. O compilador na realidade é um conjunto de ferramentas como o montador assembly, pre-processador, compilador C, C++, linker etc.

1.2 O montador Assembly

Quando um programador assembly escreve um código, esse precisa passar pelo montador. O contador é o software que lê cada uma das instruções do arquivo fonte (geralmente chamado |*.s| como em `soma.s`) e converte para o conjunto de bits que representa aquela instrução do processador/microcontrolador. O resultado será um arquivo `soma.o`. Formalmente a parte do gcc que faz isso é o as (gnu assembler). Para o nosso uso, como estamos gerando código para um processador diferente do nativo o as será chamado de `arm-none-eabi-as`

Depois, para ler o arquivo `.o` e produzir um arquivo binário que possa ser executado, chamamos o linker, neste exemplo o linker deve produzir um arquivo `.elf` que é um arquivo binário de alto nível que deve ser lido pelo SO para ser carregado na memoria e executado. O nosso linker informa que o endereço do código inicia em 0, ou seja, a memoria flash começa no endereço 0. O linker que formalmente é o `ld` será chamado `arm-none-eabi-ld`.

Caso não sera utilizado um SO, deve-se transformar o `.elf` num arquivo binário (`.bin`) para ser gravado diretamente na memoria flash do processador. O arquivo binário pode ainda ser convertido num outro formato para ser lido pelos softwares de gravação de memória flash (formato `.hex`). Alguns softwares aceitam ambos o `.bin` e `.hex`. O software que cria o arquivo binário é o `objcopy` e que para nós será o `arm-none-eabi-objcopy`.

```
arm-none-eabi-as -o soma.o soma.s
arm-none-eabi-ld -Ttext=0x0 -o soma.elf soma.o
arm-none-eabi-objcopy -O binary soma.elf soma.bin
arm-none-eabi-objcopy -I binary -O ihex soma.bin soma.hex // le o .bin e gera um .hex
```

Os fontes estão disponíveis na página da disciplina, arquivos `exemplo1_ASM.zip` e `exemplo2_ASM.zip`

```

@ simbolo de comentario, equivale ao // do C++

     .text                @ tudo daqui para baixo sera considerado texto, entenda-se

                          @ "codigo", outro segmento possivel seria .data "dados", area de variaveis
start:                    @ um label, nao ocupa espaco
       mov   r0, #5       @ carrega o registrador r0 com o valor constante 5
       mov   r1, #4       @ carrega o valor r1 com o valor constante 4
       add   r2, r1, r0   @ adiciona r0 a r1 e coloca o resultado em r2

stop:  b stop             @ aqui existe um outro label e uma instrucao de b "branch",
                          @ que salta para o label stop, logo isso  um loop infinito
```

1.3 Gravando na memória Flash - usando o BOOTLOADER

O kit de desenvolvimento vem com o microcontrolador ARM pré-gravado com um bootloader. Quando o kit é ligado ele espera um tempo para testar se recebe dados pela porta serial. Caso nao receba dados dentro de um tempo máximo, ele desiste o passa a executar o código existente na memória Flash. Caso ele perceba que dados estão sendo recebidos, ele verifica se são dados válidos e caso sejam, grava byte por byte na sua própria memória flash, gerando um reset ao final.

Através de um cabo serial ligado a um PC pode-se transferir um arquivo binário para ser gravado na memória do ARM usando o tla bootloader. Pela porta serial podemos gerar um reset do ARM e assim a qualquer momento podemos forcar a execução do bootloader. Como a maioria dos PCs não vem mais com porta serial, utilizaremos um adaptador de USB/SERIAL.

1.3.1 Software para realizar a programação serial - WINDOWS

No windows, pode-se utilizar o flashmagic tal como apresentado no manual em anexo (está no site da disciplina).

1.3.2 Software para realizar a programação serial - LINUX

No linux, utilizaremos o software `lpc21isp`. Para instalá-lo e invocá-lo.

```
1 sudo apt-get install lpc21isp
2 lpc21isp -control  -bin soma.bin /dev/ttyUSB0 115200 12000
```

O nome `soma.bin` é o arquivo que desejamos gravar na flash, `/dev/ttyUSB0` é a porta que o nosso adaptador USB/SERIAL representa e que dependendo do seu sistema poderá ter um nome diferente. Ao inserir o adaptador USB/SERIAL use o comando `dmesg` para ver as mensagens do sistema. O Linux reconhecerá o dispositivo e apresentará ali o nome dele. Abaixo temos as últimas linhas do comando `dmesg` mostrando que no meu computador o adaptador foi reconhecido como `ttyUSB0`.

```
1  [ 6786.647965] usb 3-2: new full-speed USB device number 16 using xhci_hcd
2  [ 6786.666060] usb 3-2: New USB device found, idVendor=10c4, idProduct=ea60
3  [ 6786.666067] usb 3-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
4  [ 6786.666071] usb 3-2: Product: Conversor USB <-> RS232, Visite www.ATIVAsolucoes.com.br
5  [ 6786.666074] usb 3-2: Manufacturer: Silicon Labs
6  [ 6786.666077] usb 3-2: SerialNumber: CSUSB-01-007248
7  [ 6786.667242] cp210x 3-2:1.0: cp210x converter detected
8  [ 6786.831985] usb 3-2: reset full-speed USB device number 16 using xhci_hcd
9  [ 6786.848578] xhci_hcd 0000:00:14.0: xHCI xhci_drop_endpoint called with disabled ep ffff8800c21edbc0
10 [ 6786.848586] xhci_hcd 0000:00:14.0: xHCI xhci_drop_endpoint called with disabled ep ffff8800c21edb80
11 [ 6786.849130] usb 3-2: cp210x converter now attached to ttyUSB0
```

115200 é a taxa de transferência de dados para porta serial que será utilizada. Essa taxa não pode ser mudada foi o bootloader gravado no ARM foi escrito para trabalhar a esta taxa. Finalmente 12000 é a frequencia do cristal que existe no kit de desenvolvimento 12MHz mas o `lpc21isp` pede para entrar com uma frequencia em KHz.

IMPORTANTE. Para gravar o kit devemos posicionar as chaves CH3 (parte baixa à esquerda) para que as chaves 7 e 8 fiquem na posição **on**.

1.4 Gravando na memória Flash - usando o JTAG

JTAG [1] é um protocolo padronizado pela IEEE que tem por objetivo testar os elementos existentes dentro de uma placa de circuito. Nesta placa podemos ter vários CIs, sendo um ou mais destes processadores. O JTAG utiliza tipicamente 4 fios e transmite dados de uma forma serial. Esses dados são acessados como mensagens sendo recebidas pelos CIs de uma placa. Os CIs respondem as solicitações. Pode-se usar o JTAG para realizar um debug de um CI de uma placa, para alterar um conteúdo de sua memória, para programar o conteúdo de sua memória FLASH e muitas outras coisas. O JTAG é um padrão que é bastante utilizado, principalmente no mundo dos processadores mais complexos como o ARM ou em FPGAs. No mundo dos microcontroladores mais símples quando existem ferramentas de debug geralmente elas são proprietárias do fabricante do microcontrolador e fazem o debug apenas do microcontrolador e não são capazes de testar outros CIs que possam existir na placa.

No nosso caso iremos utilizar o JTAG para gravar o código na FLASH e também para fazer um debug do código. Esta seção utilizará o arquivo `experiencias.zip` existente da página da disciplina para algumas demonstrações.

O kit ARM que utilizaremos possui um JTAG USB embutido. Ao ligarmos o cabo USB no PC (no exemplo no Linux) e na traseira do Kit e ainda entrarmos com o comando `dmesg` teremos como saída o seguinte.

```
1
2  [ 1569.325811] usb 3-2: new full-speed USB device number 14 using xhci_hcd
3  [ 1569.345633] usb 3-2: New USB device found, idVendor=0403, idProduct=baf8
4  [ 1569.345668] usb 3-2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
5  [ 1569.345672] usb 3-2: Product: OOCDLink
6  [ 1569.345675] usb 3-2: Manufacturer: JK
7  [ 1569.920851] usbcore: registered new interface driver usbserial
8  [ 1569.920880] usbcore: registered new interface driver usbserial_generic
```

[1] `https://en.wikipedia.org/wiki/Joint_Test_Action_Group`

```
[ 1569.920901] usbserial: USB Serial support registered for generic
[ 1569.946632] usbcore: registered new interface driver ftdi_sio
[ 1569.946645] usbserial: USB Serial support registered for FTDI USB Serial Device
[ 1569.946743] usb 3-2: Ignoring serial port reserved for JTAG
[ 1569.946772] ftdi_sio 3-2:1.1: FTDI USB Serial Device converter detected
[ 1569.946800] usb 3-2: Detected FT2232C
[ 1569.946801] usb 3-2: Number of endpoints 2
[ 1569.946802] usb 3-2: Endpoint 1 MaxPacketSize 64
[ 1569.946803] usb 3-2: Endpoint 2 MaxPacketSize 64
[ 1569.946805] usb 3-2: Setting MaxPacketSize 64
[ 1569.946957] usb 3-2: FTDI USB Serial Device converter now attached to ttyUSB0
```

Neste caso o Linux reconheceu o dispositivo JTAG e o mesmo será acessando através do dispositivo `/dev/ttyUSB0` (caso eu estivesse com o adaptador usbserial conectado antes de ligar o JTAG eu teria como resultado num nome diferente, tal como ttyUSB1.

1.4.1 Software para realizar a programação pelo JTAG

O dispositivo JTAG utilizado no kit é diretamente reconhecido pelo Linux, tal como foi apresentado no saída do comando `dmesg`. Se fossemos utilizar o windows, seria necessário instalar drivers para o hardware JTAG existente no kit. Os drivers estão disponíveis no CD que vem junto com cada um dos kits de desenvolvimento.

Como no caso do linux não é necessário instalar os drivers, iremos passar direto para a etapa de instalar o software que comando o JTAG (OpenOCD [2])

```
sudo apt-get install openocd
sudo openocd -f lpc1768.cfg
// o comando acima faz com que o shell fique mostrando as mensagens do opencd
// abra outro shell (janela) e rode o programa abaixo
// voce precisa ter instalado o telnet (geralmente esta instalado)
telnet localhost 4444
```

O openocd precisa de permissões especiais para ser executado, por isso usamos o `sudo` para executá-lo como `root` a cada vez. Uma outra alternativa é após a instalação do `openocd` fazer com que este programa seja sempre executado como `root` usando, apenas uma vez:

```
whereis openocd // retorna onde esta instalado o openocd - tipicamente sera /usr/bin
cd /usr/bin    // vai para o diretorio onde esta o executavel
sudo chmod u+s openocd // altera o bit para executar como root
```

Agora vamos ver alguns comandos para usar no telnet:
Inicialmente vamos bloquear a execução do processador:

```
> reset halt // faz com que o processador sera resetado e a executacao do programa
             // bloqueada

JTAG tap: lpc1768.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x4)
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x1fff0080 msp: 0x10001ffc
```

Listar os bancos de memória

```
> flash banks // mostra os bancos de memoria

#0 : lpc1768.flash (lpc2000) at 0x00000000, size 0x00080000, buswidth 0, chipwidth 0
```

Extrair o conteúdo da FLASH:

```
> reset halt                        // precisamos parar a execucao do programa
   JTAG tap: lpc1768.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x4)
   target state: halted
   target halted due to debug-request, current mode: Thread
   xPSR: 0x01000000 pc: 0x1fff0080 msp: 0x10001ffc


> dump_image old_image.bin 0x0 0x80000 // le o conteudo da flash e grava num arquivo (demorado)
```

[2] http://openocd.org/getting-openocd/

```
                                    // este comando demora pois embora o micrcontrolador possa
                                    // ter apenas um programa de 2KiB em sua memoria
                                    // um dump ira ler os 512KiB inteiros e gerar um arquivo deste tamanho

dumped 524288 bytes in 36.860481s (13.890 KiB/s)
```

Gravar um novo conteúdo na FLASH:

```
> reset halt                        // precisamos parar a execucao do programa
   JTAG tap: lpc1768.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x4)
   target state: halted
   target halted due to debug-request, current mode: Thread
   xPSR: 0x01000000 pc: 0x1fff0080 msp: 0x10001ffc

> flash write_image erase BUZZER.bin 0x0 bin // grava um arquivo.bin na flash
                                             // para rodar o novo codigo devemos pressionar o botao RESET
                                             // o RESET precisa estar habilitado no CH3

   auto erase enabled
   Verification will fail since checksum in image (0x00000000) to be written to flash is different from calculated
   vector checksum (0xefff7afe).
   To remove this warning modify build tools on developer PC to inject correct LPC vector checksum.
   wrote 4096 bytes from file main.bin in 1.255727s (3.185 KiB/s)
```

1.4.2 Experiencia com o JTAG

Agora vamos fazer uma experiência de utilizar o JTAG.

- gravar um arquivo chamado `BUZZER.bin`, resetar o kit e perceber que a buzina esta funcionando. Lembre-se antes de gravar uma nova imagem, devemos fazer um `halt` do processador
- agora vamos extrair o conteúdo da flash e gravar num arquivo chamado `LIDO.bin`
- depois disso, gravar outro arquivo chamado `LEDS.bin`, resetar o kit e ver os leds piscando
- finalmente, gravar o arquivo que foi extraido anteriormente (`LIDO.bin`). Esta operação é demorada (quase 2 minutos) pois o arquivo que foi extraido tem 512KiB e será transferido novamente para a FLASH.