Sistemas Embebidos I
Programas romable – parte 1

Arquitectura ARM

A arquitectura ARM define que a gama de endereços de 0x00000000 a 0x0000001C é reservada para suporte a excepções, sendo designada por tabela de excepções. Quando ocorre uma excepção o PC é carregado com um destes endereços e o processador executa a respectiva instrução que é, necessariamente, uma instrução de salto.

Address	Exception
0x00000000	Reset
0x00000004	Undefined instruction
0x00000008	Software interrupt
0x0000000C	Prefectch Abort
0x00000010	Data abort
0x00000014	Reserved
0x00000018	IRQ
0x0000001C	FIQ

A operação de *reset* é enquadrada como uma excepção à qual corresponde o endereço 0x00000000. Neste endereço deve estar acessível a primeira instrução a ser executada e, ao alcance de uma instrução de salto (16 Mbyte) ou do load baseado no PC com deslocamento (4 KByte), mais algum código de iniciação.

Numa situação normal de funcionamento, esta zona de memória deve ser mapeada em memória RAM para permitir redefinir, dinamicamente, o código de tratamento das excepções.

Como conciliar estes dois requisitos: ter código em ROM no endereço 0x00000000 e a tabela de excepções em memória RAM?

A solução normalmente usada consiste em mapear nesta zona do espaço de endereçamento, na altura de reset, memória não volátil com o programa de arranque. Após o arranque, e sob o controlo do programa, o mapeamento será alterado para memória RAM. Esta operação designa-se por **remap**.

Arquitectura LPC2XXX

Esta família de microcontroladores incorpora, na memória FLASH interna, um programa monitor, designado por Boot Block. Este ocupa os 8 Kbyte do topo desta memória, que corresponde ao último sector. Este sector também é visível no topo do espaço de memória interno, imediatamente abaixo do endereço 0x80000000.

Na gama de endereços de 0x00000000 a 0x0000003F são visíveis, em alternativa, o início do Boot Block, o início da memória RAM ou o início da memória FLASH.

Processo de *boot*

Após *reset*, a base do Boot Block é mapeada nos endereços 0x00000000 a 0x0000003F. Como consequência o Boot Loader é sempre executado. Se o pino P0.14 estiver a zero o Boot Loader

mantém-se a executar. Se o pino P0.14 estiver a um e o código de utilizador, gravado na FLASH, for válido, a base da FLASH é remapeada para a gama 0x00000000 – 0x0000003F e o processador prossegue a execução a partir do endereço 0x00000000 no código do utilizador.

O valor lógico presente no pino P0.14 é lido imediatamente após o *reset* e conduz ao comportamento descrito atrás. Se se pretender usar este pino como saída é necessário, através de um *pull-up* ou de um *pull-down*, definir o seu valor durante o período de *reset*. Se se pretender usar como entrada é necessário que o elemento externo que lá seja ligado providencie o valor lógico adequado durante o período de *reset.*

Critério de validação do código de utilizador

Para que o programa de utilizador seja considerado válido, a soma de todas as posições da tabela de excepções deve ser 0. Para cumprir este requisito a posição reservada da tabela, de endereço 0x00000014, deve conter o complemento para dois da soma das outras posições.

```
      .text

      b     _start
      .word 0
      .word 0
      .word 0
      .word 0
      .word 0x15fffff2
      .word 0
      .word 0

      .skip 8 * 4,0

      .global     _start
_start:
      ldr   r0, =0xe0028000         /* GPIO_BASE */
      mov   r1, #0x10000
      str   r1, [r0, #8]            /* IODIR */
      str   r1, [r0, #4]            /* IOSET */
      str   r1, [r0, #12]           /* IOCLR */
      b     .
```

Geração do programa

A geração do programa processa-se tal como descrito no guia “Ambiente de trabalho” com a diferença do *script* de localização. Neste, é acrescentada a área de memória FLASH e a secção **.text** é aí localizada.

```
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
}
```

Para determinar o valor de validação do programa é necessário observar o código gerado pelas instruções inseridas na tabela de excepções.

```
$ arm-eabi-objdump -d main.elf

main.elf:     file format elf32-littlearm

Disassembly of section .text:

00000000 <.text>:
   0: ea00000e    b     40 <_start>
      ...
  14: 15fffff2    .word 0x15fffff2
      ...
  40: e59f0010    ldr   r0, [pc, #16]     ; 58 <.text+0x58>
  44: e3a01801    mov   r1, #65536  ; 0x10000
  48: e5801008    str   r1, [r0, #8]
  4c: e5801004    str   r1, [r0, #4]
  50: e580100c    str   r1, [r0, #12]
  54: eafffffe    b     54 <_start+0x14>
  58: e0028000    and   r8, r2, r0
```

A soma das 8 words que constituem a tabela de excepções é zero (este utilitário apresenta reticencias nas posições de memória a zero).

Preparação do programa para gravação

Na gravação em ROM pode manipular-se directamente o ficheiro executável, em formato **elf**, ou pode ser necessário converter para outro formato. O formato mais comum é o **binário**, em que se produz um ficheiro cujo conteúdo é a imagem exacta daquilo que irá ser o conteúdo da ROM.

O ficheiro binário pode ser gerado a partir do executável em formato elf, pelo utilitário objcopy:

```
$ arm-eabi-objcopy -Obinary main.elf main.bin
```