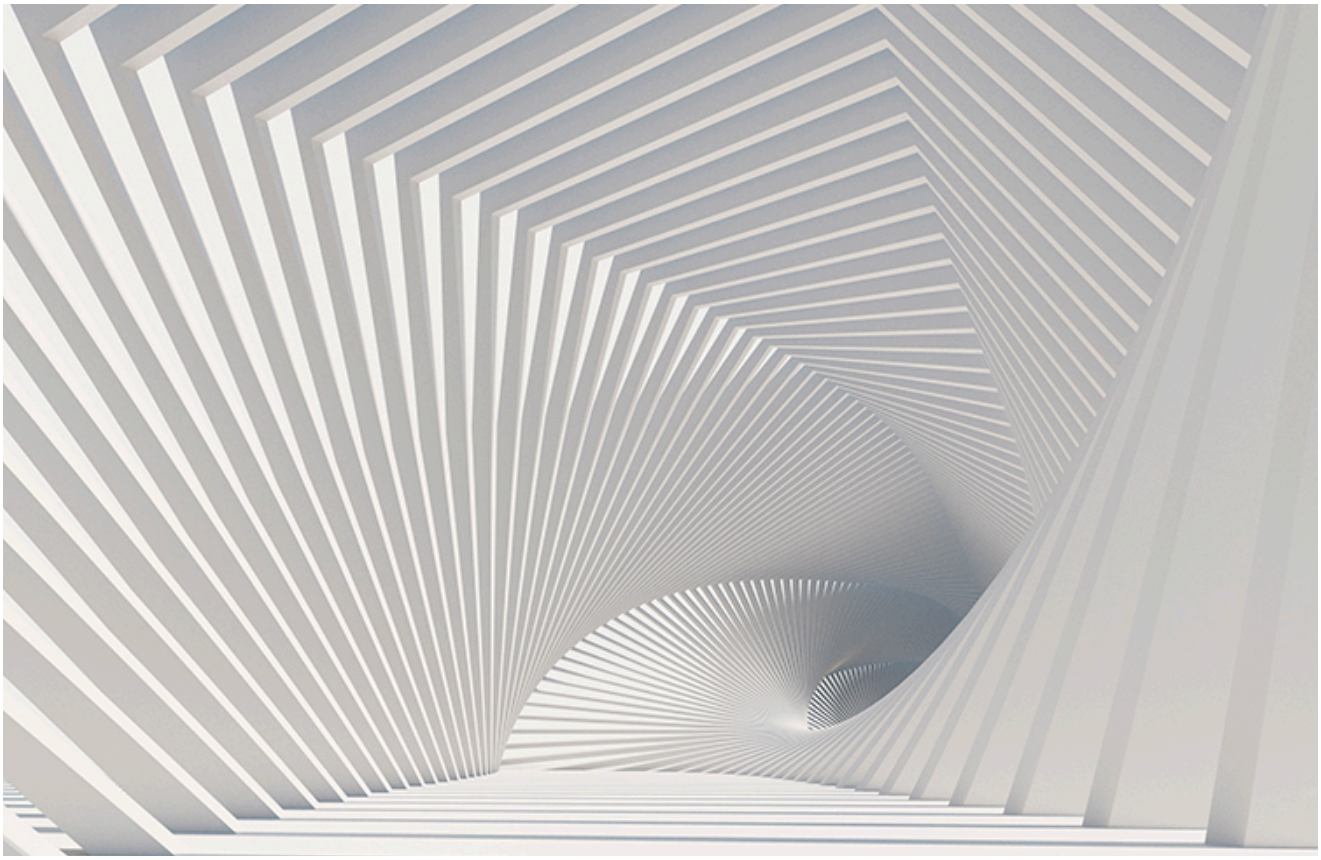
PORTANDO CÓDIGO DO CPULATOR PARA DE1-SOC



Introdução

O CPULator é um emulador de plataformas de hardware. Em particular, ele emula a placa DE1-SoC, utilizada na disciplina de Introdução aos Sistemas Embarcados.

O código desenvolvido no CPULator executa com poucas modificações na placa DE1-SoC.

As modificações são necessárias para acessar os periféricos, que são mapeados no espaço de endereçamento do processador que roda na placa. O mapeamento em memória é uma forma muito utilizada de acesso à periféricos, onde os registradores dos dispositivos são associados a endereços de memória. Assim, para ler/escrever um registrador de uma interface de dispositivo de E/S, o processador executa uma operação de leitura/escrita de memória no endereço que corresponde àquele registrador. Um controlador de memória identifica os endereços de E/S e

redireciona-os para os dispositivos, que podem então ser acessados através do barramento de memória.

CPULator:

No CPULator, o acesso a um periférico pode ser implementado em C atribuindo-se a um ponteiro o endereço de E/S do dispositivo.

Por exemplo, para ler os botões, podemos usar o seguinte código:

```
#define DEVICE_ADDR         0xFF200000
#define BUTTON_OFFSET       0x0050

volatile uint8_t *led_ptr = (volatile uint8_t *)(DEVICE_ADDR + BUTTON_OFFSET);
uint8_t key0 = *led_ptr & 0x01;     // lê o primeiro botão – key0
```

DEVICE_ADDR é o endereço inicial de uma região de memória associada a um conjunto de periféricos. O endereço de um periférico específico é obtido somando-se o deslocamento BUTTON_OFFSET ao endereço inicial.

O acesso aos botões é realizado atribuindo-se o endereço 0xFF200050 a ***led_ptr***, que é definido como um ponteiro volátil para bytes. No caso, o acesso a byte é utilizado pois existem apenas 4 botões, que são associados aos 4 primeiros bits do byte lido.

DE1-SOC Linux:

Quando executado em uma distribuição Linux que roda na placa, é necessário habilitar o acesso direto à memória a partir de um processo rodando no Linux.

Isso é necessário por que o sistema operacional cria um espaço de endereçamento virtual para cada processo, e realiza o mapeamento para a memória física com o auxílio de uma MMU (*Memory Management Unit*). Os endereços de E/S, entretanto, devem ser acessados diretamente, sem passar pelo mapeamento virtual - fisico do Linux.

O acesso direto no Linux é obtido através da abertura de um arquivo especial, o **/dev/mem**.

Ex:

```
int fd = open("/dev/mem", O_RDWR | O_SYNC);
```

fd é um índice para uma tabela com descritores de arquivos. Uma vez aberto o arquivo especial **"/dev/mem"**, o seu descritor é identificado por **fd**.

Após abrir esse arquivo, a função ***mmap()*** é utilizada para gerar um ponteiro para um bloco de endereços de E/S.

Ex:

```
#define HW_REGS_BASE 0xFF200000
#define HW_REGS_SPAN 0x00200000
#define LED_OFFSET   0x00
#define SWITCH_OFFSET 0x40
#define BUTTON_OFFSET 0x50
volatile void *peripherals;
```

```
peripherals = (void *)mmap(
      NULL,             // endereço mapeamento => NULL indica que o kernel escolhe
      HW_REGS_SPAN,     // tamanho em bytes do bloco de memória mapeado
      PROT_READ | PROT_WRITE, // acesso para leitura e escrita
      MAP_SHARED,       // endereços virtuais compartilhados com o hardware
      fd,               // índice do descritor de arquivos
      HW_REGS_BASE      // endereço base dos registradores dos dispositivos
);

led_ptr    = (uint32_t *)(peripherals + LED_OFFSET);
```

Uma vez realizado o mapeamento dos endereços do espaço virtual do processo no Linux para os endereços físicos, o acesso é similar:

```
*led_ptr = 0xFF;        // acende 8 leds
```

Código C para Portabilidade

Uma alternativa é utilizar o comando de compilação condicional **#ifdef**. Esse comando testa se uma macro do preprocessador foi definida e, em caso afirmativo, compila a seção de código correspondente a cláusula implícita **then**. Se a macro não foi definida, então a cláusula **#else**, se existir, será incluída na compilação.

A definição de uma macro, como **DE1_SOC** (note o sublinhado em vez de traço!) pode ser feita dentro do código ou na chamada do compilador.

Dentro do código:
#define DE1_SOC

Neste caso, vc tem que editar o arquivo fonte.

Uma forma mais prática é definir a macro na linha de comando do compilador:

```
gcc –DDE1_SOC …
```

Neste caso, não é necessário alterar o código fonte.

A seguir um exemplo de como organizar o código para a portabilidade entre CPULator e DE1-SoC.

A primeira parte da configuração é a inclusão dos *headers* necessários a compilação no Linux e do descritor de arquivo **fd**, que é utilizado pelas funções **init_io()** e **close_io()**.

Os endereços, *offsets* e ponteiros são os mesmos tanto para o CPULator quanto para a DE1-SoC. O que muda é a maneira como o ponteiro para o início da área de E/S é determinado.

No CPULator, basta atribuir um endereço ao ponteiro base, **peripherals**:

```
peripherals = (void *)HW_REGS_BASE;
```

Já no Linux da DE1-SoC é necessário abrir o arquivo /dev/mem/ e chamar a função de mapeamento de memória para definir esse ponteiro.

O restante do código é o mesmo para CPULator e DE1-SoC.

Se for necessário acessar VGA, um segundo mapeamento deve ser realizado, com um novo descritor de arquivo (**fd_vga** e **vga_ptr**, por exemplo).

```
#include <stdint.h>

#ifdef DE1_SOC
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int fd;
#endif

// Endereços dos periféricos
#define HW_REGS_BASE 0xFF200000
#define HW_REGS_SPAN 0x00200000

#define LED_OFFSET    0x00
#define SWITCH_OFFSET 0x40
#define BUTTON_OFFSET 0x50

// Ponteiros globais para dispositivos
volatile uint32_t *led_ptr;
volatile uint32_t *switch_ptr;
volatile uint32_t *button_ptr;
volatile void *peripherals;

void init_io() {
#ifdef DE1_SOC
// Definições específicas da DE1-SoC
    fd = open("/dev/mem", O_RDWR | O_SYNC);
    if (fd == -1) {
        printf("Erro ao abrir /dev/mem");
        exit(EXIT_FAILURE);
    }

    peripherals = (void *)mmap(NULL, HW_REGS_SPAN,
                               PROT_READ | PROT_WRITE,
                               MAP_SHARED, fd, HW_REGS_BASE);
    if (peripherals == MAP_FAILED) {
        printf("Erro no mmap");
        exit(EXIT_FAILURE);
    }
#else
// Definições para CPULator
    peripherals = (void *)HW_REGS_BASE;
#endif
    led_ptr    = (uint32_t *)(peripherals + LED_OFFSET);
    switch_ptr = (uint32_t *)(peripherals + SWITCH_OFFSET);
    button_ptr = (uint32_t *)(peripherals + BUTTON_OFFSET);
}

void close_io() {
#ifdef DE1_SOC
    munmap((void *)peripherals, HW_REGS_SPAN);
    close(fd);
#else
    // Nenhuma ação necessária no CPULator
#endif
}

int main() {
    init_io();
    *led_ptr = 0xFF;
    close_io();
    return 0;
}
```