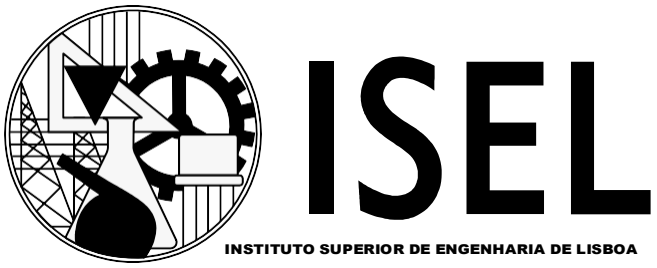
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Licenciatura em Engenharia Informática e de Computadores



Relatório do 4.º Trabalho Prático
de
Arquitetura de Computadores

Controlador de semáforos

Trabalho realizado por:

Daniel Cabaça Nº 48070

Nuno Venâncio Nº 45824

Docente: Jorge Fonseca

4 de junho de 2024

Descrição do Projeto

Introdução

Este trabalho tem como principal objetivo a exploração do hardware envolvente de um processador no desenvolvimento de programas escritos em linguagem assembly. Estão envolvidos os seguintes tópicos:

- entrada e saída de dados
- temporização
- interrupções externas
- organização de programas em rotinas
- implementação de máquinas de estados em software

Descrição do trabalho a realizar

Pretende-se o desenvolvimento do protótipo de um sistema embebido baseado no processador P16 que implemente o controlador de um sistema de semáforos para uma passadeira. Este sistema é composto por:

- um sinal luminoso circular para veículos, com uma única luz que pode acender com as cores vermelho e amarelo
- um sinal luminoso circular para peões, também com uma única luz, mas que pode acender com as cores vermelho e verde
- um botão de pressão para os peões solicitarem o atravessamento da faixa de rodagem. O sistema inclui ainda quatro interruptores para a configuração do tempo que o sinal verde de travessia de peões deve estar aberto

Arquitetura do protótipo

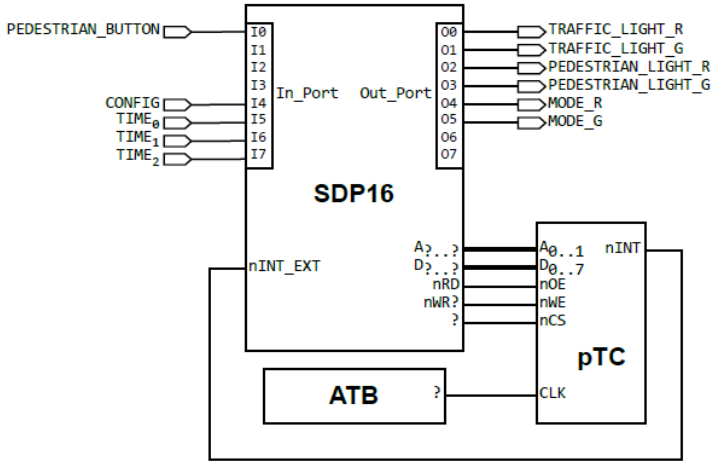


Figura 1: Diagrama de blocos do protótipo do sistema

Especificação do funcionamento do sistema

- Normalmente, o sistema encontra-se num estado em que o LED L1 apresenta uma luz amarela a piscar ao ritmo de 0,5 segundos (amarelo intermitente) e o LED L2 apresenta uma luz vermelha permanentemente acesa
- A deteção de uma transição ascendente ('0' → '1') no sinal PEDESTRIAN_BUTTON faz evoluir o sistema para um estado em que o LED L1 apresenta uma luz vermelha permanentemente acesa e o LED L2 apresenta uma luz verde permanentemente acesa
- O sistema mantém-se neste estado, no mínimo, durante um período CROSSING_TIME. Este tempo deve ser estendido por iguais períodos sempre que for detetada uma nova transição ascendente no sinal PEDESTRIAN_BUTTON
- O sistema retorna ao estado original logo que o tempo de espera se esgote
- Independentemente do estado em que o sistema se encontre, o LED L3 deverá apresentar uma luz verde permanentemente acesa
- A qualquer momento o sistema pode alternar para o modo de configuração, bastando para tal que o sinal CONFIG passe a tomar o valor '1'

No modo configuração, o sistema deve cumprir o seguinte funcionamento:

- O LED L3 deverá apresentar uma luz amarela permanentemente acesa, enquanto os LED L1 e L2 devem apresentar, respetivamente, luzes amarela e verde a piscarem ao ritmo de 0,5 segundos
- O utilizador pode definir o valor do período CROSSING_TIME alterando a combinação nos bits da entrada TIME. O sistema tem um valor pré-definido de 10 segundos
- A qualquer momento o sistema pode retornar ao modo de operação, bastando para tal que a entrada CONFIG volte a tomar o valor '0'

Solução da ligação SDP16 – pTC

De modo a estabelecer a comunicação entre o processador e o timer, será necessário fazer ligações entre os dois componentes.

Sinal	TPB	Linha
A_0	B5	4
A_1	B5	3
A_2	B5	2
A_3	B5	1
A_4	B4	4
A_5	B4	3
A_6	B4	2
A_7	B4	1
A_8	B3	4
A_9	B3	3
A_{10}	B3	2
A_{11}	B3	1
A_{12}	B2	4
A_{13}	B2	3
A_{14}	B2	2
A_{15}	B2	1

a) *Bus* de endereços

Sinal	TPB	Linha
D_0	B9	4
D_1	B9	3
D_2	B9	2
D_3	B9	1
D_4	B8	4
D_5	B8	3
D_6	B8	2
D_7	B8	1
D_8	B7	4
D_9	B7	3
D_{10}	B7	2
D_{11}	B7	1
D_{12}	B6	4
D_{13}	B6	3
D_{14}	B6	2
D_{15}	B6	1

b) *Bus* de dados

Sinal	TPB	Linha
nRD	B10	1
nWRL	B10	2
nWRH	B10	3
ALE	B10	4
RST_EXT	B11	1
CLK	B11	2
S0	B11	3
S1	B11	4
BGNT_EXT	B12	1
BREQ_EXT	B12	2
RDY_EXT	B12	3
nINT_EXT	B12	4
RFU0	B13	1
-	B13	2
nCS_EXT0	B13	3
nCS_EXT1	B13	4

c) Outros sinais

Figura 2: Ligações P16

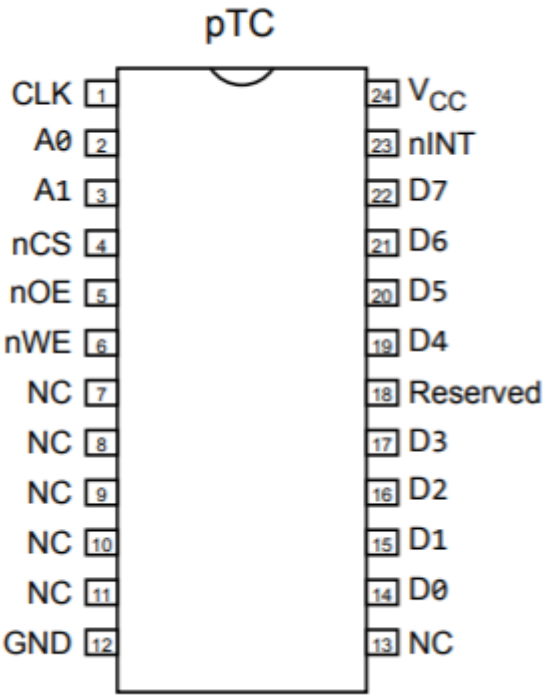


Figura 3: Ligações pTC

Para o objetivo pretendido, o seguinte esquema mapeia as ligações entre ambos componentes

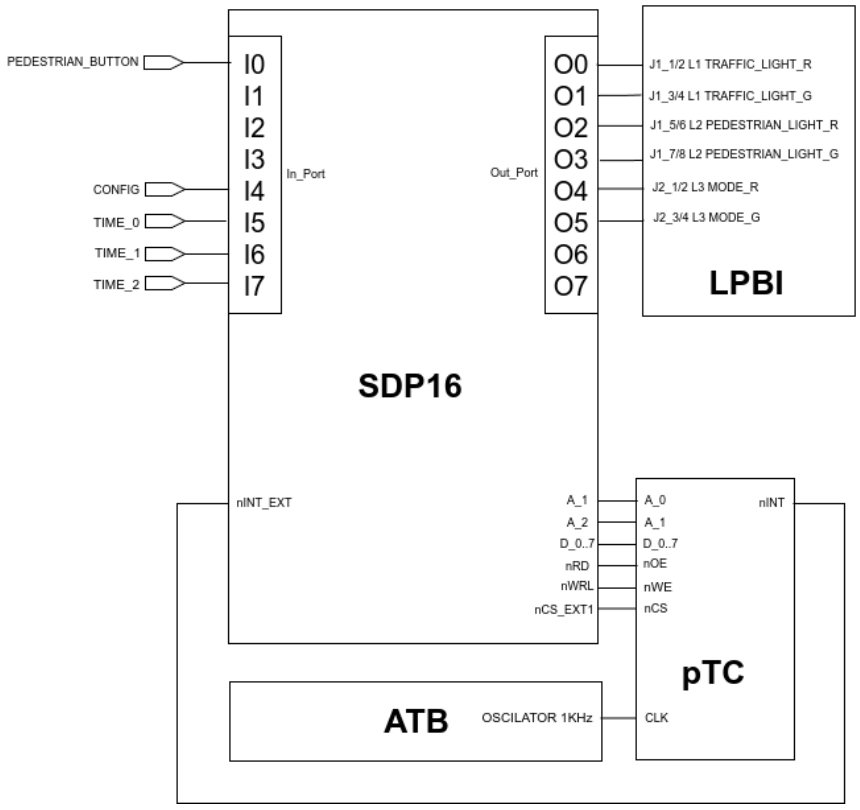


Figura 4: Ligação P16 - pTC

pTC	P16	ATB
CLK		1Khz
A0	B5 3	
A1	B5 2	
nCS	B13 3	
nOE	B10 4	
nWE	B10 2	
GND	GND	GND
VCC		VCC
nINT	B12 4	
D7	B8 1	
D6	B8 2	
D5	B8 3	
D4	B8 4	
D3	B9 1	
D2	B9 2	
D1	B9 3	
D0	B9 4	

Figura 5: Mapeamento Ligações pTC – P16

Flowchart

O seguinte Flowchart foi desenvolvido com o objetivo de facilitar a conceção do funcionamento global do protótipo. Adotámos uma abordagem de máquina de estado, que utiliza estados condicionados aos inputs e gerando outputs.

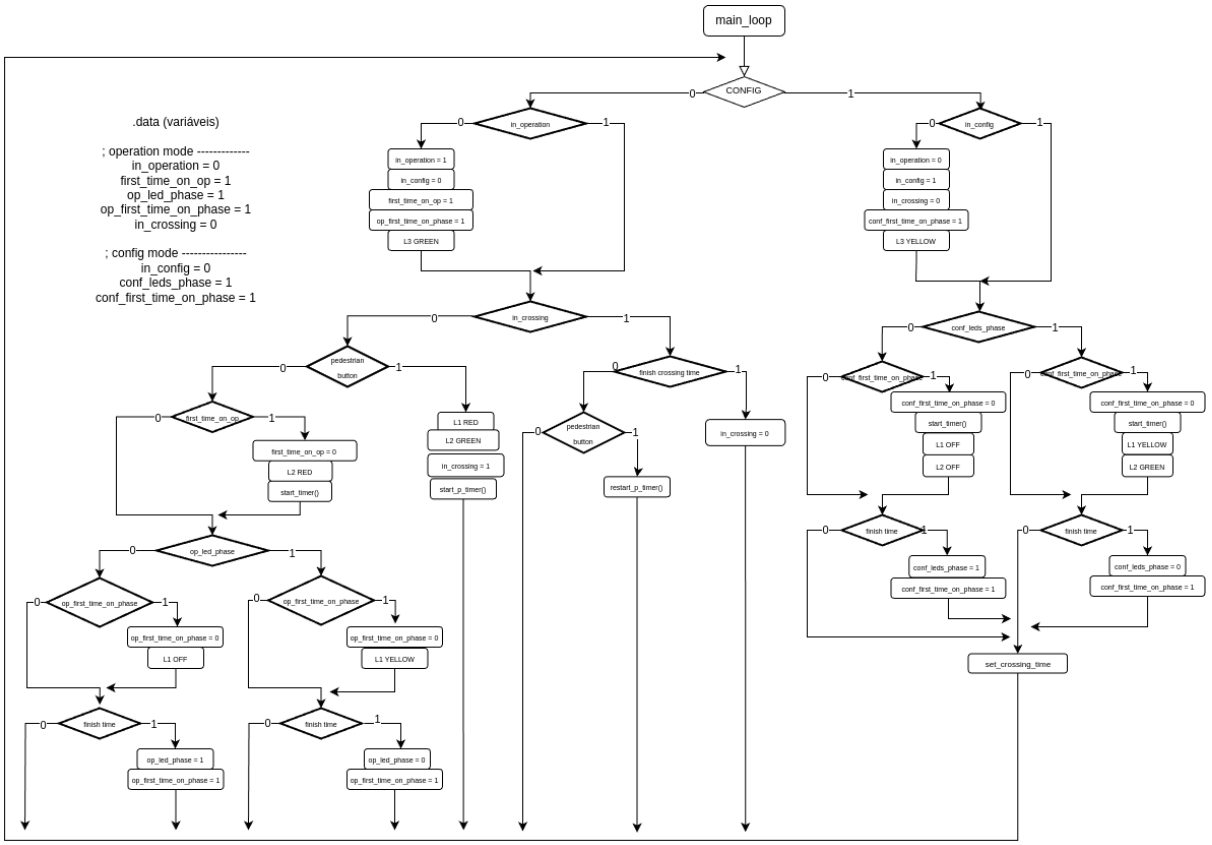


Figura 6: Flowchart da solução

Flags

in_operation: indica se estamos em modo operação
first_time_on_op: indica se estamos pela primeira vez no modo de operação
on_led_phase: indica se a fase será para acender ou apagar os leds
op_first_time_on_phase: indica se é a primeira vez na fase
in_crossing: indica se o tempo de travessia está a decorrer

in_config: indica se estamos no modo de configuração
conf_leds_phase: indica se a fase será par acender ou apagar os leds
conf_first_time_on_phase: indica se é a primeira vez na fase

Na escolha das flag, escolhemos usar flags que indicam se estamos pela primeira vez numa fase com o objetivo de poupar clocks, uma vez que guardamos esta flag em memória uma única vez e já não necessitamos de acender/desligar os leds sempre que entramos na mesma fase que a anterior.

Modo de operação

Começando no main loop, a primeira verificação que fazemos é ler o bit de modo (operação – configuração) para decidir para que ramo vamos.

Ramo de Configuração

A primeira vez que entramos neste ramo executamos as seguintes ações

- in_config = 1
- in_operation = 0
- in_crossing = 0
- conf_first_time_on_phase = 1
- colocar L3 a amarelo

Na segunda vez que passamos neste estado, vamos de imediato, sem nenhuma validação, para o próximo estado.

Os próximos estados destinam-se à operação dos leds L1 e L2, para sabermos se vamos entrar na fase de acender ou desligar, olhamos par a flag conf_first_time_on_phase, se visitarmos estas fases pela primeira vez executamos as seguintes ações:

- conf_first_time_on_phase = 0
- Inicializar o contador
- L1 OFF/YELLOW (depende da fase)
- L2 OFF/GREEN (depende da fase)

Após estas ações, verificamos se o temporizador, previamente inicializado, já chegou ao fim, se não o tiver alcançado, voltamos ao main loop e percorremos o Flowchart até chegar ao mesmo estado. Caso o tempo tenha, de facto, acabado, tomamos as seguintes ações:

- Conf_leds_phase = 0/1 (se for 0 passa a 1, se for 1 passa a 0)
- Conf_first_time_on_phase = 1

Sempre que saímos desta última fase, quer pelo fim do tempo, ou não, damos set no crossing_time de acordo com o valor presente nos últimos três bits do inport.

Ramo de Operação

A primeira vez que entramos neste ramo executamos as seguintes ações

- in_operation = 1
- in_config = 0
- first_time_on_op = 1
- op_first_time_on_phase = 1
- Ligar o L3 com a cor verde

Na segunda vez que passamos neste estado, vamos de imediato, sem nenhuma validação, para o próximo estado.

De seguida, verificamos se o tempo de travessia está a decorrer (in_crossing), se for o caso, vamos ver se o tempo já terminou, se tiver terminado, colocamos in_crossing a 0 e voltamos para o main loop. Se, por outro lado, ainda não tiver terminado, verificamos se houve alguém a carregar no botão, se sim, reiniciamos o timer, se não, não fazemos nada e saímos para o main loop.

Caso não haja ninguém a atravessar (in_crossing), verificamos de novo o botão de pedestres, se tivermos uma mudança ascendente de sinal tomamos as seguintes ações

- Ligar L1 com a cor vermelho
- Ligar L2 com a cor verde
- In_crossing = 1
- Inicializar o contador

Caso contrário, verificamos se é a primeira vez no modo operação, se for tomamos as seguintes ações

- First_time_on_op = 0
- Ligar L2 com cor vermelho
- Inicializar o contador

Se não for a primeira vez, seguimos para o próximo estado sem qualquer validação.

O próximo estado, mediante a flag op_led_phase é destinado a ligar/desligar o led L1. Se for a primeira vez neste estado executamos as seguintes operações

- L1 OFF/YELLOW (depende da fase)
- Op_first_time_on_phase = 0

Na segunda vez, seguimos para o próximo estado sem validações prévias.
Por fim, verificamos se o tempo já terminou, se não tiver acabado, voltamos par ao main loop sem fazer nada, caso contrário as seguintes ações são executadas antes de sair para o main loop

- Op_led_phase = 0/1 (se for 0 passa a 1, se for 1 passa a 0)
- Op_first_time_on_phase = 1

Código Assembly

Definições

Inport

```
; Definicoes do porto de entrada    ----------------------------------------
.equ    INPORT_ADDRESS, 0xFF80      ; Endereco do porto de entrada
.equ    PEDESTRIAN_BUTTON, 0x01     ; Mascara para botão de peão
.equ    CONFIG_SW, 0x10             ; Mascara para switch de CONFIG
.equ    TIME_SWS, 0xE0              ; Mascara para Time
```

Outport

```
  ; Definicoes do porto de saida      ----------------------------------------
  .equ    OUTPORT_ADDRESS, 0xFFC0     ; Endereco do porto de saida
  .equ    TRAFFIC_LIGHT, 0x03         ; Mascara para semáforo de trânsito
  .equ    PEDESTRIAN_LIGHT, 0x0C      ; Mascara para semáforo de peões
  .equ    MODE_LIGHT, 0x30            ; Mascara para semáforo de modo
```

pTC

```
; Definicoes do circuito pTC        ----------------------------------------
.equ    PTC_ADDRESS,  0xFF40        ; Endereco do circuito pTC
.equ    PTC_TCR, 0                  ; Deslocamento do registo TCR do pTC
.equ    PTC_TMR, 2                  ; Deslocamento do registo TMR do pTC
.equ    PTC_TC,  4                  ; Deslocamento do registo TC do pTC
.equ    PTC_TIR, 6                  ; Deslocamento do registo TIR do pTC
.equ    PTC_CMD_START, 0            ; Comando para iniciar a contagem no pTC
.equ    PTC_CMD_STOP, 1             ; Comando para parar a contagem no pTC
.equ    SYSCLK_FREQ, 100            ; Intervalo de contagem do circuito pTC
```

Cálculos relativos à temporização

Para calcular as temporizações e como precisamos de 500ms para a intermitência dos semáforos e múltiplos de 10 segundos até ao máximo de 60 para o tempo da travessia do peão, decidimos configurar cada tick a 100ms. Para tal temos de ligar o CLK com 1KHz e configurar o TMR (Timer Match Register)

do pTC com o valor 99/0x0063 (0-99 = 100 ciclos) ficando cada tick a uma frequência de 10Hz = 100ms.

Outras definições usadas

```
; Outras definicoes                  ----------------------------------------
.equ    BLINK_TIME, 0x0005          ; Tempo dos LEDs intermitentes
.equ    RED, 0x01                   ; Mascara para o LED vermelho
.equ    GREEN, 0x02                 ; Mascara para o LED verde
.equ    YELLOW, 0x03                ; Mascara para o LED amarelo
.equ    LIGHT_OFF, 0x00             ; Mascara para apagar os LEDs
```

Memória

```
; ###########################################################################
; Seccao:    data
; Descricao: Guarda as variáveis globais
;
    .data
outport_img:
    .space  1

sw_state:
    .space  1

sysclk:
    .space  2
```

Variáveis globais sem valor inicial para conter o último valor escrito no output port, o estado do switch do peão e o sysclock

```
; operation mode variables -------------------------------------------------
crossing_time:
    .word 0x0064

crossing_times_array:
    .word  100, 200, 300, 400, 500, 600, 600, 600 ; 1 tick = 100ms / 10 ticks = 1s
    ;.word  0x0064, 0x00C8, 0x012C, 0x0190, 0x01F4, 0x0258, 0x0258, 0x0258

in_crossing:     ; flag para saber se ha peao a atravessar
    .byte 0

in_operation:    ; flag para saber se o modo de operacao esta ativo
    .byte 0      ; 0 - 1a entrada em modo operação, 1 - reentradas vindas do main_loop

crossing_time_start_tick: ; para ter o valor do tick em que o peao comecou a atravessar
    .space 2

operation_time_start_tick: ; para ter o valor do tick em que o semaforo de transito apagou ou acende
    .space 2

first_time_on_op:  ; flag para saber se e para acender ou nao o led dos peoes
    .byte 1

op_led_phase:      ; flag para saber se e para acender ou nao o led do transito
    .byte 1

op_first_time_on_phase: ; flag para saber se e a primeira vez que os leds estao acesos
    .byte 1
```

Variáveis do modo operação guardadas em memória

```
; config mode variables ---------------------------------------------------
in_config:       ; flag para saber se o modo de configuracao esta ativo
    .byte 0      ; 0 - 1a entrada em modo operação, 1 - reentradas vindas do main_loop

config_time_start_tick: ; para ter o valor do tick em que os semaforos apagaram ou acenderam
    .space 2

conf_leds_phase:  ; flag para saber se e para acender ou nao os leds
    .byte 1

conf_first_time_on_phase: ; flag para saber se e a primeira vez que os leds estao acesos
    .byte 1
```

Variáveis do modo configuração guardadas em memória

Execução

Main

```
; Rotina:    main ----------------------------------------------------------
; Descricao: Ponto de entrada do programa
; Entradas:  -
; Saidas:    -
; Efeitos:   Inicializa o porto de saída, o temporizador, habilita as interrupções
;            externas e entra num loop infinito onde é verificado o estado do switch
;            do modo de operação e é chamada a rotina correspondente.
main:
    mov     r0, #0                      ; todos os LEDs apagados
    bl      outport_init
    mov     r0, #SYSCLK_FREQ            ; frequencia para o pTC
    bl      sysclk_init
    mrs     r0, cpsr                    ; habilitar interrupcoes
    mov     r1, #CPSR_BIT_I
    orr     r0, r0, r1
    msr     cpsr, r0

main_loop:
    bl      get_config_sw              ; verificar a posicao do switch de mode
    mov     r1, #CONFIG_SW
    cmp     r0, r1
    beq     main_config_mode
    bl      operation_mode
    b       main_loop
main_config_mode:
    bl      config_mode
    b       main_loop
    b       .
```

O ponto de entrada da nossa aplicação é a rotina *main* cuja funcionalidade é fazer os *inits* e configurações necessárias para o funcionamento correto do programa. Nomeadamente o *outport*, o *pTC* e habilitação do atendimento a interrupções externas.

Após a parte do *main* ter corrido, entra em *main_loop* (estado inicial do Flowchart). Esta, é um ciclo infinito que é constantemente visitado para verificar o *inport*, percebendo se estamos em modo operação ou configuração através do valor devolvido pela rotina get_*config*_sw.

Modo operação

```
; Rotina:    operation_mode ------------------------------------------------------
; Descricao: Acende os LEDs de acordo com o modo de operação e verifica se um
;            peão pressionou o botão de peão.
; Entradas:  -
; Saidas:    -
; Efeitos:   Faz a gestão do modo de operação do semáforo.
operation_mode:
    push   lr
    push   r4
    push   r5

    ; values 1 and 0 to use in flags to save clocks ---
    mov     r4, #0                          ; False
    mov     r5, #1                          ; True

    ; check if is the first time in operation mode --------
    ldr     r0, in_operation_addr0
    ldrb    r1, [r0, #0]
    cmp     r1, r5
    beq     not_first_time_in_operation_mode
    ; set in_config to 0, in_operation to 1 and in_crossing to 0
    strb    r5, [r0, #0]                    ; in_operation = 1
    ldr     r0, in_config_addr0
    strb    r4, [r0, #0]                    ; in_config = 0
    ldr     r0, in_crossing_addr0
    strb    r4, [r0, #0]                    ; in_crossing = 0
    ; set operation first time on phase to 1 -------------
    ldr     r0, op_first_time_on_phase_addr0
    strb    r5, [r0, #0]                    ; op_first_time_on_phase = 1
    ; turn on mode light L3 to green ---------------------
    mov     r0, #GREEN
    bl      mode_light_set_color
```

O estado inicial da rotina operação é o *operation_mode* que lê o valor em *in_operation_addr0* para perceber se é a primeira vez em modo operação, se for a primeira vez em modo operação, os valores in_*config* e in_*operation* e op_*first*_time_on_phase são configurados de acordo com o estado de atuação. O LED 3 é também ligado com o valor GREEN (presente em r0) e chamando a rotina *mode_light_set_color*.

```
not_first_time_in_operation_mode:
    ; check if a pedestrian is crossing ------------------
    ldr     r0, in_crossing_addr0
    ldrb    r0, [r0, #0]
    cmp     r0, r5
    beq     pedestrian_crossing
    ; check if a pedestrian pressed the button to cross --
    bl      check_pedestrian_button
    cmp     r0, r5
    beq     pedestrian_ask_to_cross
    ; check if is the first time in operation mode --------
    ldr     r0, op_first_time_on_phase_addr0
    ldrb    r1, [r0, #0]
    cmp     r1, r4
    beq     not_first_time_on_op
    ; first time in operation mode to 0 ------------------
    strb    r4, [r0, #0]                        ; first_time_on_op = 0
    ; turn on pedestrian light L2 to red ------------------
    mov     r0, #RED
    bl      pedestrian_light_set_color
    ; start timer to turn ON/OFF L1 LED -------------------
    bl      sysclk_get_ticks                    ; get actual tick number to start timer
    ldr     r1, operation_time_start_tick_addr0
    str     r0, [r1, #0]                        ; operation_time_start_tick = sysclk_get_ticks
not_first_time_on_op:
    ; check if L1 LED are in ON or OFF phase --------------
    ldr     r0, op_led_phase_addr0
    ldrb    r0, [r0, #0]
    cmp     r0, r5
    beq     led_on_phase
; leds are in OFF phase
    mov     r0, #LIGHT_OFF                      ; turn OFF traffic light
    bl      traffic_light_set_color
    bl      operation_mode_led_phases
    b       end_operation_mode
led_on_phase:
    mov     r0, #YELLOW                          ; turn ON traffic green
    bl      traffic_light_set_color
    bl      operation_mode_led_phases
    b       end_operation_mode
```

O próximo troço de código, (*not_first_time_in_operation_mode*), começa por verificar se existe alguém em travessia através da flag in_crossing, se não estiver ninguém, vai também verificar se houve uma transação de 1 para 0 (retorno da rotina *check_pedestrian_button*). Depois coloca a *flag first_time_on_op* a zero para indicar que a próxima vez que cá entrar, já não será a primeira vez, liga o LED L2 a vermelho atravéz da rotina *pedestrian_light_set_color* e vai ler e guardar o valor dos *ticks* atuais.

No troço *not_first_time_on_op* verificamos se é para ligar ou desligar o led através do valor armazenado na *flag op_led_phase* e chamando em seguida a rotina *operation_mode_led_phases.*

```
pedestrian_ask_to_cross:
    mov     r0, #RED
    bl      traffic_light_set_color
    mov     r0, #GREEN
    bl      pedestrian_light_set_color
    ldr     r0, in_crossing_addr0
    strb    r5, [r0, #0]                          ; R5 = 1, in_crossing = 1
    bl      start_pedestrian_timmer
    b       end_operation_mode
```

Se durante o modo operação um pedestre premir o botãodo semáforo é efetuado um salto para o troço *pedestrian_ask_to_cross, onde* é ligado L1 a vermelho e L2 a verde. Colocamos o valor da *flag in_crossing* a 1, sinalizando que está alguém na travessia da passadeira e inicializamos o respetivo timer.

```
pedestrian_crossing:
    ; check if crossing time finished ---------------------
    ldr     r0, crossing_time_start_tick_addr
    ldr     r0, [r0, #0]
    bl      sysclk_elapsed
    ldr     r1, crossing_time_addr0
    ldr     r1, [r1, #0]
    cmp     r0, r1
    bhs     pedestrian_time_finished
    ; check if anothers pedestrian pressed the button to cross
    bl      check_pedestrian_button
    cmp     r0, r5
    beq     pedestrian_ask_to_cross
    b       end_operation_mode
pedestrian_time_finished:
    ldr     r0, in_crossing_addr0
    strb    r4, [r0, #0]                        ; R4 = 0, in_crossing = 0

end_operation_mode:
    pop     r5
    pop     r4
    pop     pc
```

Durante do troço *pedestrian_crossing* verificamos o valor dos *ticks* em memória (o primeiro valor lido quando o peão iniciou o atravessamento) e os atuais afim de perceber se o tempo já terminou. Também importante salientar que aproveitamos para verificar novamente se durante o tempo em que o semáforo está verde para os peões alguém carregou novamente no botão, extendendo assim o tempo de atividade.

```
; Rotina:    start_pedestrian_timmer -------------------------------------------------
; Descricao: Inicia a contagem do tempo de travessia.
; Entradas:  -
; Saidas:    -
; Efeitos:   crossing_time_start_tick = sysclk
start_pedestrian_timmer:
    push    lr
    bl      sysclk_get_ticks
    ldr     r1, crossing_time_start_tick_addr
    str     r0, [r1, #0]
    pop     pc

operation_time_start_tick_addr:
    .word operation_time_start_tick

crossing_time_start_tick_addr:
    .word crossing_time_start_tick
```

Como já vimos, sempre que alguém carrega no botão da passadeira temos de iniciar o *timer*. Simplesmente vamos ler e guardar o número de ticks atuais.

```
; Rotina:    operation_mode_led_phases ------------------------------------
; Descricao : Verifica se é necessário mudar de fase nos LEDs do semáforo de
;             trânsito.
; Entradas  : -
; Saidas    : -
; Efeitos   : -
operation_mode_led_phases:
    push    lr
    push    r4
    push    r5

    mov     r4, #0                          ; False
    mov     r5, #1                          ; True

    ldr     r0, op_first_time_on_phase_addr
    ldrb    r1, [r0, #0]
    cmp     r1, r5
    bne     op_not_first_time_in_phase
    strb    r4, [r1, #0]                    ; set first_time_in_phase = False
op_not_first_time_in_phase:
    ldr     r0, operation_time_start_tick_addr
    ldr     r0, [r0, #0]
    bl      sysclk_elapsed
    mov     r1, #BLINK_TIME & 0xFF
    movt    r1, #BLINK_TIME >> 8 & 0xFF
    cmp     r0, r1
    blo     operation_mode_led_phases_end
    ldr     r0, op_led_phase_addr
    ldrb    r1, [r0, #0]
    mvn     r1, r1                          ; R1 = ~r1
    and     r1, r1, r5                      ; R1 = r1 & 1, mascara para ficar só com o último bi
    strb    r1, [r0, #0]                    ; op_led_phase = r1
    ldr     r0, op_first_time_on_phase_addr
    strb    r5, [r0, #0]                    ; first_time_in_phase = True
operation_mode_led_phases_end:
    pop     r5
    pop     r4
    pop     pc

op_first_time_on_phase_addr:
    .word   op_first_time_on_phase

op_led_phase_addr:
    .word   op_led_phase
```

A rotina *operation_mode_led_phases* verifica se estamos nesta fase pela primeira vez e coloca a 0 a *flag op_first_time_on_phase* caso seja verdade, evitando assim repetir a escrita no *outport* para acender o LED se este já estiver aceso ou apagar se este já estiver apagado.
Em seguida, a *label op_not_first_time_in_phase* é chamada quando sabemos que não é a primeira vez seguida que entramos neste estado. Vamos cálcular quantos *ticks* de diferença entre o tempo atual e o valor armazenado e comparamos, para perceber se o tempo já chegou ao fim ou não, comparando com o BLINK_TIME. No cenário no qual o tempo terminou, invertemos o valor da flag op_led_phase (1 luz amarela acesa, 0 luz apagada) e recolocamos novamente a *flag op_first_time_on_phase* a 1.

Modo configuração

```
; Rotina:    config_mode ----------------------------------------------------
; Descricao: Acende os LEDs de acordo com o modo de configuração.
; Entradas:  -
; Saidas:    -
; Efeitos:   Faz a gestão do modo de configuração do semáforo.
config_mode:
    push    lr
    push    r4
    push    r5

    ; values 1 and 0 to use in flags to save clocks ---
    mov     r4, #0                          ; False
    mov     r5, #1                          ; True

    ; check if is the first time in config mode --------
    ldr     r0, in_config_addr
    ldrb    r1, [r0, #0]
    cmp     r1, r5
    beq     not_first_time_in_config_mode
    ; set in_config to 1, in_operation to 0 and in_crossing to 0
    strb    r5, [r0, #0]                    ; in_config = 1
    ldr     r0, in_operation_addr
    strb    r4, [r0, #0]                    ; in_operation = 0
    ldr     r0, in_crossing_addr
    strb    r4, [r0, #0]                    ; in_crossing = 0
    ; set config first time on phase to 1 -------------
    ldr     r0, conf_first_time_on_phase_addr
    strb    r5, [r0, #0]                    ; conf_first_time_on_phase = 1
    ; turn on mode light L3 to yellow ------------------
    mov     r0, #YELLOW
    bl      mode_light_set_color
not_first_time_in_config_mode:
    ; check if L1 and L2 LEDS are in ON or OFF phase ---
    ldr     r0, conf_leds_phase_addr
    ldrb    r0, [r0, #0]
    cmp     r0, r5
    beq     config_mode_leds_on_phase
; leds are in OFF phase
    mov     r0, #LIGHT_OFF                  ; turn OFF traffic light
    mov     r1, #LIGHT_OFF                  ; turn OFF pedestrian light
    bl      config_mode_leds_phases
    b       config_mode_end
config_mode_leds_on_phase:
    ; check if is first time in ON phase ---------------
    mov     r0, #YELLOW                     ; turn ON traffic yellow
    mov     r1, #GREEN                      ; turn ON pedestrian green
    bl      config_mode_leds_phases
config_mode_end:
    bl      set_crossing_time
    pop     r5
    pop     r4
    pop     pc

in_config_addr:
    .word   in_config
in_operation_addr:
    .word   in_operation
in_crossing_addr:
    .word   in_crossing
```

Muita da lógica já abordada no modo de operação está também presente no modo de configuração. Primeiro, verificamos se estamos neste estado pela primeira vez, se for o caso, troca o valor para zero e configura as flags de acordo.
Por fim, liga o LED L3 a amarelo e lê e guarda os ticks atuais.

Se não for a primeira vez neste estado, lêmos o valor da *flag conf_leds_phase* para saber se temos de apagar ou acender os leds e operar sobre os mesmos. Se os leds tiverem na fase off, passamos o valor 0 em r0 e r1 para desligar os leds, se for para acender, em r0 passamos 3 (amarelo) e 2 em r1 (verde), chamando em seguida a rotina *config_mode_leds_phases*.
No final chamamos a rotina *set_crossing_time* para guardar o valor da configuração do tempo de atravessamento do peão.

```
; Rotina:    config_mode_leds_phases ----------------------------------------
; Descricao : Verifica se é necessário mudar de fase nos LEDs do semáforo de
;             configuração.
; Entradas  : R0 - cor a colocar no L1 (transito)
;             R1 - cor a colocar no L2 (peoes)
; Saidas    : -
config_mode_leds_phases:
    push    lr
    push    r4
    push    r5
    push    r6

    mov     r5, #1                              ; True
    ldr     r3, conf_first_time_on_phase_addr
    ldrb    r2, [r3, #0]
    cmp     r2, r5
    bne     not_first_time_in_phase
    ; save the light colors
    mov     r6, r0                              ; R0 = cor a colocar no L1 (transito), guardada em R
    mov     r4, r1                              ; R1 = cor a colocar no L2 (peoes), guardada em R4
    ; start timer to turn ON/OFF L1 and L2 LEDS ------------
    bl      sysclk_get_ticks                    ; get actual tick number to start timer
    ldr     r1, config_time_start_tick_addr
    str     r0, [r1, #0]                        ; config_time_start_tick = sysclk_get_ticks
    mov     r0, r6                              ; R0 = cor a colocar no L1 (transito)
    bl      traffic_light_set_color             ; R0 = cor a colocar no L1 (transito)
    mov     r0, r4                              ; R0 = cor a colocar no L2 (peoes)
    bl      pedestrian_light_set_color
    mov     r0, #0 && 0xFF
    movt    r0, #0 >> 8 & 0xFF
    ldr     r3, conf_first_time_on_phase_addr
    strb    r0, [r3, #0]                        ; set first_time_in_phase = False
not_first_time_in_phase:
    ldr     r0, config_time_start_tick_addr
    ldr     r0, [r0, #0]
    bl      sysclk_elapsed
    mov     r1, #BLINK_TIME & 0xFF
    movt    r1, #BLINK_TIME >> 8 & 0xFF
    cmp     r0, r1
    blo     config_mode_leds_phase_end
    ldr     r0, conf_leds_phase_addr
    ldrb    r1, [r0, #0]
    mvn     r1, r1                              ; R1 = ~r1
    and     r1, r1, r5                          ; R1 = r1 & 1, mascara para ficar só com o último bi
    strb    r1, [r0, #0]                        ; config_leds_on = r1
    ldr     r0, conf_first_time_on_phase_addr
    strb    r5, [r0, #0]                        ; first_time_in_phase = True
config_mode_leds_phase_end:
    pop     r6
    pop     r5
    pop     r4
    pop     pc

conf_first_time_on_phase_addr:
    .word   conf_first_time_on_phase
config_time_start_tick_addr:
    .word   config_time_start_tick
conf_leds_phase_addr:
    .word   conf_leds_phase
```

Nesta rotina, vamos ler o valor da *flag conf_first_time_on_phase* para ver se estamos neste estado pela primeira vez e iniciamos o *timer*, chamamos as funções respetivas para cada semáforo (pedestre ou de trânsito) com os devidos valores previamente fornecidos em r0 e r1. No fim, dizemos que não estamos neste estado pela primeira vez.
Novamente, se não for a primeira vez no estado, apenas verificamos se o tempo já terminou, se isso se verificar, invertemos o valor da *flag conf_leds_phase* para alternar de modo a que na próxima vez, se as luzes tiverem ligadas serem desligadas e vice-versa.

```
; Rotina:    isr ----------------------------------------------------------------
; Descricao: Incrementa o valor da variável global sysclk.
; Entradas:  -
; Saidas:    -
; Efeitos:   Incrementa o valor da variável global sysclk
isr:
    push    r0
    push    r1

    ; incrementar sysclk
    ldr     r0, sysclk_addr0
    ldr     r1, [r0, #0]
    add     r1, r1, #1
    str     r1, [r0, #0]

    ; "limpar" a interrupção no pTC
    ldr     r0, ptc_addr
    strb    r1, [r0, #PTC_TIR]

    pop     r1
    pop     r0
    movs    pc, lr

ptc_addr:
    .word PTC_ADDRESS

sysclk_addr0:
    .word sysclk
```

Rotina de atendimento às interrupções que incrementa o valor da variável sysclk usada para contar o tempo de 100 em 100 ms.

Serviços

```
; Rotina:    set_crossing_time ------------------------------------------------
; Descricao: Atribui o tempo de travessia especificado à variável crossing_time.
; Entradas:  -
; Saidas:    -
; Efeitos:   crossing_time = crossing_times_array[TIME_SWS]
set_crossing_time:
    push    lr
    bl      inport_read
    mov     r1, #TIME_SWS & 0xFF
    movt    r1, #TIME_SWS >> 8 & 0xFF
    and     r2, r0, r1                  ; selecionar apenas bits dos Time switches
    lsr     r2, r2, #5                  ; shift right para obter o index
    add     r2, r2, r2                  ; r2 = r2 * 3
    ldr     r0, crossing_times_array_addr
    ldr     r1, [r0, r2]                ; transferir o valor da array de times, index = r2
    ldr     r0, crossing_time_addr
    str     r1, [r0, #0]                ; crossing_time = crossing_times_array[index]
    pop     pc

crossing_times_array_addr:
    .word crossing_times_array

crossing_time_addr:
    .word crossing_time
```

Rotina usada para estabelecer a temporização do *crossing time*. Lemos o valor do *inport* com a máscara dos 3 últimos bits (TIME_SWS), *shiftamos* para a direita por 5 e com este valor como *index* do *array crossing_times_array*, vamos buscar a temporização pretendida e guardar o seu valor em crossing_time, o qual é lido para calcular o tempo de atravessamento na rotina *operation_mode*.

```
; Rotina:    get_config_sw ----------------------------------------------------
; Descricao: Devolve se um peão pressionou o botão de peão.
; Entradas:  -
; Saidas:    R0 - 1 se o botão de peão foi pressionado, 0 caso contrário
; Efeitos:   -
get_config_sw:
    push    lr
    bl      inport_read
    mov     r1, #CONFIG_SW
    and     r0, r0, r1
    pop     pc
```

A rotina *get_config_sw* lê o valor do inport com a máscara do 5º bit afim de perceber o valor presente no *switch* de configuração.

```
; Rotina:    sw_is_released -----------------------------------------------------
; Descricao: Deteta se existiu uma transicao descendente no bit identificado em
;            IN_PEDESTRIAN_MASK.
; Entradas:  R0 - valor do porto de entrada
;            R1 - bit a analisar
; Saidas:    R0 - 1 se houve transicao descendente, 0 caso contrario
; Efeitos:   -
sw_is_released:
    push    lr
    and     r0, r0, r1                  ; r0 = sw_new_state = inport_value & pin_mask
    ldr     r1, sw_state_addr
    ldrb    r2, [r1, #0]                ; r2 = sw_state
    cmp     r0, r2                      ; r0 = sw_new_state, r2 = sw_state
    beq     sw_is_released_0
    strb    r0, [r1, #0]                ; sw_state = sw_new_state
    mov     r1, #1
    cmp     r0, r1                      ; r0 = sw_new_state
    bne     sw_is_released_1
sw_is_released_0:
    mov     r0, #0
    b       sw_is_released_end
sw_is_released_1:
    mov     r0, #1
sw_is_released_end:
    pop     pc

sw_state_addr:
    .word sw_state
```

Rotina cuja função é detetar se existiu ou não uma transição descendente no bit que representa o botão do peão, retornando 1 em caso afirmativo e 0 em caso negativo.

```
; Rotina:    traffic_light_set_color -------------------------------------------
; Descricao: Atribui a cor especificada em R0 ao semáforo de trânsito (L1).
; Entradas:  R0 - Cor a atribuir ao semáforo de trânsito (L1)
; Saidas:    -
; Efeitos:   -
traffic_light_set_color:
    push    lr
    mov     r1, r0                      ; r1 = color
    mov     r0, #TRAFFIC_LIGHT          ; r0 = TRAFFIC_LIGHT
    bl      outport_write_bits
    pop     pc
```

Rotina que tem como objetivo escrever no outport a cor passada em r0 referente ao LED do trânsito

```
; Rotina:    pedestrian_light_set_color ----------------------------------------
; Descricao: Atribui a cor especificada em R0 ao semáforo de peões (L2).
; Entradas:  R0 - Cor a atribuir ao semáforo de peões (L2)
; Saidas:    -
; Efeitos:   -
pedestrian_light_set_color:
    push    lr
    mov     r1, r0                      ; r1 = color
    mov     r0, #PEDESTRIAN_LIGHT       ; r0 = PEDESTRIAN_LIGHT
    lsl     r1, r1, #2                  ; r1 = color << 2
    bl      outport_write_bits
    pop     pc
```

Rotina que tem como objetivo escrever no outport a cor passada em r0 e depois *shiftada* para a esquerda de 2 valores, referente ao LED dos peões.

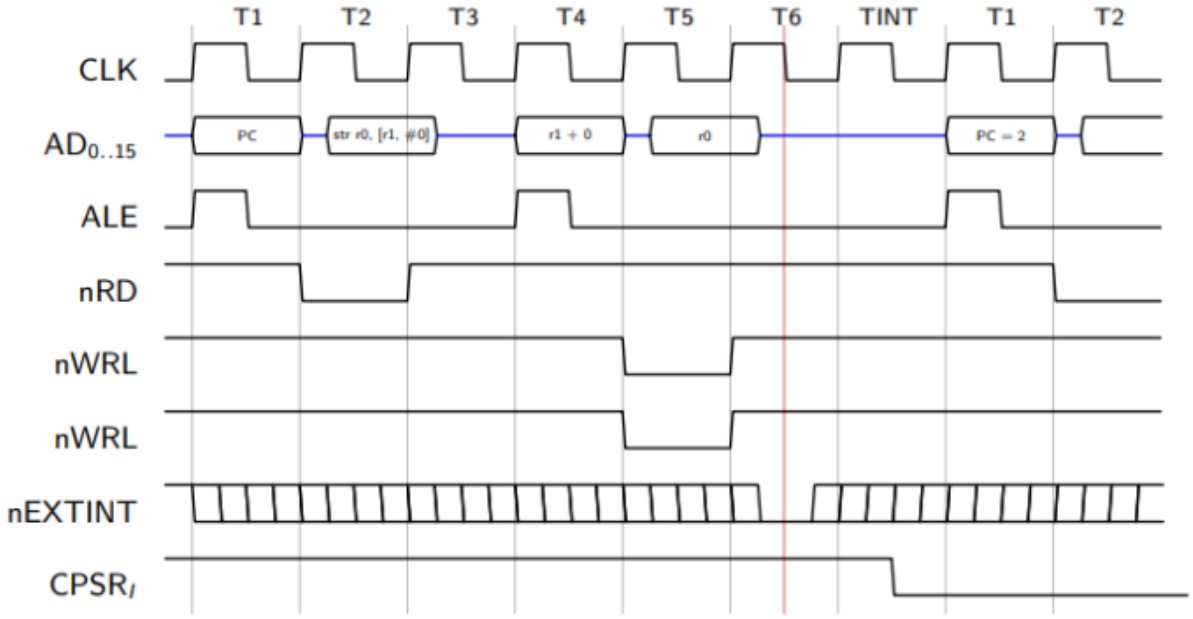
```
; Rotina:    mode_light_set_color ---------------------------------------------
; Descricao: Atribui a cor especificada em R0 ao semáforo de modo (L3).
; Entradas:  R0 - Cor a atribuir ao semáforo de modo (L3)
; Saidas:    -
; Efeitos:   -
mode_light_set_color:
    push    lr
    mov     r1, r0                      ; r1 = color
    mov     r0, #MODE_LIGHT             ; r0 = MODE_LIGHT
    lsl     r1, r1, #4                  ; r1 = color << 4
    bl      outport_write_bits
    pop     pc
```

Rotina que tem como objetivo escrever no outport a cor passada em r0 e depois *shiftada* para a esquerda de 4 valores, referente ao LED que sinaliza o modo de operação (config ou operation).

Foram usadas as rotinas de inicialização e configuração dos periféricos dadas nas aulas, sem termos efetuado alterações às mesmas. Nomeadamente:

- *sysclk_init*
- *sysclk_get_ticks*
- *sysclk_elapsed*
- *inport_read*
- *outport_set_bits*
- *outport_init*
- *outport_write_bits*
- *outport_write*
- *ptc_init*
- *ptc_start*
- *ptc_stop*
- *ptc_get_value*
- *ptc_clr_irq*

Latência máxima do sistema no atendimento dos pedidos de interrupção geradas pelo pTC



A latência máxima no atendimento de interrupções ocorre quando o pedido é iniciado no inicio de uma intrução de acesso à memória, ou seja, 6 clocks da intrução, mais 1 clock de tratamento das 'flags' e colocação do PC em 0x0002, + 6 clocks de redirecionamento para a rotina 'isr' uma vez que a mesma não se encontra no endereço 0x0002.
Uma vez que o clock do P16 funciona a uma frequencia de 50KHz (20 micro segundos), temos 79 x 20 = 260 microsegundos = 0,26 milisegundos.

Tempo gasto, no pior dos casos, na execução da rotina ISR

No pior dos casos, ou seja, instruções de acesso à memória (6 clocks), o tempo gasto na execução da rotina de atendimento a interrupções são 1,46 milisegundos.

Intrução	**Número de clocks usados**
Instrução de acesso à memória	6
Tratamento das flags e PC = 0x0002	1
ldr PC, isr_addr	6
push R0	6
push R1	6
ldr R0, sysclk_addr	6
ldr R1, [R0, #0]	6
add R1, R1, #1	3
str R1, [R0, #0]	6
ldr R0, ptc_addr	6
strb R1, [Ro, #PTC_TIR]	6
pop R1	6
pop R0	6
movs PC, LR	3
TOTAL	73

Uma vez que o clock do P16 funciona a uma frequencia de 50KHz (20 micro segundos), temos 73 x 20 = 1460 microsegundos = 1,46 milisegundos.

Conclusões

Na resolução deste trabalho prático, sentimos diversas dificuldades. Primariamente, na esquematisação do flowchart, o qual reconhecemos que elaboramos de uma forma complexa, cheia de flags para mudança e de estados. Teria ficado de mais fácil compreensão e por sua vez implementação em assembly se tivessemos usado "cases".
Outra dificuldade que sentimos foi ao tentar fazer "debug", usando o p16sim, não é possível configurar o clock a 1KHz, ou seja, tinhamos de alterar o SYSCLK_FREQ. Ao usar a placa SPD16, é impossível, fazer debug passo a passo com o atendimento a interrupções ligado, tendo para isso de desligar a ligação nINT_EXT da placa e liga-la numa das saídas da ATB. No nosso caso, tinhamos um bug na rotina ISR, estavamos a usar ldrb e strb em vez de ldr e str, o qual por debug não conseguimos detetar, apenas a rever o código.
Consideramos que estas dificuldades fazem parte do processo de aprendizagem, as quais nos fizeram ser mais atentos e minuciosos no processo de experimentação. Apesar de tudo, fomos percistentes e conseguimos atingir os objetivos.
Este trabalho, contribuio para termos um conhecimento mais sólido do conteúdo da matéria da cadeira, sobretudo no que se refere aos tópicos usados neste trabalho, entrada e saída de dados, temporização, interrupções externas, organização do programa em rotinas e implementação de máquinas de estados em software.