**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

Licenciatura em Engenharia Informática e de Computadores

Logo

Description automatically generated

Arquitetura de Computadores

**Relatório Trabalho 4**

Trabalho realizado por:

|  |  |
| --- | --- |
| Nome: André Monteiro | Nº 43842 |
| Nome: Constança Costa | Nº 50541 |
| Turma: LEIC24D  Docente: Rui Duarte |  |

2022 / 2023 Semestre Verão

13 de junho de 2023

**Índice**

1. **Introdução**
2. **Desenvolvimento do trabalho**
   1. Arquitetura do protótipo
   2. Questões a serem respondidas
   3. Código
3. **Conclusão**

# **Introdução**

Neste relatório, iremos utilizar o hardware adjacente ao processador P16, mais especificamente a estruturação de rotinas, implementação de máquinas de estado, manipulação de entrada e saída de dados, controlo de temporização e interrupções externas.

No processo de desenvolvimento do trabalho tiramos partido das funções do PTC, ou seja, possibilitar a implementação de mecanismos de temporização e controlo precisos, de modo a possibilitar atrasos controlados ou medição de intervalos.

# **Desenvolvimento do trabalho**

***Uma imagem com texto, recibo, diagrama, Paralelo

Descrição gerada automaticamente2.1.*** **Arquitetura do Protótipo**

Figura 1- StateFlow

***2.2* Questões a serem respondidas**

1)

* ATB 1KHz - pTC CLK
* pTC A1 - SDP16 A15
* pTC A0 – SDP16 A14
* pTC D7..0 – SDP16 D7..0
* pTC nWE – SDP16 nWRL
* pTC nCS – SDP16 nCS\_EXT1

2) Neste trabalho usámos duas frequências de tempo, 250 e 2, que no código estão referidas como ‘SYSCLK\_FREQ’ e ‘REACT\_TIME\_FREQ’, respetivamente, onde os valores são 249 e 1, pois o pTC faz a sua contagem incluindo o valor 0, pelo que foi necessário subtrair 1 ao valor desejado para obter o valor a utilizar.

O valor 250 foi obtido através da divisão de 1s = 1000ms por 4 de maneira que se obtivesse um valor ao qual o pTC conseguisse chegar.

O valor 2 foi obtido através da contagem de tempo que a rotina de verificação do valor do User e da rotina de interrupção demoram com o clock do P16, que se dá da seguinte forma:

Inport\_read :

Ldr (6 ciclos)

Ldrb (6 ciclos)

Mov (3 ciclos)

Isr:

Push (6 ciclos) \* 2

Ldr (6 ciclos) \* 3

Strb (6 ciclos)

Add (3 ciclos)

Str (6 ciclos)

Pop (6 ciclos) \* 2

Movs (3 ciclos)

Wait\_user:

Inport\_read

Mov (3 ciclos) \* 2

And (3 ciclos)

Cmp (3 ciclos)

Bne (3 ciclos)

Somando todos os valores anteriores, obtemos:

6+6+3+6\*2+6\*3+6+3+6+6\*2+3+3\*2+3+3+3 = 90

Multiplicando agora pelo tempo de um ciclo do clock do P16, temos que

90 \* 1/50000 = 0.0018 s = 1.8 ms

Pelo que para haver tempo suficiente para se percorrer todo o código necessário, foi usado o menor inteiro maior que o valor obtido, que dá 2.

3) Como visto na questão anterior, a rotina de atendimento a pedidos de interrupção (isr) demora 6\*2+6\*3+6+3+6+6\*2+3 = 60 ciclos de clock, que resulta em 60\*1/50000 = 0.0012, pelo que a latência desta rotina é de 1.2ms.

4) Pelo resultado da questão anterior, o tempo máximo que a demora a rotina a ser executada é 1.2ms.

***2.3 Código***

.equ STACK\_SIZE, 64

;endereços dos port em uso

.equ INPORT\_ADDRESS, 0xFF80

.equ OUTPORT\_ADDRESS, 0xFFC0

.equ PTC\_ADDRESS, 0xFF78

;offsets dos registos do ptc

.equ PTC\_TCR, 0

.equ PTC\_TMR, 2

.equ PTC\_TC, 4

.equ PTC\_TIR, 6

;mascaras

.equ USER, 0x01

.equ STIMULUS, 0x01

.equ TIME, 0xF0

.equ RESULT, 0xFE

; Definicao dos valores dos simbolos utilizados no programa

.equ AVG\_TIME, 200

.equ PTC\_CMD\_START, 0

.equ PTC\_CMD\_STOP, 1

.equ LIMIT, 63

.equ OUT\_OF\_RANGE, -64

.equ MIN\_TIME, 1

.equ MAX\_TIME, 10

.equ MULT\_BY\_4, 2

.equ ON, 1

.equ OFF, 0

.equ CHANGE\_PART, 4; de high para low ou de low para high

.equ TO\_RESULT, 1

.equ SHOW\_RES\_TIME, 5

.equ SYSCLK\_FREQ, 249

.equ REACT\_TIME\_FREQ, 1

; Seccao:    startup

; Descricao: Guarda o código de arranque do sistema

.section .startup

    b \_start

    ldr pc, isr\_addr

\_start:

    ldr sp, tos\_addr

    ldr pc, main\_addr

tos\_addr:

    .word tos

main\_addr:

    .word main

isr\_addr:

    .word isr

; Seccao:    text

; Descricao: Guarda o código do programa

.text

main:

    b init

;estado inicial onde liga todos os leds

init:

    mov r0, #STIMULUS

    mov r1, #RESULT

    orr r0, r0, r1

    bl outport\_write

    b wait\_user\_1to0

;espera que o SW USER seja colocado a 0

wait\_user\_1to0:

    bl inport\_read

    mov r1, #USER

    and r0, r0, r1

    mov r1, #OFF

    cmp r0, r1

    bne wait\_user\_1to0

;espera que o SW USER seja colocado a 1

wait\_user\_0to1:

    bl inport\_read

    mov r1, #USER

    and r0, r0, r1

    mov r1, #ON

    cmp r0, r1

    bne wait\_user\_0to1

    b setup\_test

;espera do tempo colocado pelo utilizador em segundos

;Entradas: R0 - numero de segundos a esperar

delay:

    push lr

    bl mult\_by\_4

delay\_loop:

    bl check\_user

    bl sysclk\_get\_ticks

    cmp r0, r1

    bne delay\_loop

delay\_end:

    pop pc

;espera durante 'SHOW\_RES\_TIME' segundos

delay\_result:

    push lr

    mov r0, #SHOW\_RES\_TIME

    bl mult\_by\_4

delay\_result\_loop:

    bl sysclk\_get\_ticks

    cmp r0, r1

    bne delay\_result\_loop

delay\_result\_end:

    pop pc

;multiplica o valor recebido em r0 por 4 e devolve o resultado em r1

mult\_by\_4:

    lsl r1, r0, #MULT\_BY\_4

    mov pc, lr

;verifica se o SW USER foi colocado a 0 e dá abort em caso positivo

check\_user:

    push lr

    push r0

    push r1

    bl inport\_read

    mov r1, #USER

    and r0, r0, r1

    mov r1, #OFF

    cmp r0, r1

    beq init

    pop r1

    pop r0

    pop pc

;faz as preparações do teste, isto é, desliga todos os leds excepto o do STIMULUS

setup\_test:

    bl reset\_sysclk

    bl get\_time

    mov r3, r0

    mov r0, #STIMULUS

    bl outport\_write

    b test

;estado de controlo do teste

test:

    bl wait\_time

    mov r0, #OFF

    bl outport\_write

    bl measure\_time

    bl get\_result

    b init

;dá enable às interrupções e chama a função 'delay' para o tempo de espera

;indicado parando depois o contador do ptc e dando reset ao sysclk para

;preparar a proxima contagem

wait\_time:

    push lr

    mov r0, #SYSCLK\_FREQ

    bl ptc\_init

    mrs r0, cpsr ; lê valor actual do cpsr

    mov r1, #0x10 ; mask do bit a activar IE

    orr r0, r0, r1 ; junta aos outros bits de estado

    msr cpsr, r0 ; carrega novo cpsr

    mov r0, r3

    bl delay

    bl ptc\_stop

    bl reset\_sysclk

    pop pc

;lê o tempo do inport, dando-lhe shift da parte alta para a parte baixa

;colocando-o dentro do range no caso em que esteja fora deste

get\_time:

    push lr

    bl inport\_read

    mov r1, #TIME

    and r0, r0, r1

    lsr r0, r0, #CHANGE\_PART

    and r0, r0, r0

    beq less\_than\_1

    mov r1, #MAX\_TIME

    cmp r0, r1

    bge over\_10

    pop pc

;caso seja menor coloca o valor a 1

less\_than\_1:

    mov r0, #MIN\_TIME

    pop pc

;caso seja maior coloca o valor a 10

over\_10:

    mov r0, #MAX\_TIME

    pop pc

;reformata o ptc para medir em ms e depois chama a função 'wait\_user'

measure\_time:

    push lr

    mov r0, #REACT\_TIME\_FREQ

    bl ptc\_init

    bl wait\_user

    pop pc

;espera que o utilizador coloque o SW de USER a 0

wait\_user:

    push lr

wait\_user\_loop:

    bl inport\_read

    mov r1, #USER

    and r0, r0, r1

    mov r1, #OFF

    cmp r0, r1

    bne wait\_user\_loop

    pop pc

;verifica a diferença entre o valor medido e a média de 200 ms

;devolvendo o resultado em r0

get\_result:

    push lr

    bl ptc\_stop

    bl sysclk\_get\_ticks

    mov r2, #AVG\_TIME

    sub r0, r0, r2

    bl show\_result

    pop pc

;demonstra o resultado recebido em r0 e chama a função 'delay\_result'

show\_result:

    push lr

    bl in\_scale

    lsl r0, r0, #TO\_RESULT

    bl outport\_write

    bl reset\_sysclk

    mov r0, #SYSCLK\_FREQ

    bl ptc\_init

    bl delay\_result

    pop pc

;verifica se o resultado está dentro da escala -63 a 63

in\_scale:

    mov r1, #LIMIT

    cmp r0, r1

    bge out\_of\_scale

    mvn r1, r1

    add r1, r1, #1

    cmp r1, r0

    bge out\_of\_scale

    mov pc, lr

;caso não esteja na escala, devolve o valor de -64 em r0

out\_of\_scale:

    mov r0, #OUT\_OF\_RANGE

    mov pc, lr

;coloca o sysclk a 0

reset\_sysclk:

    ldr r0, sysclk\_addr

    mov r1, #OFF

    str r1, [r0]

    mov pc, lr

;dá load ao valor da variável sysclk e devolve-o em r0

sysclk\_get\_ticks:

    ldr r0 , sysclk\_addr

    ldr r0, [r0]

    mov pc, lr

;função de interrupção que incrementa o valor da variável 'sysclk'

isr:

    push r0

    push r1

    ldr r0, PTC\_ADDR

    strb r0, [r0, #PTC\_TIR]

    ldr r0, sysclk\_addr

    ldr r1, [r0]

    add r1, r1, #1

    str r1, [r0]

    pop r1

    pop r0

    movs pc, lr

;inicia o ptc, colocando o valor recebido em r0 como TMR

ptc\_init:

    push lr

    push r0

    bl ptc\_stop

    pop r0

    mov r1, #PTC\_ADDRESS & 0xFF

    movt r1, #(PTC\_ADDRESS >> 8) & 0xFF

    strb r0, [r1, #PTC\_TMR]

    strb r0, [r1, #PTC\_TIR]

    bl ptc\_start

    pop pc

;começa a contagem do ptc

ptc\_start:

    ldr r0, PTC\_ADDR

    mov r1, #PTC\_CMD\_START

    strb r1, [r0, #PTC\_TCR]

    mov pc, lr

;para a contagem do ptc

ptc\_stop:

    ldr r0, PTC\_ADDR

    mov r1, #PTC\_CMD\_STOP

    strb r1, [r0, #PTC\_TCR]

    mov pc, lr

;vai ler o valor de contagem do ptc e devolve-o em r0

ptc\_get\_value:

    ldr r1, PTC\_ADDR

    ldrb r0, [r1, #PTC\_TC]

    mov pc, lr

PTC\_ADDR:

    .word PTC\_ADDRESS

;lê o valor do inport, devolvendo-o em r0

inport\_read:

    ldr r1, inport\_addr

    ldrb r0, [r1]

    mov pc, lr

inport\_addr:

    .word INPORT\_ADDRESS

;escreve o valor de r0 no outport

outport\_write:

    ldr r1, outport\_addr

    strb r0, [r1]

    mov pc, lr

outport\_addr:

    .word OUTPORT\_ADDRESS

sysclk\_addr:

    .word sysclk

.data

sysclk:

    .word 0

.bss

    .stack

    .space STACK\_SIZE

tos:

# Conclusão

Em conclusão, ao compreender e utilizar os recursos do hardware de forma adequada, é possível otimizar o processamento de dados, controlar o fluxo de informações e lidar com interrupções externas de forma eficiente.

Além disso, a organização dos programas em rotinas e a implementação de máquinas de estados possibilitam a criação de sistemas mais complexos.

Ao dominar esses tópicos, os programadores podem obter um maior controle sobre o funcionamento do processador, permitindo o desenvolvimento de soluções mais eficazes e adaptadas às necessidades específicas de cada aplicação.