Generador de Polsos

Dispositius Programables — Enginyeria de Sistemes TIC

Previs

Tasca prèvia 1

Tasca prèvia 1 Escriviu una subrutina de nom to1 que generi un to d'1 kHz durant tu.

```
to1:
        ldi r20, 150
        rcall wait1kHz
        ldi r20,150
        rcall wait1kHz
        ret
wait1kHz:
        ldi r18, 11

wait2:
        ldi r17, 241
wait1:
        subi r17, 0x01
        brne wait1
        subi r18, 0x01
        brne wait2
        sbi PINB, 0x00
        subi r20, 0x1
        brne wait1kHz
        ret
```

Subrutina wait1kHz
En aquesta subrutina es fan tres bucles per fer 1kHz.

16.000.000 Hz / 1000 Hz = 16000 cicles

pero com volem un semicicle per fer la frequencia llavors seran 8000 cicles de clock.

posarem un valor arvitrari a r17 menor que 255 ja que amb aquest valor ens pasem de 8000 cicles cada mig segon. Per tant r17 → 241, i fent la formula dels cicles qu evam fer al previ de la practica passada ens va sortir r18 = 11.

to1
/*en el r20 escriurem cuants cops volem cridar la subrutina wait1kHz, dins la subrutina cada 8000 cicles fem toggle i li restem 1 a aquest registre. Per tant gastem 1 valor del r20 en fer nomes un semiperiode. llavors per a que transcorrin 150ms haurem de cridar la subrutina 2 cops. Això es degut a que 150ms/8000cicles => 150ms/0,0005s => 150ms/0,5ms => 300 cops*/

Tasca prèvia 2

Escriviu una subrutina de nom sl1 que generi una pausa de tu.

```
sl1:
        ldi r20, 150
        rcall wait1kHz_2
        ldi r20, 150
        rcall wait1kHz_2
        ret
wait1kHz_2:
        ldi r18, 11

wait2_2:
        ldi r17, 241
wait1_2:
        subi r17, 0x01
        brne wait1_2
        subi r18, 0x01
        brne wait2_2
        nop
        subi r20, 0x1
        brne wait1kHz_2
        ret
```

En aquest cas farem una subrutina de la mateixa durada que el previ 1 pero sense canviar el led d'estat per tant psarem els mateixos valors que anteriorment pero en comptes de posar-hi
SBI PINB, 0x00 → que el que fa es fer toggle en la pota 8 del port B

Donsc hi possarem un simple nop per fer passar un clock.

Tasca prèvia 3

Escriviu una subrutina de nom punt que, aprofitant les anterior subrutines to1 i sl1, generi el punt de Morse.

Aprofitant les subrutines anteriors hem creat la següent subrutina punt:

```
punt:
        rcall to1
        rcall sl1
        rjmp loop
```

Bàsicament cridem to1 que ens fa un punt i seguidament un silenci per deiexar espai en el següent punt o raya. I finalment un rjmp que torni al bucle.

Tasca prèvia 4

Escriviu una subrutina de nom ratlla que, aprofitant les anterior subrutines to1 i sl1, generi la ratlla de Morse.

```
ratlla:
        rcall to1
        rcall to1
        rcall to1
        rcall sl1
        rjmp loop
```

Seguint la mateixa dinàmica del punt, com que la ratlla dura 3 punt doncs simplement cridem tres punts i un silenci per generar un temps abans del següent punt o ralla. i seguidament tornem al bucle amb rjmp.

Tasca prèvia 5

Escriviu en el fitxer p4-codi1.s el programa de codi Morse que generi un punt o una ratlla en funció del polsador premut. Definiu els pins d'entrada dels dos polsadors i el pin de sortida on es generarà el senyal de Morse.

```
main:
        ldi r16, 0xFF
        ldi r19, 0x0
        ldi r21, 0b10010000
        out DDRB_o,r16
        out DDRD_o,r19 /* definim port D com a entrada */
        out 0x35, r19 /* activem bit PUD */
        out PORTD_o, r21 /* definim els pins que utilitzarem pull up */
```

per aconseguir això primer de tot hem de configurar els pull up per a que no hi hagin valors indessitgats.

1. configurem el port D com a sortida per posar-hi els polsadors.
2. activem el bit pud possant tots els bits d'aquests a 0 per a que es faci el pull up tal i com diu el manual.
3. per ultiim definim quins seran els ports que faran servir el pull up, en aquest cas 2 ja que utilitzarem 2 botons.

```
loop:
        ldi r22, 0b00010000
        ldi r23, 0b10000000
        in r24, PIND
        and r24, r22
        brne punt
        in r25, PIND
        and r25, r23
        brne ratlla
        rjmp loop
```

en el loop hi collocarem un in per saber les entrades del portd i les posarem a r24, seguidament farem un and amb r22 previament definit per saber si s'ha premut el boto. si el resultat dona 0 s'activarà el flag Z i s'executara la subrutina punt. si el flag z no s'activa passarem a mirar la seguent operacio i farem el mateix pero amb la subrutina ratlla.

Tasca prèvia 6

Inseriu en el fitxer p4-exemple.s els comentaris necessaris per tal d'explicar amb tot detall què fan les línies del codi.

```
.set DDRB_o , 0x4
.equ PORTB_o , 0x5
DDRD_o = 0x0a
PORTD_o = 0x0b
TCCR0A_o = 0x24
TCCR0B_o = 0x25
OCR0A_o = 0x27


.global main
waitabit: ldi r19,41 /*subrutina feta en anteriors practiques que fan un bucle d'espera
 ( semiperiode 500ms = periode de 1000ms*/
wait3: ldi r18,0xFF
wait2: ldi r17,0xFF
wait1: subi r17,0x01
        brne wait1
        subi r18,0x01
        brne wait2
        subi r19,0x01
        brne wait3
        ret

main:   ldi r16,0b01000000
        out DDRD_o,r16 /*configurem bit 6, pota 6, com a sortida i les demes com a entr
ada en el portD*/
        ldi r16,0b00000000
        out PORTD_o,r16/*definim els pins del portD = 0*/
        ldi r16,124
        out OCR0A_o,r16 /*definim limit timer amb 124 */
        ldi r16,0b01000010 /* els dos primers bits es per activar el toggle OCOA on enc
enem el timer. Els seguents dos bits son per tenir OCB0 desconenctat. I els dos ultims
son per posar el mode CTC (clear data on compare match), per feer toggle just cuan el t
immer arriba al seu limit. Els demes bits son reservats*/
        out TCCR0A_o,r16
        ldi r16,0b00000011/* amb els primers dos bits i amb els tres ultims posem el pr
escaler a 64. el 4rt bit es per WGM02 que serveix pr a acabar de configurar amb el WGM0
1 i 00 del TCCR0A i el CTC */
        out TCCR0B_o,r16

loop:  call waitabit /*cridem waitabit*/
        in r20,TCCR0A_o/*mirem el valor del timer*/
        cbr r20,0b01000000/* posem el segon bit de r20 a 0*/
        out TCCR0A_o,r20/*parem el timer ja que hem posat el segon bit a 0*/
        call waitabit/*cridem waitabit*/
        in r20,TCCR0A_o/*llegim timer*/
        sbr r20,0b01000000/*posem a 1 el segon bit de r20*/
        out TCCR0A_o,r20/* encenem el timer despres de posar el segon bit a 1*/
        rjmp loop
```

Tasca prèvia 7

Modifiqueu el fitxer p4-codi1.s aprofitant els recursos que s'usen en el fitxer p4-exemple.s. Si cal, re-definiu els pins. Anomeneu el fitxer modificat p4-codi2.s

Falta entendre i aprendre com funciona al 100% el timer per a poder explicar amb definicio com funciona aquest.

Tasca prèvia 8

Penseu com es podria detectar en qualsevol instant la situació anòmala en què els dos polsadors estan simultàniament premuts. Modifiqueu el fitxer p4-codi2.s per tal que quan es produeixi aquesta simultaneïtat no s'enviï cap codi Morse i s'indiqui que es tracta d'una anomalia encenent el LED de la placa Arduino. Anomeneu el fitxer modificat p4-codi3.s.

" " "

Tasques

Tasca 9

Verifiqueu que heu comprès què fa el codi del p4-exemple.s

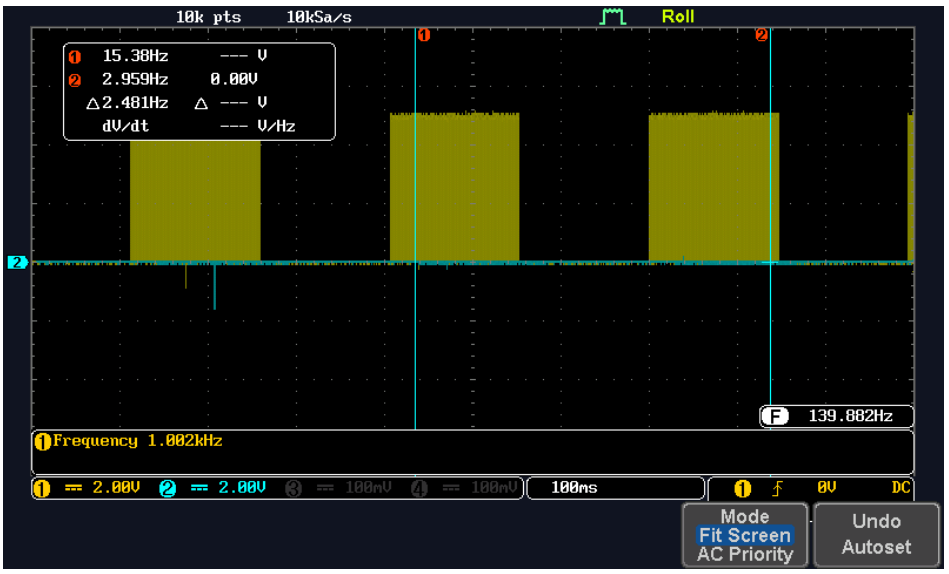
Efectivament hem compres el que fa el codi d'exemple tal i com esta explicat en exemple.s explicat en el previ 6

```
.set DDRB_o , 0x4
.equ PORTB_o , 0x5
DDRD_o = 0x0a
PORTD_o = 0x0b
TCCR0A_o = 0x24
TCCR0B_o = 0x25
OCR0A_o = 0x27


.global main
waitabit: ldi r19,41 /*subrutina feta en anteriors practiques que fan un bucle d'espera
 ( semiperiode 500ms = periode de 1000ms*/
wait3: ldi r18,0xFF
wait2: ldi r17,0xFF
wait1: subi r17,0x01
        brne wait1
        subi r18,0x01
        brne wait2
        subi r19,0x01
        brne wait3
        ret

main:   ldi r16,0b01000000
        out DDRD_o,r16 /*configurem bit 6, pota 6, com a sortida i les demes com a entr
ada en el portD*/
        ldi r16,0b00000000
        out PORTD_o,r16/*definim els pins del portD = 0*/
        ldi r16,124
        out OCR0A_o,r16 /*definim limit timer amb 124 */
        ldi r16,0b01000010 /* els dos primers bits es per activar el toggle OCOA on enc
enem el timer. Els seguents dos bits son per tenir OCB0 desconenctat. I els dos ultims
son per posar el mode CTC (clear data on compare match), per feer toggle just cuan el t
immer arriba al seu limit. Els demes bits son reservats*/
        out TCCR0A_o,r16
        ldi r16,0b00000011/* amb els primers dos bits i amb els tres ultims posem el pr
escaler a 64. el 4rt bit es per WGM02 que serveix pr a acabar de configurar amb el WGM0
1 i 00 del TCCR0A i el CTC */
        out TCCR0B_o,r16

loop:  call waitabit /*cridem waitabit*/
        in r20,TCCR0A_o/*mirem el valor del timer*/
        cbr r20,0b01000000/* posem el segon bit de r20 a 0*/
        out TCCR0A_o,r20/*parem el timer ja que hem posat el segon bit a 0*/
        call waitabit/*cridem waitabit*/
        in r20,TCCR0A_o/*llegim timer*/
        sbr r20,0b01000000/*posem a 1 el segon bit de r20*/
        out TCCR0A_o,r20/* encenem el timer despres de posar el segon bit a 1*/
        rjmp loop
```

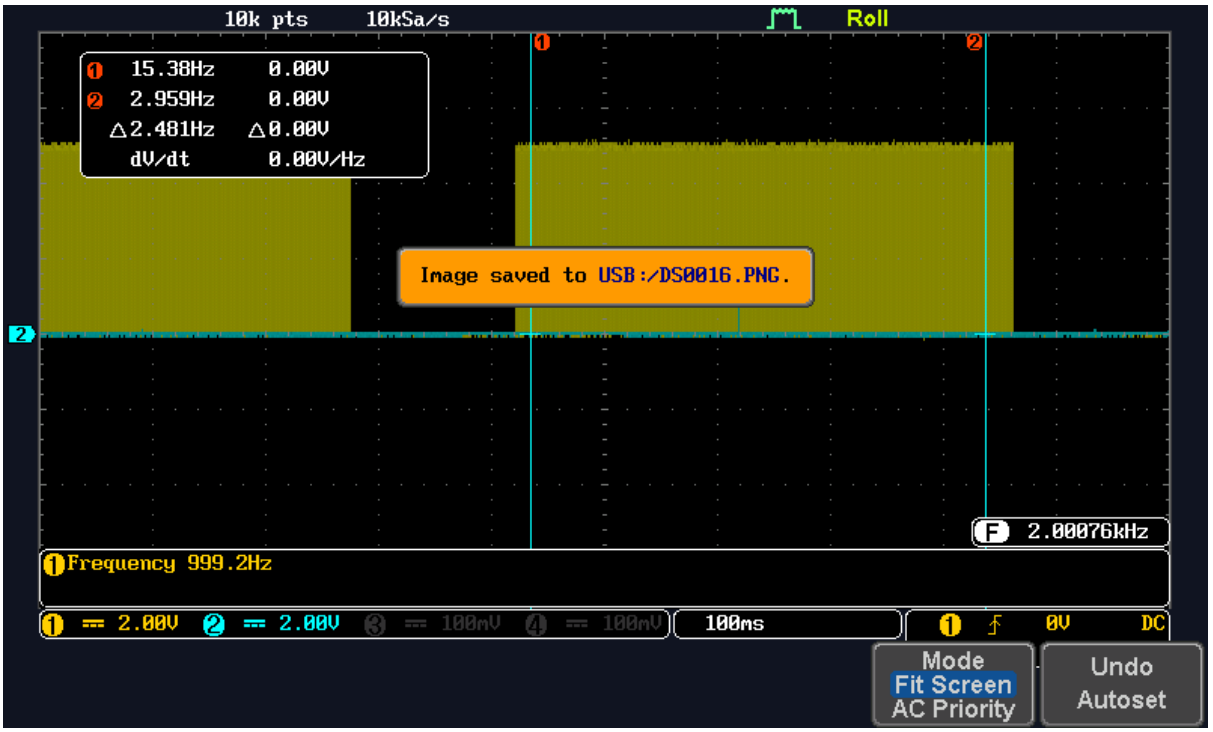
Tasca 10

Comproveu el correcte funcionament de p4-codi1.s

Tal i com hem definit en la progrmacia del codi demanat hauriem de poder veure com cuan apretem el botò veiem una frequencia d'1kHz durant to1 = 150ms, i si apretem l'altre bitò veurem una frequencia de 1kHz durant tres to1 = 450ms.



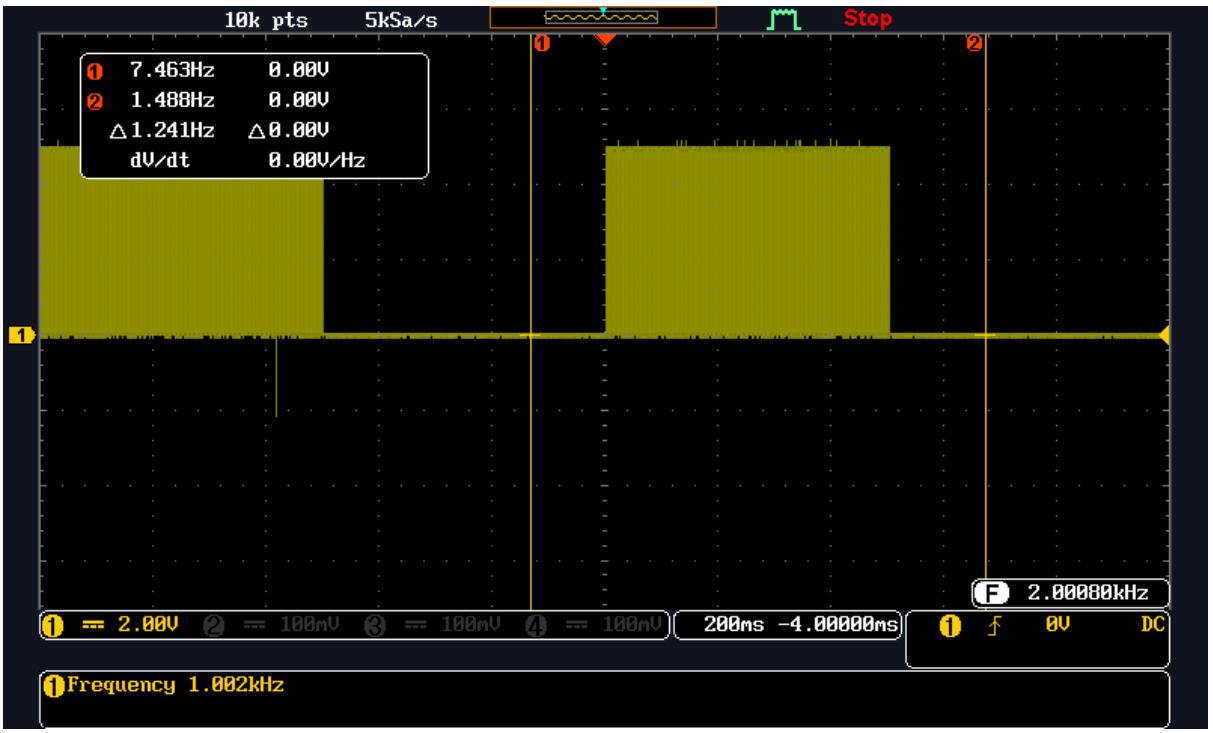
veiem el correcte funcionament de la ratlla de 150ms de duranda. aqui tambe observem el correcte funcionament de 1khz



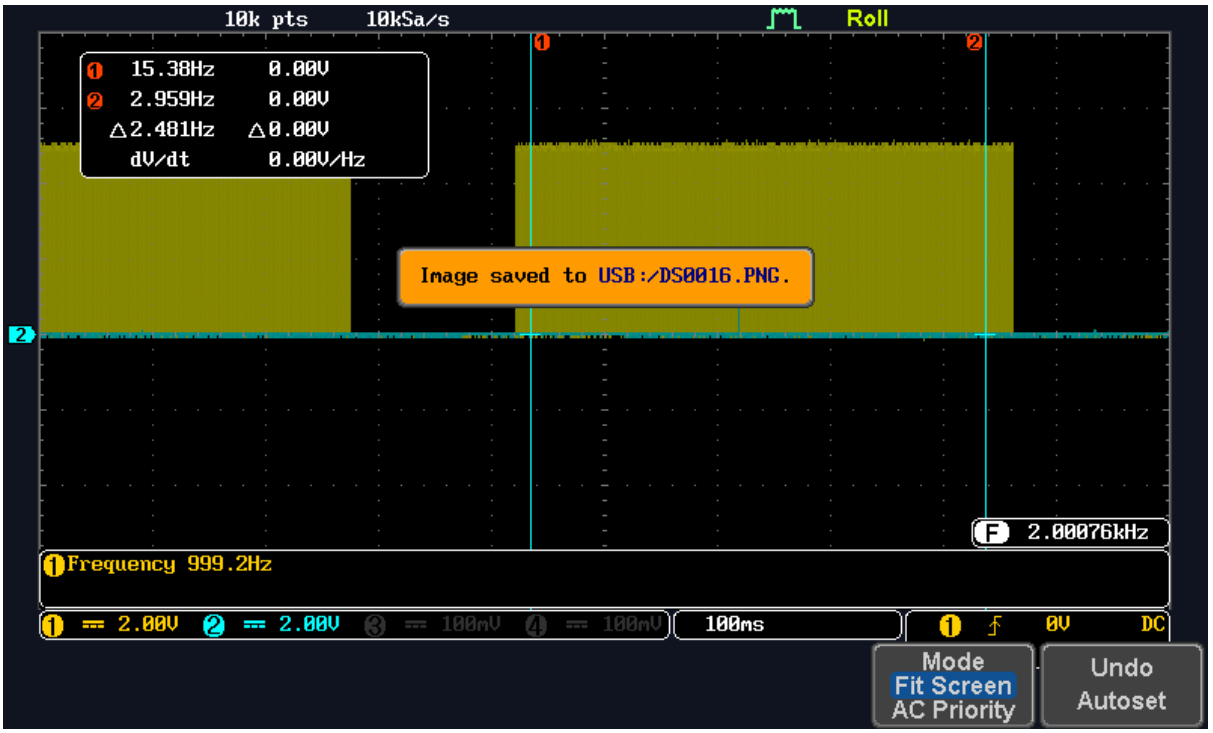
veiem el correcte funcionament de la ratlla de 450ms de duranda

Tasca 11

Comproveu el correcte funcionament de p4-codi2.s



veiem com ens funciona correctament el timer en el punt encara que no l'acabo d'entendre correctament. Vaig necessitar l'ajuda del professorat.

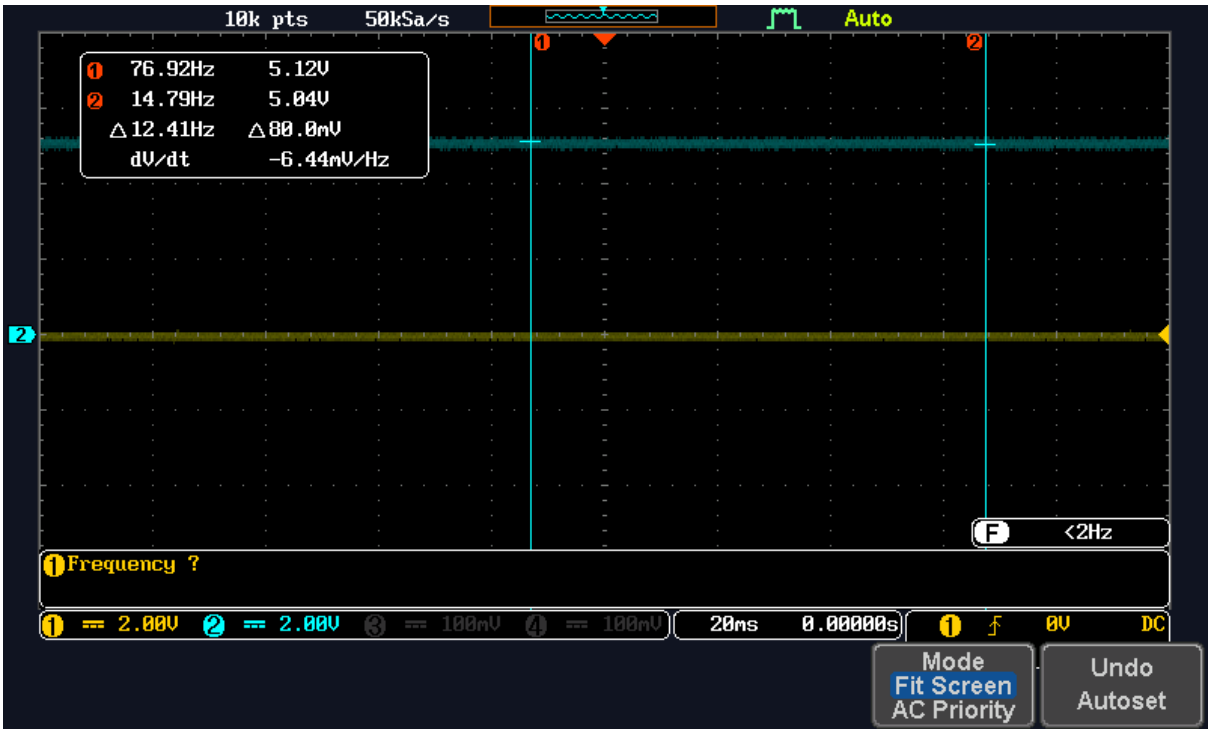


veiem com ens funciona correctament el timer en la ratlla encara que no l'acabo d'entendre correctament. Vaig necessitar l'ajuda del professorat.

Tasca 12

Comproveu el correcte funcionament de p4-codi3.s

Hauriem de veure com al polsar els dos botons a l'hora ens salta el led d'error.



efectivament veiem com s'encen el led en el canal dos (blau) al preme els dos botons a l'hora.