FIB. Computer Interfacing *Segon parcial* **13/1/2021**

Cognoms, Nom _____ **DNI** _____

Una resposta sense justificar es considerarà nul·la !

P1. (1 punt) Disposem d'un circuit basat en el PIC18F45K22 per controlar un motor amb un senyal PWM de 5 kHz. Al programa s'utilitza el mòdul CCP4 vinculat amb Timer6. S'ha configurat un *Prescaler* de 1:4 i un valor de PR6=74. Malauradament, el cristall oscil·lador del circuit s'ha cremat i hem de substituir-lo.

¿De quina freqüència hem de comprar el cristall de quars?

Sabem que cada cicle de PWM dura un període $T_{PWM} = \frac{1}{5\,kHz} = 200\mu s$
La duració d'un cicle de PWM ve donada per: $T_{PWM} = (PRx + 1) \cdot \frac{4}{F_{osc}} \cdot PRE = 75 \cdot \frac{4}{F_{osc}} \cdot 4 = 200\mu s$
Si resolem per Fosc ens dóna 6MHz

Amb les configuracions donades a l'enunciat, i si no usem els bits DC4B<1:0> (deixem el seu valor per defecte a 0): ¿quants *duty cycles* diferents podríem tenir?

Si no fem servir els bits DC4B, només ajustem el byte del registre CCPR4L. El valor d'aquest registre es comparará amb PR6. Per tant, el valor de PR6 és el que ens limita el número de *duty cycles* diferents.

Com que PR6=74, podrem ajustar 75 *duty cycles* diferents.

Si utilitzéssim també els bits DC4B<1:0>, ¿quants *duty cycles* diferents podríem tenir?

Per cadascun dels valors que puguem posar a CCPR4L, el perifèric del micro ens dóna la possibilitat d'ajustar els 2 bits DC4B, per tant podem escollir entre 4 valors diferents (00, 01, 10, 11).

Si ajustant el CCPR4L podíem escollir entre 75 *duty cycles*, ara en afegir els 2 bits DC4B<1:0>, multipliquem per 4 les opcions. Per tant, podrem ajustar 300 *duty cycles* diferents.

P2. (1 punt) Donat el següent codi que s'executa al PIC del laboratori (amb Fosc=8MHz):

```
void main() {
  TRISCbits.TRISC1=0;
  T0CONbits.T0CS=0;
  INTCONbits.TMR0IF=0;
  while(1) {
    while(!TMR0IF);
    TMR0IF=0;
    LATC1=!LATC1;
  }
}
```

Quina és la freqüència de la ona quadrada que generem pel pin RC1?

Llegint el codi veiem que estem usant el Timer0. Cada cop que el Timer faci overflow, posarà a 1 el Flag d'interrupció, i el programa canviarà el nivell de la senyal del pin RC1.
Només tenim explícitament la configuració del Clock Source (T0CS=0): Internal Instruction cycle (Fosc/4). La resta de paràmetres els hem d'obtenir del valor dels bits per defecte quan encenem el micro. Podem consultar aquest valor a la definició del T0CON al Formulari. Veiem que tots els bits per defecte son '1'. Per tant, el Timer és de 8 bits, no fa servir Prescaler, i el Timer ja està activat i comptant. Com que hi haurà overflow cada 256 ticks, tindrem un Flag d'interrupció cada: $T_{TMR0IF} = \frac{4}{F_{OSC}} \cdot 256 = \frac{4}{8MHz} \cdot 256 = 128\mu s$
Cada 2 Flags d'interrupció tenim un període complet de la senyal del pin RC1.
Per tant, $T_{RC1} = 2 \cdot 128\mu s = 256\mu s$. De manera que la freqüència serà $\frac{1}{256\mu s} = 3906{,}25\ Hz$

P3. (1,5 punts) Hem connectat la sortida d'ona quadrada d'un generador de funcions al pin CCP1 (RC2) del nostre PIC. El micro treballa amb una freqüència Fosc=8MHz. La intenció és mesurar el temps que dura el període d'aquesta senyal connectada al CCP1. Tenim el següent codi programat al micro:

```
void main() {
  ANSELCbits.ANSC2=0;
  TRISCbits.TRISC2=1;
  T1CON = 0x33;
  IPR1bits.CCP1IP=1;
  PIE1bits.CCP1IE=1;
  PIR1bits.CCP1IF=0;
  RCONbits.IPEN=0;
  INTCONbits.GIE=1;
  INTCONbits.PEIE=1;
  CCP1CON = 0x05;
  while(1);
}
```

```
unsigned int lastTmr;
unsigned int nTicksPeriod;
   // el 1r valor gravat a nTicksPeriod
   // no serà vàlid. A partir del 2n, sí.
   // No us preocupeu per aquest detall.

void interrupt highRSI() {
  if (CCP1IE && CCP1IF)
  {
    CCP1IF=0;
    nTicksPeriod=CCPR1-lastTmr;
    lastTmr=CCPR1;
  }
}
```

Si el valor calculat a **nTicksPeriod** és de 25000, quina és la freqüència de la senyal que treu el generador de funcions?

Veiem al codi que CCP1CON el configurem en mode *Capture (cada flanc ascendent)*. L'estratègia és que cada flanc ascendent de la senyal connectada a CCP1 provocarà una crida a interrupció. A la RSI llegirem CCPR1 (valor gravat de ticks del Timer vinculat), i fem la resta amb l'anterior lectura.
Per tant, necessitem saber la duració d'1 tick de Timer1.
Observant el valor de T1CON, podem saber que el clock source és (Fosc/4) i fem servir un Prescaler de 1:8. Per tant, la duració d'1 tick de Timer1 serà de $T_{TMR1} = \frac{4}{F_{OSC}} \cdot 8 = \frac{4}{8MHz} \cdot 8 = 4\mu s$
Si sabem que han passat 25000 ticks entre flancs ascendents, el període de la senyal a CCP1 serà:
$T_{CCP1} = 4\mu s \cdot 25000 = 100ms$
Així doncs, la freqüència de la senyal serà de 10Hz.

Si la senyal fos de freqüència 2.5 Hz, ¿quant valdria el valor calculat a **nTicksPeriod**? Tindríem algun problema amb el càlcul? Si es així, descriu quins canvis faries al codi per a evitar el problema (no cal que escriguis el codi, només descriu els canvis).

El període d'una senyal de 2.5 Hz seria de $\frac{1}{2.5Hz} = 400ms$. Si cada tick de Timer1 són 4µs, la quantitat de ticks que el programa hauria de calcular en nTicksPeriod seria de $\frac{400ms}{4\mu s} = 100000\ ticks$.
Aquest algorisme ens donarà problemes per mesurar el període de senyals tant lentes, doncs ens basem en fer una simple resta del CCPR1 (que és de 16 bits). Exemple: per una senyal que duri 65536 ticks, el Timer haurà donat una volta completa, i la resta valdrà 0.

Una possible solució seria comptar el número d'*overflows* del Timer1 que hi ha hagut entre una lectura de CCPR1 i la següent. Per incrementar el comptador d'*overflows* podem usar la interrupció TMR1IF, provocant una crida a RSI on s'incrementi el comptador. A la interrupció de CCP1 llegirem el comptador i el resetejarem a 0 de nou.
Al nou algorisme, per prendre la lectura dels ticks transcorreguts, sumarem:
(CCPR1 + numOverflows*65536) - lastTmr
També haurem de treballar amb una variable de tipus més gran, que admeti gravar números >=65536. Per exemple: unsigned long.

FIB. Computer Interfacing *Segon parcial* **13/1/2021**

Cognoms, Nom _____ **DNI** _____

Una resposta sense justificar es considerarà nul·la !

P4. (0,5 punts) Quants bits de resolució tindrà un A/D Flash constituït per 15 comparadors?

Amb N comparadors obtenim N+1 nivells. Llavors 15 comparadors configuren 16 nivells.
Per codificar 16 nivells diferents són necessaris 4 bits, 4 = log2(16)

P5. (0,5 punts) Quants bits de resolució tindrà un A/D d'aproximacions successives que realitza la conversió en 16 passes? No considereu com a pas de conversió ni la preparació ni l'entrega del resultat.

A cada pas d'aproximació s'obté un nou 1 bit, llavors amb 16 passes s'obtenen 16 bits.

P6. (1 punt) Es vol implementar un sensor de temperatura que permeti obtenir 1º de resolució en el rang de temperatures 0º a 90º amb el sensor TC1047A. Quina hauria de ser la resolució del A/D en bits si les tensions de referència són 0v i 5v?

The output voltage range for these devices is typically 100 mV at –40°C, 500 mV at 0°C, 750 mV at +25°C, and +1.75 V at +125°C. As shown in Figure 12.14, the TC1047A has a 10 mV/°C voltage slope output response.

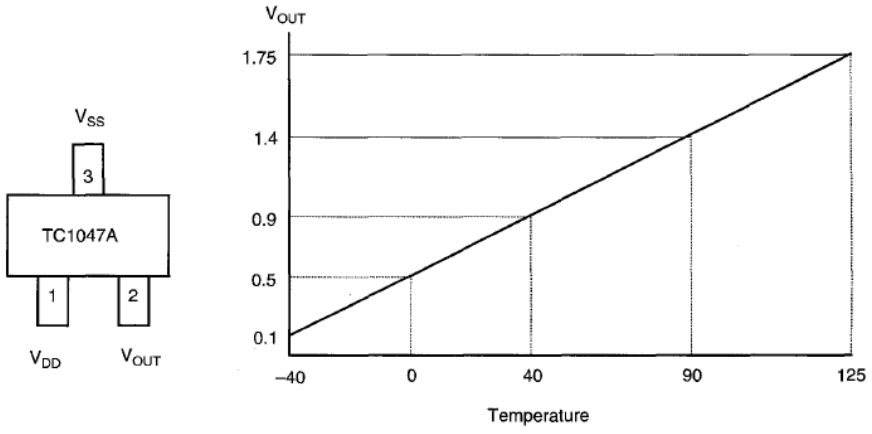


Figure 12.14 ■ TC1047A V_{OUT} vs. temperature characteristic

De 1,4 volts a 0,5volts hi hauran d'haver 90 divisions. És a dir 0,9v/90 divisió = 1/100 volts per divisió. Llavors en 5 volts hi hauran d'haver 500 divisions (com a mínim). En conseqüència necessitem un A/D de 9 bits = [log2(500)]. Cal arrodonir per damunt per garantir el mínim de 500 divisions.

P7. (0,5 punts) Quin seria el temps total de mostreig si ACQT<2:0> = 100, ADCS<2:0> = 101, Fosc = 8 Mhz?

REGISTER 17-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT<2:0>			ADCS<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **ADFM:** A/D Conversion Result Format Select bit
1 = Right justified
0 = Left justified

bit 6 **Unimplemented:** Read as '0'

bit 5-3 **ACQT<2:0>:** A/D Acquisition time select bits. Acquisition time is the duration that the A/D charge holding capacitor remains connected to A/D channel from the instant the GO/DONE bit is set until conversions begins.
000 = 0(1)
001 = 2 TAD
010 = 4 TAD
011 = 6 TAD
100 = 8 TAD
101 = 12 TAD
110 = 16 TAD
111 = 20 TAD

bit 2-0 **ADCS<2:0>:** A/D Conversion Clock Select bits
000 = FOSC/2
001 = FOSC/8
010 = FOSC/32
011 = FRC(1) (clock derived from a dedicated internal oscillator = 600 kHz nominal)
100 = FOSC/4
101 = FOSC/16
110 = FOSC/64
111 = FRC(1) (clock derived from a dedicated internal oscillator = 600 kHz nominal)

Note 1: When the A/D clock source is selected as FRC then the start of conversion is delayed by one instruction cycle after the GO/DONE bit is set to allow the SLEEP instruction to be executed.

$T_{Total\ AD} = T_{ACQ} + 12\ T_{AD}$
$T_{ACQ} = 8\ T_{AD}$
$T_{AD} = 16/F_{OSC}$

$T_{Total\ AD} = 40$ useg

P8. (0,5 punts) Seria possible utilitzar la configuració de l'exercici P7 per adquirir un senyal d'àudio (20Hz 20KHz) sense que es produeixi *aliasing*? I senyal d'ultrasons d'ecografia (1MHz)?
La freqüència de mostreig és $F_{mostreig} = 1/ T_{Total\ AD} =$ 25 KHz
En cap dels dos casos la configuració emprada permet evitar l'*aliasing*. Necessitem un mínim de 40KHz per àudio i 2MHz per ecografia.

Cognoms, Nom _____ **DNI** _____

Una resposta sense justificar es considerarà nul·la !

P9. (2 punts) Justifica per què és millor fer servir una Fosc de 3.6864Mhz que una de 8Mhz si volem transmetre dades pel port sèrie d'un PIC18F45k22 a 115200 bauds.

Fent servir un clock de 8MHz, el millor baudrate que podem aconseguir si volem transmetre a 115200 és 117647 amb SYNC=0, BRG16=1, BRGH=1 i SPBRG=16. Aquesta configuració suposa un 2,12% d'error.

Fent servir un clock de 3.6864MHz, podem aconseguir transmetre a 115200 de forma exacta amb diverses configuracions, per exemple amb SYNC=0, BRG16=1, BRGH=1 i SPBRG=7. Aquesta configuració suposa un 0% d'error.

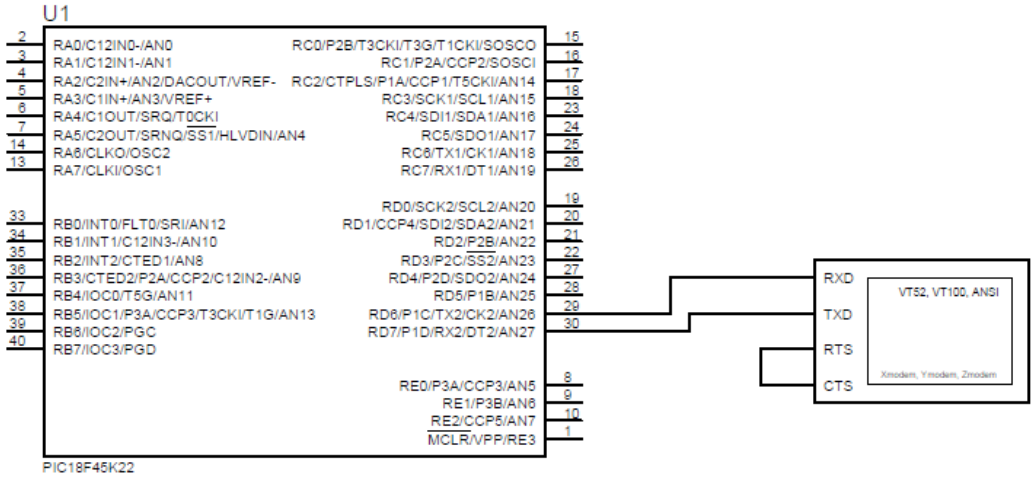
Configura els bits necessaris dels registres TxSTA, RCSTA, BAUDCON i SPBRG per enviar i rebre dades pel port sèrie asíncron a 115200bauds, 8 bits de dades, sense paritat i un bit d'stop. Fosc=3.6864Mhz

Hi ha diverses configuracions que ho permeten. Mostrem una que ho permet suposant que fem servir l'UART1

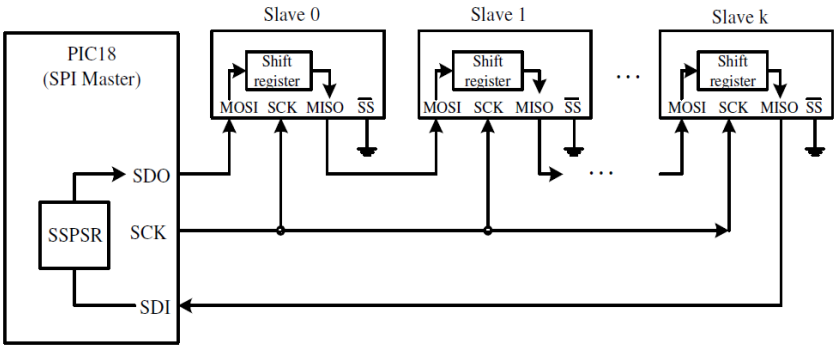
```
SPEN=1;
TRISCbits.RC7=1; //a l'enunciat no ho demana. Ho afegim per completar la solució
TRISCbits.RC6=1; //a l'enunciat no ho demana. Ho afegim per completar la solució
SPBRG=7;
BRGH=1;
BRG16=1;
SYNC=0;
TX9=0;
SENDB=0;
TXEN=1;
RX9=0;
CREN=1;
```

Realitza la connexió del virtual terminal de proteus amb un PIC18F45k22 si volem transmetre per l'EUSART2

Realitzem el connexionat per enviar i rebre:



P10. (1 punt) Hem connectat un PIC18 fent de master a un bus SPI tal i com es mostra a la següent imatge:



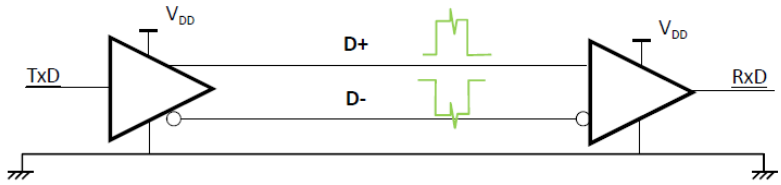
Si tenim 20 slaves, quant trigarem a enviar un missatge de 8 bits a l'últim slave del bus? Suposeu que el data rate del bus és de 10Mbps (Mega bits per segon)

Al bus SPI no tenim overhead en les dades que enviem. Aprofitant el connexionat circular del bus podem enviar dades a l'últim slave del bus, per això haurem d'enviar 8 bits a cada slave: 8bits * 20slaves = 160bits

160bits*(1segon/10Mbits)=16µs

P11. (0.5 punts) Quina avantatge aporta el fet que l'USB disposi de dos fils de dades (D+ i D-)?

Gràcies a disposar de dos fils de dades l'USB permet enviar i rebre les dades de forma diferencial. Les mateixes dades que van per D+ es transmeten negades per D-. Si algun soroll afecta a la transmissió, al rebre i reconstruir aquella dada, el soroll serà quasi eliminat



En definitiva, el principal avantatge és immunitat al soroll.

En el cas de l'USB també es fa servir per indicar si connectem un dispositiu de full o low speed: