Computer Interfacing (CI) FIB. Segon parcial **13/01/2023**

Cognoms, Nom _____ DNI _____

Tota resposta sense justificar es considerarà nul·la !

P1. (2 punts)
El següent codi genera una ona quadrada amb el CCP1 d'una freqüència tal que, si la reproduïm amb un altaveu, espanta a una guineu que sempre vol robar les coses a una bona amiga nostra.

```
#define XTAL_FREQ 8000000
int semiperiod;
void main (){
        ANSELC=0;
        TRISCbits.TRISC2=0;
        CCP1CONbits.CCPxM=0b0010;
        CCPTMRS0=0;
        semiperiod=2000;
        CCPR1=semiperiod;
        TMR1GE=0;
        T1CON=0x03;
        PEIE=1; CCP1IE=1; GIE=1;
        while(1);
}
```

```
void interrupt no_robis(){
        if(CCP1IE && CCP1IF){
                CCPR1+=semiperiod;
                CCP1F=0;
        }
}
```

1.1 Pots indicar quina és la freqüència que genera aquest codi? (1p)

Veiem que el codi configura el CCP en mode compare toggle associat al timer 1, que està configurat sense prescaler i Fosc/4 com a clock source. El match entre timer1 i el CCPR1 serà cada 2000 tics de timer1, llavors el pin del CCP fa toggle cada 2000 tics (que serà el semiperiode de la senyal generada):

1 tic de timer 1 = 4/8M = 0,5µs
semiperiode = 2000 tics *0,5µs = 0,001s
període = 0,002s
Freqüència=1/0,002s=500Hz

1.2 Una persona d'una altra facultat vol reproduir el mateix so fent servir CCPxCONbits.CCPxM=0b1100 com a configuració del mòdul CCP1. Creus que aconseguirà reproduir la mateixa freqüència que el codi del primer apartat? (1p)

En aquest cas, la persona de l'altra facultat, està configurant el CCP en mode PWM. Recordem que en aquest cas el CCP treballarà amb un timer de 8 bits (2, 4 o 6). Podem demostrar si és correcte que pot generar una freqüència de 500Hz trobant una configuració que ho aconsegueixi o buscant les freqüències màximes i mínimes que es poden generar amb el PWM. En aquesta solució ho demostrarem trobant una configuració que genera una freqüència de 500Hz:

1/500Hz=(PRx+1)*4*1/8M*PRE

Es pot resoldre que amb PRE =16 i PRx=249 es generen el 500Hz (s'ha de configurar el duty cycle al 50% per aconseguir que l'ona generada sigui quadrada).

P2. (1 punts)

Hem descobert que alguns pokemons són més fàcils de capturar quan la seva veu té una freqüència superior a 100KHz. Per millorar les nostres estadístiques de captura volem digitalitzar el senyal provinent d'un micròfon amb un PIC18F45K22 (FOSC= 1 MHz) i així llançar pokeballs només quan les nostres opcions de capturar el pokemon siguin prou bones. Si suposem que TAD>=1µs i TACQ>7.5µs, Quin és el millor temps de mostreig d'AD que pots aconseguir? Amb quin valor del registre ADCON2 ho aconsegueixes? Aquesta configuració de l'AD permet capturar sense aliasing el so en el rang de freqüències que ens interessa?

La millor configuració que satisfà les condicions de l'enunciat és:
TAD=2/Fosc=2µs -> ADCS=0b000
TACQ=4TAD=8µs -> ACQT=0b010
ADCON2=X0010000

El temps d'una mostra serà 11TAD+TACQ+1TCY. En la correcció també s'accepta 12TAD+TACQ i també s'ha acceptat si no s'ha tingut en compte el temps de descàrrega del condensador del circuit de sample&hold, ja que per llegir el valor de l'ADRES no cal esperar a que s'hagi descarregat (tot i que sí s'ha d'esperar per demanar a l'AD la següent mostra).
Temps=12*2+8=32µs
F Mostreig=31,250KHz, insuficient per satisfer el criteri de Nyquist. No podem asegurar que no es produeixi aliasing.

P3. (1 punts)

Configurem un conversor AD de 10 bits amb Vref-=1V i amb Vref+4V i ADFM=1. Quin valor en volts hi ha a Vin si després de la conversió trobem que ADRESH=0x01 i ADRESL=0x01?

ADRES=ADRESH<<8+ADRESL=ADRESH*256+ADRESL=257

$$ADRES = 2^N - 1 * (V_{IN} - V_{REF-})/(V_{REF+} - V_{REF-})$$

Vin=1,75V

P4. (1 punts)

Quants bits d'AD són necessaris si volem mesurar la distància a un objecte mitjançant un sensor que ens dóna tensions entre 2V i 4V, corresponents a distàncies d'entre 1 i 4 metres (de manera lineal) i necessitem una resolució de 0.01 metres. Les tensions de referencia són VREF– = 0V i VREF+ = 5V.

Necessitem una resolució de 3m/0,01metres=300 steps d'AD entre 2V i 4V.
Com que els valors de referència d'AD no són entre 2V i 4V, sinó que es troben entre 0V i 5V sabem que necessitem 750 steps a resolució completa.
$2^N = 750$ -> N=9,55 -> necessitarem 10 bits d'AD.

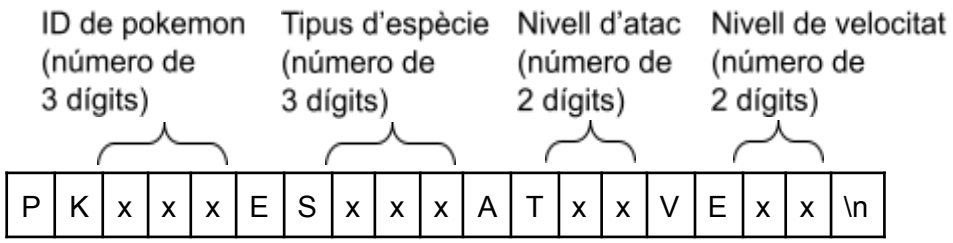
Computer Interfacing (CI) FIB. Segon parcial **13/01/2023**

Cognoms, Nom _____ **DNI** _____

Tota resposta sense justificar es considerarà nul·la !

P5. (2 punts)

El mateix microcontrolador PIC18F45K22 de la pregunta P2 (Fosc=1MHz) dedicat a la captura de pokemons necessita enviar la informació de cada pokemon capturat a través d'una línia sèrie UART a un ordinador central que emmagatzema les dades.
Per cada pokemon, enviarem la següent trama formada per caràcters ASCII:



Exemple:

P	K	1	2	2	E	S	0	2	1	A	T	1	1	V	E	3	9	\n

Taula ASCII *(apartat 5.2)*

048 0	064 @	080 P	096 `	112 p
049 1	065 A	081 Q	097 a	113 q
050 2	066 B	082 R	098 b	114 r
051 3	067 C	083 S	099 c	115 s
052 4	068 D	084 T	100 d	116 t
053 5	069 E	085 U	101 e	117 u
054 6	070 F	086 V	102 f	118 v
055 7	071 G	087 W	103 g	119 w
056 8	072 H	088 X	104 h	120 x
057 9	073 I	089 Y	105 i	121 y
058 :	074 J	090 Z	106 j	122 z
059 ;	075 K	091 [	107 k	123 {
060 <	076 L	092 \	108 l	124 |
061 =	077 M	093]	109 m	125 }
062 >	078 N	094 ^	110 n	126 ~
063 ?	079 O	095 _	111 o	127 ⌂

La configuració de la línia sèrie serà asíncrona, sense paritat, amb 8 bits de dades, 1 bit d'stop i amb Baudrate=4800.

5.1 Configura els bits necessaris dels registres TXSTA, RCSTA, BAUDCON i SPBRG per a que puguem enviar les dades descrites abans amb el perifèric UART1 del micro. I especifica el % d'error que cometem en el Baudrate amb la configuració triada. (1p)

Provem les fórmules per veure quina ens dóna un Baudrate real (BR_{real}) amb un % d'error acceptable.

Divisor 64: $n = \frac{F_{osc}}{64 \cdot 4800} - 1 \cong 2.26$	Divisor 16: $n = \frac{F_{osc}}{16 \cdot 4800} - 1 \cong 12.02$	Divisor 4: $n = \frac{F_{osc}}{4 \cdot 4800} - 1 \cong 51.08$
Arrodonim a **SPBRG=2**	Arrodonim a **SPBRG=12**	Arrodonim a **SPBRG=51**
$BR_{real} = \frac{F_{osc}}{64 \cdot (2+1)} \cong 5208.3$	$BR_{real} = \frac{F_{osc}}{16 \cdot (12+1)} \cong 4807.7$	$BR_{real} = \frac{F_{osc}}{4 \cdot (51+1)} \cong 4807.7$
$\%err = \frac{5208.3-4800}{4800} \cdot 100 \cong 8.5\%$	$\%err = \frac{4807.7-4800}{4800} \cdot 100 \cong 0.16\%$	$\%err = \frac{4807.7-4800}{4800} \cdot 100 \cong 0.16\%$
$\|\%err\|$ superior al 5%, **descartat**	$\|\%err\|$ inferior al 5%, **acceptable**	$\|\%err\|$ inferior al 5%, **acceptable**

Qualsevol dels divisors 16 ó 4 ens serveix. Escollim el divisor 16, per exemple. La taula 16-3 diu el valor que hem d'assignar a SYNC, BRG16 i BRGH. L'enunciat diu comunicació asíncrona, per tant **SYNC=0** ja està bé. Els altres dos bits podem escollir dos casos, ens quedem arbitràriament amb **BRG16=0, BRGH=1**.
A part, al registre TXSTA necessitem activar el **TXEN=1** (Transmit Enable). I assegurar-nos que TX9=0 (dades de 8 bits), tot i que aquest és el valor per defecte al SFR.
També al registre RCSTA necessitem activar el **SPEN=1** (Serial Port Enable). La resta del registre no ens afecta, doncs no estem fent recepció de dades.

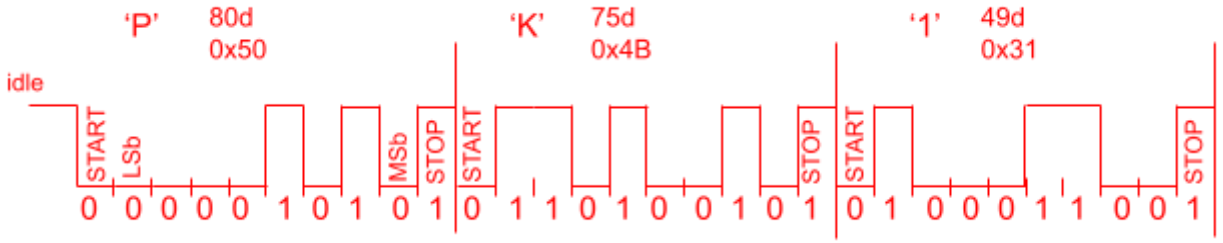
5.2 Dibuixa el cronograma dels bits que surten pel pin TX1 durant l'enviament dels primers tres caràcters de la trama d'**exemple** de l'enunciat. L'estat *idle* treu un '1' lògic. Ajuda't amb la taula ASCII que adjuntem (els números estàn en **base decimal**). (0.5p)

Primers tres caràcters: 'P' 'K' '1'. Per cada caràcter, anem a la taula ASCII, consultem el número que està en decimal com diu l'enunciat, i el convertim a binari per treure'l per la línia TX.
Per exemple, el tercer caràcter: volem enviar un caràcter '1'. Segons la taula ASCII és el valor 49 (decimal). En binari: 00110001. Compte: quan treiem les dades pel pin TX, primer surt el LSbit, i al final el

MSbit.

Recordem que per cada caràcter (byte de dades) transmès per UART, cal posar un START bit al principi, i finalitzar amb 1 STOP bit.

El cronograma quedaria així, amb un temps per cada bit de $t_{bit} = \frac{1}{4800} \cong 208.3\ \mu s$:



5.3 Calcula quants pokemons per segon podríem notificar amb aquesta comunicació sèrie, si enviéssim trames contínuament, sense pausa entre trames. (0.5p)

S'accepta que els càlculs els fem amb el baudrate ideal de 4800, o amb el real obtingut (4807.7).
Tenim un temps per cada bit de $t_{bit} = \frac{1}{4800} \cong 208.3\ \mu s$
Per cada caràcter de la trama, cal enviar 10 bits (1 start + 8 dades + 1 stop).
$t_{char} = 10 \cdot t_{bit} \cong 2083.3\ \mu s$
Per cada pokemon enviem una trama que té 19 caràcters (el '\n' del final és també un caràcter).
$t_{trama} = 19 \cdot t_{char} \cong 39.58\ ms$

Fem la inversa i sabrem el número de pokemons per segon:
$pokemons/s = \frac{1}{t_{trama}} = \frac{1}{39.58ms} \cong 25.26\ pokemons/s$

P6. (1 punt)

Volem enviar les trames de la pregunta P5 usant un bus SPI, configurat a una velocitat de 3 Mb/s.

¿Quant temps trigarem en enviar 1 trama?
¿Amb aquest sistema de comunicacions, es podran detectar errors en la transmissió?

En bus SPI surten pel pin SDO els bits de les dades, sense cap bit extra degut al protocol de transmissió.
Per tant, per enviar 1 trama sencera, caldran 19·8 bits = 152 bits.
La duració d'1 bit ve determinada per la velocitat:
$t_{bit} = \frac{1}{3 \cdot 10^6 bits/s} \cong 0.33\ \mu s$

$t_{trama} = 152\ bits \cdot t_{bit} \cong 50.67\ \mu s$

En una comunicació SPI no hi ha sistema de detecció d'errors en la transmissió.

Computer Interfacing (CI) FIB. Segon parcial **13/01/2023**

Cognoms, Nom _____ DNI _____

Tota resposta sense justificar es considerarà nul·la !

P7. (2 punts)

Observeu aquests dos codis, orientats a saber l'amplada d'un pols que arriba a un *PIN*. Per cada un dels casos teniu la versió en C i ASM per veure que la compilació ha estat òptima.

CODI 1

```
// versió C

        TMR1GE = 0;
        T1CON = 0x13;
        while( PIN == 0);
        T_START = TMR1;
        while( PIN == 1);
        T_END = TMR1;



// versió ASM

        BCF TMR1GE
        MOVLW 13h
        MOVWF T1CON
loop1   BTFSC PIN
        BRA loop1
        MOVFF  T_START_L, TMR1L
        MOVFF  T_START_H, TMR1H

loop2   BTFSS PIN
        BRA loop2
        MOVFF T_END_L, TMR1L
        MOVFF T_END_H, TMR1H
```

CODI 2

```
// versió C

        TMR1GE = 0;
        T1CON = 0x13;
        CCP1CON   = 0x05;
        CCP2CON   = 0x04;
        CCPTMRS0 = 0x00;
        while (CCP2IF==0);
        T_START = CCPR1;
        T_END = CCPR2;

// versió ASM

        BCF TMR1GE
        MOVLW 13h
        MOVWF T1CON
        MOVLW 5
        MOVWF CCP1CON
        MOVLW 4
        MOVWF CCP2CON
        CLRF CCPTMRS0
loop    BTFSC CCP2IF
        BRA loop
        MOVFF  T_START_L, CCPR1L
        MOVFF  T_START_H, CCPR1H
        MOVFF  T_END_L, CCPR2L
        MOVFF  T_END_H, CCPR2H
```

En ambdós casos tenim connectat al xip un oscil·lador de **10 MHz**. A la versió 1, el senyal amb el pols arriba al PIN i a la versió 2 l'hem connectat als pins CCP1 i CCP2. Considereu els pins ben configurats com a entrades. El pols serà un flanc de pujada seguit d'un flanc de baixada.

7.1 En quin dels dos casos (1 o 2) podrem detectar polsos amb més precisió? Per què? (0.5p)

En el cas 2: quan tinguem un pols ascendent la unitat CCP1 guardarà el valor del Timer1 al registre CCPR1 i el mateix pel flanc descendent (CCP2: Timer1 a CCPR2). Això es farà per hardware i quan poguem anirem a recollir les dades (CCPR1 i CCPR2) per copiar-les a les variables. Detectarem que ja estan les dades a punt amb el flag de CCP2IF (això no vol dir que hi hagi cap interrupció involucrada als codis).

En el cas 1 això es faria per software i presenta dos inconvenients:

- des del flanc de pujada (sortida del loop1), triguem uns quants cicles fins arribar a l'espera del flanc de baixada (entrada loop2). Si en aquest temps ja ha arribat el de baixada, no ho podrem detectar.
- en cas de que arribin interrupcions després de sortir dels loops, els timers seguiran corrent i no farem la seva còpia a les variables START i END, portant a errors.

En ambdós casos podem tenir el problema d'overflow del Timer!

7.2 Quina és l'amplada mínima de pols que podrem detectar en el cas millor? (1p)

Amb la configuració donada: Fosc=10 MHz, font del Timer1 a Fosc/4 i PREscaler=2, cada tick de timer serà de: 100ns x 4 x 2 = 800ns.
Com que el que capturem als CCPR són polsos del Timer1, el pols més petit que podem detectar seria un tick de timer (CCPR1 = K, CCPR2 = K+1).
Si heu suposat que Fosc era 8 MHz, el Timer1 faria cicles de 1us.

7.3 Proposa un canvi senzill al codi en C per augmentar aquesta precisió (a una sola línia). (0.5p)

Podríem tenir més precisió augmentant d'alguna manera la freqüència que arriba al Timer1, com que ens diuen fer-ho per codi el millor és canviar el valor al registre de configuració T1CON. Podem posar el Prescaler a 1 (és a dir no fer servir Prescaler) i augmentarem x2 la precisió.
També ens podem plantejar fer servir Fosc en comptes Fosc/4 (acceptat a la correcció), tot i que el manual diu que en modes CCP no és possible.