ÚS DELS TIMERS I INTERRUPCIONS

En aquesta pràctica farem servir dos dels recursos més importants dels que sol disposar qualsevol microcontrolador:

- **Timers (Temporitzadors/Comptadors).**
- **Interrupcions.**

Ja coneixem els perifèrics més senzills que són els ports GPIO (entrades/sortides digitals). Els ***Timers*** (Temporitzadors/Comptadors) són un altre tipus de perifèric. Son comptadors, que compten polsos de rellotge del sistema. Els podem configurar per fixar el límit fins al que volem que comptin, i executar alguna acció quan s'arribi a aquest límit. També poden generar interrupcions cada cop que es compleix un comptatge. El nostre microcontrolador té 4 timers (TAx), amb 5 comptadors (TAx.n) cadascun. Consulteu la documentació ***Technical Reference Manual***, capítol 19, especialment els apartats "19.2.3 Timer Mode Control" y "19.2.6 Timer_A Interrupts".

Les ***interrupcions*** permeten interaccions entre programa i usuari mitjançant els perifèrics (pantalla, teclat, UART, Timers, botons, etc...). En el nostre microcontrolador, podem decidir per a cada pin individualment (per exemple el pin P1.3) i per a cada perifèric (per exemple el Timer TA0.0) si volem activar o no que puguin generar interrupcions.

Quan un perifèric sol·licita (o genera) una interrupció, el processador interromp el programa per atendre-la:

- desa l'estat del programa,
- identifica la interrupció,
- salta a la subrutina (Interrupt Service Routine, ISR) corresponent que hàgim programat,
- l'executa,
- recupera l'estat del programa,
- torna al punt on s'havia interromput y segueix amb el curs normal del programa.

Recordeu que les interrupcions són totalment asíncrones, és a dir que poden produir-se en qualsevol punt del programa a partir de la instrucció amb la que les habilitem. Això també vol dir que per fer servir les interrupcions les hem d'habilitar prèviament a la seva utilització.

Sovint, ens resultarà útil programar una ISR per modificar el valor d'alguna variable global nostra, de manera que al tornar d'executar la ISR, el curs del programa segueixi amb el nou valor.

L'habilitació/inhabilitació de les interrupcions es fa a 3 nivells:

- **Primer nivell**: cada font d'interrupcions té un bit d'habilitació (Enable). Per exemple, pel port GPIO x, tenim el registre PxIE que permet habilitar/inhabilitar cada pin com a font d'interrupció del port. En el cas d'un timer, ho configurarem amb el registre TaxCCTLn (on "x" és el numero del timer, i "n" és la interrupció que volem configurar). Estudieu la funció **init_botons()** proporcionada amb el programa de partida de la pràctica 2 per veure un exemple de com s'han de configurar aquestes interrupcions. Consulteu la documentació ***Technical Reference Manual***, apartat 12.2.7 i 19.2.6.
- **Segon nivell**: a nivell de perifèric, mitjançant l'anomenat "controlador NVIC". Cada perifèric hi té una entrada per inhabilitar/habilitar el conjunt de totes les seves interrupcions. Per defecte, el microcontrolador s'inicialitza amb les interrupcions de tots els perifèrics inhabilitades a nivell

de l'NVIC. Estudieu la funció **init_interrupciones()** proporcionada amb el programa de partida de la pràctica 2 per veure un exemple de com s'ha de configurar aquest controlador. Consulteu la documentació esmentada en els comentaris d'aquesta mateixa funció: Datasheet ("Table 6-39. NVIC Interrupts", apartat "6.7.2 Device-Level User Interrupts", i Technical Reference Manual, apartat "2.4.3 NVIC Registers").

- **Tercer nivell:** a nivell global del microcontrolador. En qualsevol moment podem inhabilitar/habilitar temporalment totes les interrupcions que hàgim activat en els altres dos nivells, amb les funcions "**_disable_interrupt()**" i "**_enable_interrupt()**". Per defecte, el microcontrolador s'inicialitza amb les interrupcions inhabilitades a nivell global. Això ens permet configurar totes les interrupcions que vulguem a nivell individual, sense que se'ns dispari cap abans d'hora, i terminarem habilitant-les globalment quan ho tinguem tot a punt.

Important:

- Per a que funcioni una font d'interrupció, l'hem d'habilitar a tots 3 nivells.
- Cada cop que decidim activar una interrupció, no ens hem d'oblidar d'escriure la seva ISR. En cas contrari, al disparar-se una interrupció sense ISR, el programa saltaria automàticament a una ISR predefinida, **Default_Handler()**, que conté un bucle infinit, on es quedaria "atrapat" i controlat per evitar mals majors.
- A diferencia de les altres funcions "normals", a una ISR, NO la cridem nosaltres, ja que lògicament, no sabem mai quan es produiran les respectives interrupcions. Es el microcontrolador qui s'encarrega de desviar el curs del programa cap a la ISR pertinent.

Els nom de les rutines d'atenció a les interrupcions (ISRs) ens venen predefinits en la taula

```
/* Interrupt vector table.
```

dins l'arxiu

startup_msp432p401r_ccs.c

que se'ns genera automàticament al crear un projecte nou.
Les ISR no reben cap paràmetre ni tornen cap valor:

```
void Nom_predefinit_de_la_ISR(void) {

    //instruccions…

}
```

Recursos del Boosterpack

En aquesta pràctica, a part dels recursos descrits prèviament del nostre microcontrolador, també farem servir recursos nous de la placa del Boosterpack (la placa del robot que està sobre la nostre placa del microcontrolador i que te entre altres coses un joystick i pantalla LCD).

Aquesta placa disposa dels següents recursos connectats al nostre microcontrolador en els següents ports/pins.

Recursos	Px.y
Joystick Dreta	P4.5
Joystick Esquerra	P4.7
Joystick Centre	P4.1
Joystick Amunt	P5.4
Joystick Avall	P5.5
Polsador S1	P5.1
Polsador S2	P3.5
LED_R (vermell)	P2.6
LED_G (verd)	P2.4
LED_B (blau)	P5.6

Taula 1 Assignació de port.pin als recursos del boosterpack

Tasca a realitzar

Per practicar amb tots aquests recursos nous ho farem en dos parts:

1. Generar una base de temps de l'ordre de 0.1 ms (10 kHz) per controlar la lluminositat del LED vermell.
    a. Crear una base de temps d'una mil·lèsima de segon (una freqüència de 1 kHz).
    b. Utilitzarem aquesta base de temps per commutar l'estat del LED vermell cada cert nombre d'interrupcions d'aquesta base de temps de forma asimètrica. Per això, implementeu un comptador que arribi fins a 100 i que llavors doni la volta (i.e. passa a valer 0). A partir d'aquest comptador, el LED s'ha d'encendre sempre que arribeu al valor màxim de 100 i s'ha d'apagar en un valor que sigui variable. Aquesta variable controlarà la lluminositat (relació estat on/off del LED).
    c. Implementeu que aquest valor sigui variable. Aquest valor és modificarà de la següent manera, sempre respectant els límits anterior de màxim 100 i mínim 0, ambdós inclosos:
        i. Joystick amunt: increment d'aquest valor en una quantitat *step*
        ii. Joystick avall: decrement d'aquest valor en una quantitat *step*
        iii. Joystick esquerra: la quantitat *step* és veu decrementada en 5.
        iv. Joystic dreta: la quantitat *step* és veu incrementada en 5.
    d. Un cop heu comprovat el correcte funcionament, baixeu ara les interrupcions a 0.1 ms (freqüència de 10 kHz).

2. Generar la següent seqüència de colors amb el RGB, controlat amb una base de temps d'un segon i guardada en un array d'estructures (o una única estructura).
    a. Crear una base de temps d'un segon (freqüència de 1 Hz).
    b. Els LEDs RGB han de ser configurats per tal de recórrer el següent patró amb el temps definit sobre la base de temps anterior.
    c. Heu d'utilitzar una estructura (o un array d'estructures) per tal de configurar el patró a seguir. L'objectiu és tenir el codi estructurat d'una forma que en el futur sigui fàcil modificar el patró, incloent el nombre d'etapes d'aquest.

Color RGB	**Vermell**	**Groc**	**Verd**	**Blau**	**Blanc**
Temps	2	1	3	2	1

Taula 2 Patró dels LEDs RGB a generar

Per aquesta pràctica utilitzarem la funció "init_ucs_24MHz()", disponible a la llibreria "lib_PAE2.h".
Amb aquesta funció, l'ACLK ens queda a 37268 Hz, i l'SMCLK funciona a 24MHz. Penseu que la configuració de la UCS repercuteix en el funcionament de tot el sistema, per tant es recomana que aquesta configuració es faci al començament de l'execució del programa (just després d'aturar el "Watchdog Timer" tal com hem anat fent en les pràctiques anteriors).

Pels timers, podeu usar qualsevol d'aquestes dos fonts de rellotge. Llegiu bé el codi de partida i trieu la més adient en cada cas. Justifiqueu la vostra decisió en el informe.

Tasca Opcional

Si disposeu de més temps podeu aprofitar per tal de familiaritzar-vos amb el robot i específicament amb la pantalla LCD de la qual disposem. Analitzeu el codi de la tasca 1 "Codi pràctica 1 – Robot". Veureu com aquest permet usar la pantalla. Per fer això, primer cal inicialitzar-la usant la funció *halLcdInit()* (que per altre banda, com ja s'ha comentat abans, requereix també de la configuració dels clocks i cal haver executat la funció *init_ucs_24MHz()* prèviament). Doneu una ullada al header de la llibreria PAE2 (el fitxer *lib_PAE2.h*) per veure totes les funcions de les quals disposem. La que més fareu servir, exceptuant les anteriorment mencionades, probablement sigui *halLcdPrintLine()*. Aquesta funció ens permet fer *prints* en la pantalla LCD.

Tingueu en compte que en el món de les arquitectures encastades aquests tipus de funcionalitats són força cares en termes computacionals. Hi ha fins i tot microcontroladors que o directament no tenen la funció *printf* disponible o només de forma reduïda, donada la seva complexitat i cost computacional per un microcontrolador. Per tant, eviteu abusar d'imprimir coses en pantalla i penseu que pot alterar de forma molt significativa la latència/temporització del vostre codi. En aquest sentit, NO CRIDEU MAI la funció per imprimir al LCD des de dintre una interrupció.