

CONFIGURACIÓ DE PORTS

En aquesta pràctica anirem introduint com es controlen els ports d'Entrada/Sortida de propòsit general, o **GPIOs (General Purpose Digital Input/Outputs)** del microcontrolador que fem servir.

A més, farem us de diverses instruccions de programació que ens permeten controlar el flux del programa, per fer salts condicionals i bucles al C:

- `if`
- `for`
- `do ... while`
- `switch ... case`

El primer que hem de tenir en compte és quins recursos farem servir del nostre sistema i a quins ports del microcontrolador estan connectats.

En aquesta pràctica farem servir el dos pulsadors S1 i S2, el LED vermell separat (LED1) i els tres LEDs RGB (identificats com a LED2). Els ports GPIOs connectats als recursos esmentats són els que indiquem a la taula 1.

Recurs	Port.pin	Recurs	Port.pin
S1	P1.4	LED2_RED	P2.0
S2	P1.1	LED2_GREEN	P2.1
LED1	P1.0	LED2_BLUE	P2.2

TAULA 1 CONNEXIÓ DELS RECURSOS AL MCU

Hem de tenir clar que els pulsadors són dispositius d'entrada, i per tant els pins als que estan connectats han d'estar programats com a entrades. En canvi, els LEDs són dispositius de sortida i per tant els pins associats els hem de programar com a sortides.

Aquesta és pràcticament la primera tasca que hem de dur a terme quan estem fent un programa per a un microcontrolador: **configurar els ports** de la manera en que volem que treballin. És el que anomenarem **Inicialització dels ports**. I per fer-ho hem de consultar el *Technical Reference Manual* del nostre microcontrolador, en aquest cas el **MSP432P401R**. Aquesta configuració la trobem al **capítol 12** d'aquest manual, capítol que recomanem molt que el llegiu, tot i que ara en farem un resum.

La forma de configurar el funcionament dels ports és escrivint el contingut d'uns registres específics del microcontrolador. Cada port sol tenir associats una sèrie de registres per controlar el seu funcionament i aquests registres els hem d'estudiar molt detingudament per tal d'aconseguir el resultat adient a les nostres aplicacions. Pot passar amb molta facilitat que per programar incorrectament només un d'aquests registres (ficar 1 on hauria d'anar 0 o viceversa), la nostra aplicació no faci el que volem, o faci just el contrari. És per tant un punt on haurem de tenir molta cura, ja que **una mala configuració pot provocar un curt-circuit en el pin**, portant a un mal funcionament del dispositiu o fins i tot provocant danys en aquest.

Els pins GPIO estan agrupats en ports, de 8 pins cadascun, per exemple:

- P1.0...P1.7: pins 0 a pin 7 del port 1
- Px.0...Px.7: pins 0 a pin 7 del port x

Així mateix, els diferents registres de configuració i control d'un determinat port són de 8 bits, 1 bit per cada pin. És a dir, el bit 0 d'un registre sempre fa referència al pin 0 del port que configura/controla, el bit 1 al pin 1, ... i el bit 7 al pin 7.

Hem de tenir present que:

- Un mateix pin del microcontrolador pot tenir diferents funcions. Per exemple, pot funcionar com un senyal digital d'entrada o sortida, o com una entrada analògica, o com una sortida de polsos... Això dependrà de cada pin específic, ho hem de mirar al document "*MSP432xx datasheet*". En un moment determinat només pot tenir una d'aquestes funcions i nosaltres hem de configurar quina volem.
- Si un pin específic el configurem com a Entrada/Sortida Digital, hem d'especificar si és Entrada o Sortida.
- Si un pin pot treballar amb interrupcions, i ho volem així, també ho hem de configurar.

A més, aquestes configuracions les podem anar canviant durant l'execució del programa segons ens interressi. Això vol dir que un mateix pin del microcontrolador pot funcionar d'una manera determinada en un moment i després d'una altra.

Com hem dit abans, la configuració dels ports digitals del nostre microcontrolador la trobem al **capítol 12** del *Technical Reference Manual*, i per a aquesta pràctica ens interessarà estudiar sobretot les **pàgines 678 i 680**, on ens expliquen com funcionen i com es configuren, mentre que les possibles funcions de cada pin estan al *datasheet*, **pàgines 7** ("Pin diagram", figure 4.1) i **16 a 25** ("Signal descriptions", table 4.2).

Analitzem una mica els **registres que hem de configurar**:

- **PxSELO, PxSEL1**: Amb aquests registres decidim sobre cada pin d'un port (el port que posem com a "x") si volem que funcioni com a entrada/sortida digital o com a una altra funció alternativa que tingui assignada (timer, entrada analògica...). Si un *bit n* està a 0 en tots dos registres, vol dir que aquest *pin n* del *port x* funcionarà com a GPIO, mentre que si el *bit n* d'un dels registres està a 1 vol dir que funcionarà com a una funció alternativa.
- **PxDIR**: Aquest registre indica quins pins d'un port, dels que han estat configurat com a I/O digital, seran d'entrada i quins seran de sortida. Un 0 a un bit del registre implica que el pin corresponent del port és una entrada, mentre que un 1 significa que serà una sortida.
Per a informació més detallada sobre la configuració de les funcions alternatives, segons les diferents combinacions de *PxSELO*, *PxSEL1*, i *PxDIR*, podeu consultar el capítol 6.10 del *datasheet* del nostre microcontrolador.
- **PxREN**: Aquest registre serveix per indicar que passa si un pin és una entrada digital i no posem res a l'entrada (entrada flotant o d'alta impedància). Si no ho configuréssim, l'estat del pin seria indeterminat! Amb aquest registre podem decidir que en aquest cas el valor d'entrada sigui un '1' o un '0' (veure *Technical Reference Manual*, configuració de les resistències de pullup/pulldown).
NOTA: Els polsadors de la nostre placa de desenvolupament no tenen cap resistència pull-up/pull-down a l'entrada i el costat del polsador que no està connectat al microcontrolador està a GND (i.e. '0').

Recordeu que al nom de tots aquests registres la "x" s'ha de substituir pel número de port que volem configurar.

EXEMPLE:

Suposem que volem fer servir els pins 0 al 5 del port 1 com a I/O digitals i els pins 6 i 7 com a primera funció alternativa. Hauríem d'omplir el registre P1SEL0 amb "11000000", i el P1SEL1 amb tot de 0s. Això en "C" ho escriurem: **P1SEL0 = 0xC0; P1SEL1 = 0;**

A més volem que dels pins que hem definit com a GPIOs, els 0, 1 i 2 siguin entrades i els 3, 4 i 5 sortides. Hem d'escriure a P1DIR el següent "00111000" i en "C" serà: **P1DIR = 0x38;** (de fet els dos "0" de més pes podrien ser "1" perquè abans hem definit aquests pins com funció alternativa no com a I/O digital).

Quan escrivim als registres de configuració, hem de plantejar-nos moltes vegades el fet d'escriure només alguns bits del registre sense modificar els altres bits. Per fer això farem servir els **operadors lògics "OR" i "AND"**, que en "C" s'escriuen amb els símbols "**|**" i "**&**" respectivament, juntament amb màscares convenientment calculades.

Per exemple, si posem **P1DIR |= 0x03;** significa que estem posant a "1" els bits 0 i 1 (màscara 0x03 en binari és 00000011) del registre P1DIR sense modificar els altres bits d'aquest registre de configuració, ja que el OR només modificarà el bits que posem a 1 a la nostra màscara.

Si fem **P1DIR &= 0xF7;** vol dir que posem a "0" el bit 3 del registre P1DIR i deixem sense modificar la resta de bits. Això és perquè la funció AND només modifica els bits que nosaltres posem a "0" a la nostra màscara.

I com podem accedir als ports que programem com a I/O digitals?

Ho fem mitjançant dos registres addicionals:

- **PxIN:** Aquest registre serveix per llegir el que entrem pels pins del port "x" que haguem configurat com a entrada. En aquest registre, mai escriurem, és de només lectura. Tot i que aquest registre recull l'estat dels 8 pins del port, només tindran sentit aquells que hàgim configurat com a entrades.
- **PxOUT:** La funció és similar, però ara serà per escriure un estat a les sortides. En el cas de les entrades, no tindria sentit intentar escriure un estat de sortida, així que el registre PxOUT té una funció addicional de configuració (veure *Technical Reference Manual*, configuració de les resistències de pullup/pulldown). Per tant, al manipular pins de sortida d'un determinat port, haurem de tenir cura de no desconfigurar els pins del mateix port que hàgim configurat prèviament com a pins d'entrada, fent ús dels operadors lògics i màscares adients esmentats abans.

CONTINUEM AMB L'EXEMPLE:

Suposem que hem fet la configuració anterior i volem treure tot "1" pels 3 pins de sortida configurats i volem llegir el que hi ha als 3 pins d'entrada a la variable Estat_Entrades, podem fer:

P1OUT |= 0x38; //posem a 1 els pins 3, 4, i 5 del port 1, sense tocar els altres pins.

Estat_Entrades = P1IN & 0x07; //La màscara ens permet llegir l'estat d'entrada dels pins 0, 1, i 2, posant els bits corresponent als altres pins a 0 en la nostra variable resultant Estat_Entrades.

EXERCICI PROPOSAT

Amb aquestes nocions podem configurar ja els registres de control del microcontrolador pels recursos que volem fer servir de la placa (descrits abans, polsadors i LEDs).

Ho trobareu parcialment fet al programa "Practica_02_PAE.c" a la funció "init_botons()" pels polsadors i el LED vermell solitari de la placa de desenvolupament. És molt recomanable que ho feu vosaltres, o com a mínim que ho analitzeu i entengueu molt bé. Fixeu vos en les diferents maneres de calcular les màscares, segons els operadors lògics utilitzats.

Preneu com a base aquest programa " Practica_02_PAE.c " per posar en pràctica les instruccions "do ... while", "if", "for" i "switch ... case".

No us oblideu d'incloure les llibreries estàndard <stdio.h> i <stdint.h>.

Recordeu també que la primera instrucció dels nostres programes sempre haurà de ser:

```
WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
```

FUNCIONALITAT

El vostre programa ha de fer el següent:

- Si polsem S1, un nombre imparell de vegades, s'han d'encendre els 3 LEDs RGB. Aquest és el estat 1.
- Si polsem S1, un nombre parell de vegades, s'ha d'invertir l'estat dels 3 LEDs RGB. Aquest és el estat 2.
- Si polsem S2, s'han de desplaçar els LEDs encesos de tal forma que el nou estat del LED RGB verd correspon al anterior del vermell (R -> G) i el nou estat del LED RGB blau correspon al anterior del verd (G -> B). El nou valor del LED RGB vermell passarà a ser 0. Aquest és el estat 3. Pulsacions consecutives del S2 han de produir aquest efecte.

Independentment d'això, el LED simple vermell ha d'anar fent pampallugues a un ritme fix.

US DEL SWITCH ... CASE I IF

Farem servir la instrucció "switch ... case" per detectar quin polsador (S1 o S2) hem pitjat. Farem la següent seqüència:

- Si polsem S1, un nombre imparell de vegades la variable `estado` en aquest cas ha de prendre el valor 1.
- Si polsem S1, un nombre parell de vegades, la variable `estado` en aquest cas ha de prendre el valor 2.
- Si polsem S2, la variable `estado` en aquest cas ha de prendre el valor 3.

Estat actual	Pulsador	Estado futur
1	S1	2
2	S1	1
X	S2	3

Recomenació: Per verificar que els polsadors estan correctament configurats, és recomana que una de les primeres proves sigui posar un breakpoint dintre de la funció `PORT1_IRQHandler()` i verificar que després de prémer cadascun dels botons el codi arriba al breakpoint.

DINTRE DE LA FUNCIÓ MAIN()

Les interrupcions sempre han de ser curtes. En aquest sentit serà en la funció main on hem d'implementar la següent funcionalitat depenent del valor de la variable global `estado`.

- Si `estado` val 1, s'han d'encendre els 3 LEDs RGB.
- Si `estado` val 2, s'han d'invertir l'estat dels 3 LEDs RGB.
- Si `estado` val 3, s'han de desplaçar els LEDs encesos de tal forma que el nou estat del LED RGB verd correspon al anterior del vermell (R -> G) i el nou estat del LED RGB blau correspon al anterior del verd (G -> B). El nou valor del LED RGB vermell passarà a ser 0. **No heu de modificar cap altre pin del port excepte els dels LEDs RGB.**

US DEL FOR

En el nostre programa volem fer una funció que permeti generar un temps d'espera. Denominarem a aquesta funció "`delay_t()`" i ha de tenir un paràmetre d'entrada que serveixi per generar retards de diferents durades. Ho podem fer de moltes maneres, però en aquest cas ho farem amb un *bucle* "for".

Al programa veureu que està definida la funció sense implementar: `void delay_t (uint32_t temps)`, on "temps" és el paràmetre que enviarem per determinar la durada del temps d'espera quan cridem a la funció `delay_t()`.

Nota: El ús de la funció `__delay_cycles()` és només temporal fins que feu la implementació definitiva. Un cop realitzada, borreu o comenteu la seva crida.

Nota important: veureu que a la configuració dels polsadors també es fa la configuració d'algunes interrupcions. Això és necessari pel funcionament del nostre sistema però explicarem el seu funcionament a una altra pràctica. Per ara només necessitareu saber que farem servir els registres dels vectors d'interrupcions, "**PxIV**", per saber quin polsador hem polsat.

Per saber quin polsador s'està fent actuar, **heu d'analitzar els valors dels registres PxIV dintre de les rutines d'atenció a les interrupcions (ISR) corresponents a cada port x "`Portx_IRQHandler()`".** A aquestes funcions, s'hi entra "automàticament" cada cop que polsem S1 o S2 ja que hem configurat les interrupcions associades a aquests ports. Consulteu el **capítol 12, p. 695, del Technical Reference Manual** per determinar a quins valors d'aquests registres correspon cada interrupció (recordeu que teniu la descripció de quin és el pin de cada port connectat a cada dispositiu del nostre robot a la taula 1).

Teniu en compte també que heu d'actualitzar els diferents estats a les funcions d'atenció a les interrupcions (ISR) dels polsadors mitjançant la variable global "`estado`", i al bucle principal de la funció `main()` feu servir aquesta mateixa variable per fer les operacions desitjades sobre els LEDs en cada moment.