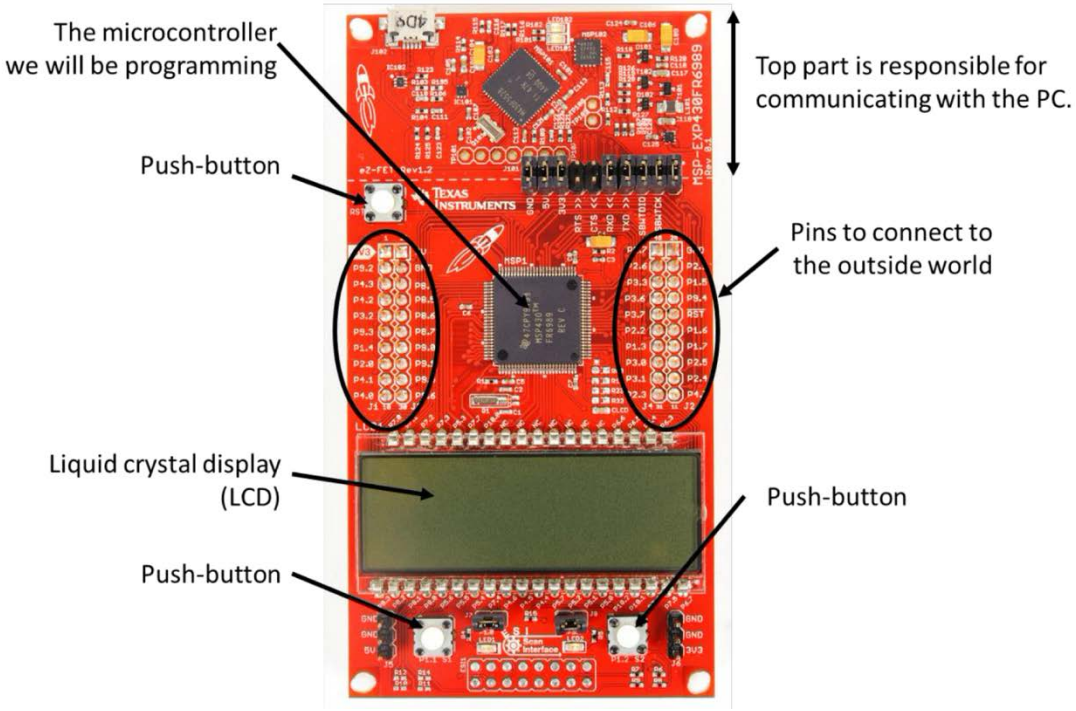
PRÁCTICA2: PUERTOS DE E/S DEL MSP340

El objetivo de esta práctica es aprender a configurar y manejar algunos de los dispositivos de E/S de la placa MSP430FR6989 mostrada a continuación.



La comunicación con los distintos dispositivos de la placa se hace a través de 12 puertos de E/S (de P1 a P11 y PJ). La mayoría de estos puertos contienen 8 líneas de E/S. Cada una de las líneas puede configurarse individualmente como entrada o como salida, y por tanto se pueden leer o escribir. Además cada línea puede configurarse individualmente con una resistencia de pullup o pulldown. Como funcionalidad básica, estas resistencias establecen un estado lógico en un pin cuando se encuentra en estado reposo (pull up establece un estado high y pull down un estado low).

Los puertos pueden accederse a nivel de byte o combinarse varios puertos para acceder a ellos a nivel de palabra (16 bits). Los puertos P1 y P2, P3 y P4, P5 y P6, P7 y P8, P9 y P10, y P11 y PJ, están asociados y se puede acceder a ellos con los nombres PA, PB, PC, PD, PE y PF, respectivamente. El acceso a estos puertos se hace a nivel de palabras de 16 bits.

Los pines de GPIO (General Purpose Input Output) se indican como PX.Y, donde X representa el número de puerto e Y representa el número de pin.

Pines de GPIO:
• LED1 (red) = P1.0 • LED2 (green) = P9.7 • Switch1 = P1.1 • Switch2 = P1.2 • Timer UART Transmit = P3.4 • Timer UART Receive = P3.5

Registros asociados a los puertos

Registros de entrada (PxIN)

Cada bit de los registros PxIN refleja el valor de la señal de entrada en el correspondiente pin de E/S cuando éste está configurado como función de E/S. Se trata de registros de sólo lectura.

Bit = 0 Entrada a nivel bajo
Bit = 1 Entrada a nivel alto

Registros de salida (PxOUT)

Cada bit de los registros PxOUT representa el valor de salida a través del pin de E/S correspondiente, cuando éste está configurado como función de E/S.

Bit = 0 Salida a nivel bajo
Bit = 1 Salida a nivel alto

Si el pin está configurado como función de E/S

Bit = 0 Resistencia Pulled Down
Bit = 1 Resistencia Pulled Up

Registros de dirección (PxDIR)

Cada bit de los registros PxDIR selecciona la dirección del correspondiente pin de E/S, independientemente de la funcionalidad seleccionada para el pin.

Bit = 0 Entrada
Bit = 1 Salida

Registros de las resistencias de Pullup o PullDown (PxREN)

Cada bit de los registros PxREN habilita o deshabilita las resistencias de pullup o pulldown para el correspondiente pin de E/S.

Bit = 0 Resistencia de pullup o pulldown deshabilitada
Bit = 1 Resistencia de pullup o pulldown habilitada

PxDIR	PxREN	PxOUT	Configuración de E/S
0	0	x	Entrada
0	1	0	Entrada con resistencia pulldown
0	1	1	Entrada con resistencia pullup
1	x	x	Salida

Registros de función (PxSEL0, PxSEL1)

Los pines de los puertos se encuentran a menudo multiplexados con otras funciones de periféricos. Cada pin usa 2 bits para seleccionar la funcionalidad del pin (puerto de E/S o alguna de las tres funciones de periféricos).

PxSEL1	PxSEL0	Función de E/S
0	0	E/S de propósito general
0	1	Función del módulo primario
1	0	Función del módulo secundario
1	1	Función del módulo terciario

Acceso y modificación de los bits de los registros

El manejo de los bits de los registros asociados a los puertos de E/S se realiza mediante máscaras de bits y operaciones lógicas.

- Para poner a 1 un determinado bit de un registro sin modificar el resto de bits se realiza la operación OR con un 1 en el bit correspondiente y el resto de bits a 0.

  Ejemplo: Poner a 1 el bit 3 del registro P1OUT:
  P1OUT = P1OUT | 0x08 (o bien P1OUT |= 0x08)

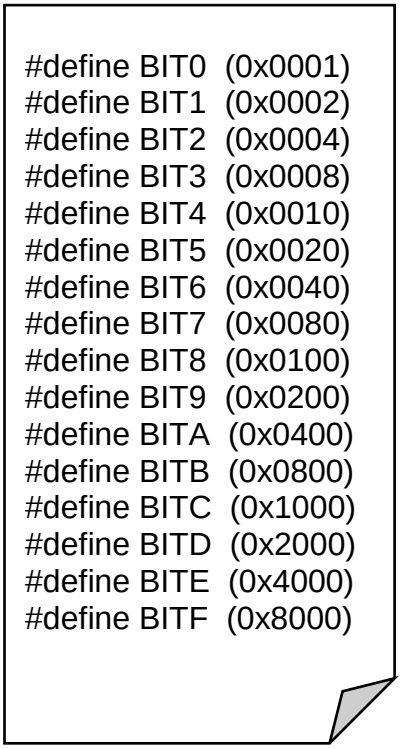
- Para poner a 0 un determinado bit de un registro sin modificar el resto de bits se realiza la operación AND con un 0 en el bit correspondiente y el resto de bits a 1.

  Ejemplo: Poner a 0 el bit 3 del registro P1OUT:
  P1OUT = P1OUT & 0xF7 (o bien P1OUT &= 0xF7)

- Para conmutar el valor de un bit de un registro sin modificar el resto de bits se realiza la operación XOR con un 1 en el bit correspondiente y el resto de bits a 0.

  Ejemplo: Conmutar el bit 3 del registro P1OUT:
  P1OUT = P1OUT ^ 0x08 (o bien P1OUT ^= 0x08)

Como ayuda en esta tarea, en el fichero msp430.h se encuentran las siguientes declaraciones que pueden usarse como máscaras de bits.



```
#define BIT0  (0x0001)
#define BIT1  (0x0002)
#define BIT2  (0x0004)
#define BIT3  (0x0008)
#define BIT4  (0x0010)
#define BIT5  (0x0020)
#define BIT6  (0x0040)
#define BIT7  (0x0080)
#define BIT8  (0x0100)
#define BIT9  (0x0200)
#define BITA  (0x0400)
#define BITB  (0x0800)
#define BITC  (0x1000)
#define BITD  (0x2000)
#define BITE  (0x4000)
#define BITF  (0x8000)
```

Por tanto las asignaciones anteriores podrían hacerse:

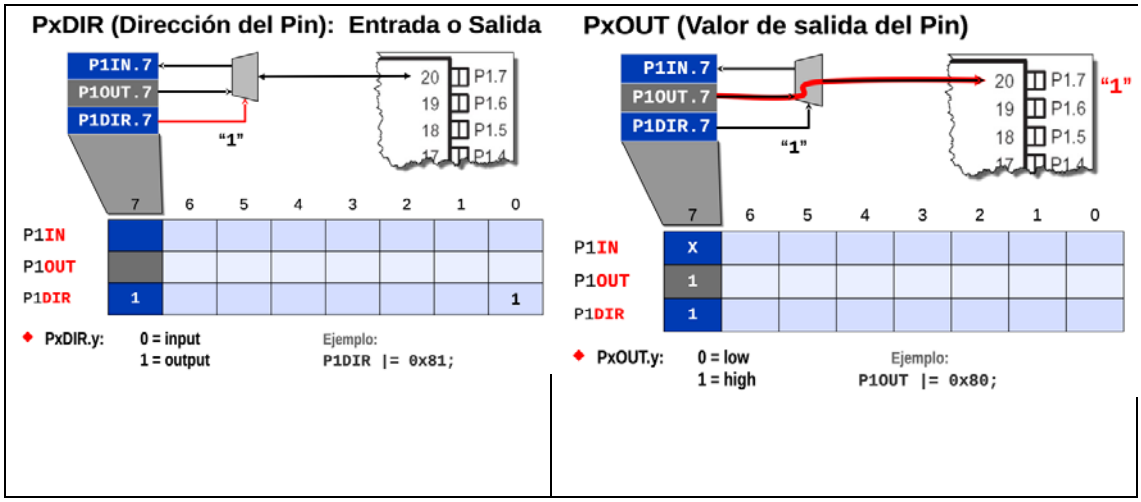
- Poner a 1 el bit 3 del registro P1OUT:
  P1OUT = P1OUT | BIT3 (o bien P1OUT |= 0xBIT3)

- Poner a 0 el bit 3 del registro P1OUT:
  P1OUT = P1OUT & ~BIT3 (o bien P1OUT &= ~BIT3)

- Conmutar el bit 3 del registro P1OUT:
  P1OUT = P1OUT ^ BIT3 (o bien P1OUT ^= BIT3)

Ejemplos de uso de los dispositivos de GPIO

Para utilizar los dispositivos de GPIO deben primero configurarse como entradas o salidas, escribiendo en el bit adecuado del registro de dirección correspondiente. El siguiente ejemplo configura los pines P1.7 y P1.0 como salidas, y pone a 1 la salida del pin P1.7.



En el siguiente ejemplo demo.c para la placa msp430g2553, se configuran los leds como salidas (P1.0 y P1.6) y a continuación se conmuta cada cierto tiempo el valor de salida de ambos pines, consiguiendo así el parpadeo de los leds.

```
#include <msp430g2553.h>
void main(void) {
  WDTCTL = WDTPW + WDTHOLD;  // Stop watchdog timer
  P1DIR |= 0x41;  // set P1.0 & 6 to outputs
                  //(red & green LEDs)
  for(;;) {
     volatile unsigned int i;
     P1OUT ^= 0x41; // Toggle P1.0 & 6 using XOR
     i = 50000; // Delay
     do (i--);
     while (i != 0);
  }
}
```

Nota: Para conmutar los leds rojo y verde de la placa usada en las prácticas es necesario configurar las terminales de los puertos 1.0 y 9.7 como salidas, y luego intercambiar sus estados lógicos.

Ejercicios para placa de desarrollo MSP430FR6989:

1. Generar un programa HOLA MUNDO en C embebido comprobando que el entorno de desarrollo está listo para trabajar.

2. Generar un programa que configure el led rojo (P1.0) como salida y lo haga parpadear con un cierto retardo (30.000 ciclos), implementado mediante un bucle. Probar el mismo ejercicio variando la frecuencia de parpadeo.
   Estudiar función `__delay_cycles(300000);`

3. Generar el programa anterior configurando el led verde (P9.7) como salida y haciéndolo parpadear con un delay de 30.000 ciclos. Probar el mismo ejercicio variando la frecuencia de parpadeo.

4. Generar un programa reuniendo todo lo anterior que haga parpadear ambos leds al mismo tiempo.

5. Implementar el siguiente código describiendo su funcionamiento y explicando las partes fundamentales del código.

```
#include <driverlib.h>

int main(void) {

    volatile uint32_t;
    int i;

    // Stop watchdog timer
    WDT_A_hold(WDT_A_BASE);

    // Set P1.0 to output direction
    GPIO_setAsOutputPin(
        GPIO_PORT_P1,
        GPIO_PIN0
        );

    // Disable the GPIO power-on default high-impedance mode
    // to activate previously configured port settings
    PMM_unlockLPM5();

    while(1)
    {
        // Toggle P1.0 output
        GPIO_toggleOutputOnPin(
            GPIO_PORT_P1,
                    GPIO_PIN0
                    );

        // Delay
        for(i=10000; i>0; i--);
    }
}
```

```
void GPIO_setAsOutputPin(uint8_t selectedPort,
                         uint16_t
selectedPins) {
    uint16_t baseAddress =
GPIO_PORT_TO_BASE[selectedPort];

    #ifndef NDEBUG
    if(baseAddress == 0xFFFF)
    {
        Return;
    }
    #endif

    // Shift by 8 if port is even (upper 8-
bits)
    if((selectedPort & 1) ^ 1)
    {
        selectedPins <<= 8;
    }

    HWREG16(baseAddress + OFS_PASEL0) &=
~selectedPins;
    HWREG16(baseAddress + OFS_PASEL1) &=
~selectedPins;
    HWREG16(baseAddress + OFS_PADIR) |=
selectedPins;

    Return;
}
```

```
void GPIO_toggleOutputOnPin(uint8_t
selectedPort,
                            uint16_t
selectedPins) {
    uint16_t baseAddress =
GPIO_PORT_TO_BASE[selectedPort];

    #ifndef NDEBUG
    if(baseAddress == 0xFFFF)
    {
        returný
    }
    #endif

    // Shift by 8 if port is even (upper 8-
bits)
    if((selectedPort & 1) ^ 1)
    {
        selectedPins <<= 8;
    }

    HWREG16(baseAddress + OFS_PAOUT) ^=
selectedPins;

Return;

}
```