Labs para el workshop de programación en C sobre MSP430 (SASE 2012)

Lab. 1: uso básico de puertos de E/S

```
/*****************************************************
 * Lab01: uso básico de puertos de E/S.
 *        Hace parpadear el led rojo una vez por segundo.
 * Micro: MSP430G2211
 *****************************************************/

#include <msp430.h>

void main(void)
{
	unsigned int i;

	WDTCTL = WDTPW + WDTHOLD;  // Detener el watchdog

	// Luego del reset, las señales de reloj MCLK y SMCLK se obtienen
	// del DCO, con una frecuencia aproximada de 1.1 MHz

	// Configura el Puerto P1 como E/S (con P1.0 maneja el led rojo)

	P1SEL = 0x00;    // P1 completo como E/S
	P1DIR = 0x01;    // P1.0 salida; el resto entrada
	P1OUT = 0x00;    // Apaga el led controlado por P1.0

	while(1)
	{
		for(i=0; i<45833; ++i);   // Retardo de aprox. 0.25 seg

		for(i=0; i<45833; ++i);   // Retardo de aprox. 0.25 seg

		P1OUT ^= 0x01;            // Cada 0.5 seg conmuta el estado del led
		                          // controlado por P1.0
	}
}
```

Luego de la compilación aparecen las siguientes advertencias, a las que por ahora no le prestaremos atención, sino que las trataremos más adelante:

```
remark #10371-D: (ULP 1.1) Detected no uses of low power mode state changes using LPMx or
_bis_SR_register() or __low_power_mode_x() in this project.

remark #10372-D: (ULP 4.1) Detected uninitialized Port 2 in this project. Recommend initializing all
unused ports to eliminate wasted current consumption on unused pins.

"../lnk_msp430g2211.cmd", line 79: warning #10374-D: Interrupt vector "PORT1" does not have an
interrupt handler routine.
"../lnk_msp430g2211.cmd", line 80: warning #10374-D: Interrupt vector "PORT2" does not have an
interrupt handler routine.
"../lnk_msp430g2211.cmd", line 85: warning #10374-D: Interrupt vector "TIMERA1" does not have an
interrupt handler routine.
"../lnk_msp430g2211.cmd", line 86: warning #10374-D: Interrupt vector "TIMERA0" does not have an
interrupt handler routine.
"../lnk_msp430g2211.cmd", line 87: warning #10374-D: Interrupt vector "WDT" does not have an interrupt
handler routine.
"../lnk_msp430g2211.cmd", line 88: warning #10374-D: Interrupt vector "COMPARATORA" does not have an
interrupt handler routine.
"../lnk_msp430g2211.cmd", line 91: warning #10374-D: Interrupt vector "NMI" does not have an interrupt
handler routine.
```

Lab. 2: Analizando detalles de programación del programa del Lab. 1

En el código del Lab. 1 se usan algunas variables que no se han declarado en el programa, sino en un archivo de cabecera incluido a partir "msp430.h". Estas variables hacen referencia a ciertos registros del micro:

WDTCTL
WDTPW
WDTHOLD
P1SEL
P1DIR
P1OUT

A modo de ejemplo vamos a tomar a algunas de ellas y las vamos a definir con otros nombres, para entender como se puede usar una variable para acceder a un registro mapeado en memoria.

En la hoja de datos del MSP430G2211, en la tabla de periféricos con acceso por bytes, se observa que los registros asociados al puerto P1 están mapeados en las siguientes direcciones de memoria:

Port P1	Port P1 resistor enable	P1REN	027h
	Port P1 selection	P1SEL	026h
	Port P1 interrupt enable	P1IE	025h
	Port P1 interrupt edge select	P1IES	024h
	Port P1 interrupt flag	P1IFG	023h
	Port P1 direction	P1DIR	022h
	Port P1 output	P1OUT	021h
	Port P1 input	P1IN	020h

En el programa siguiente se efectuan definiciones análogas a las de P1SEL, P1DIR y P1OUT:

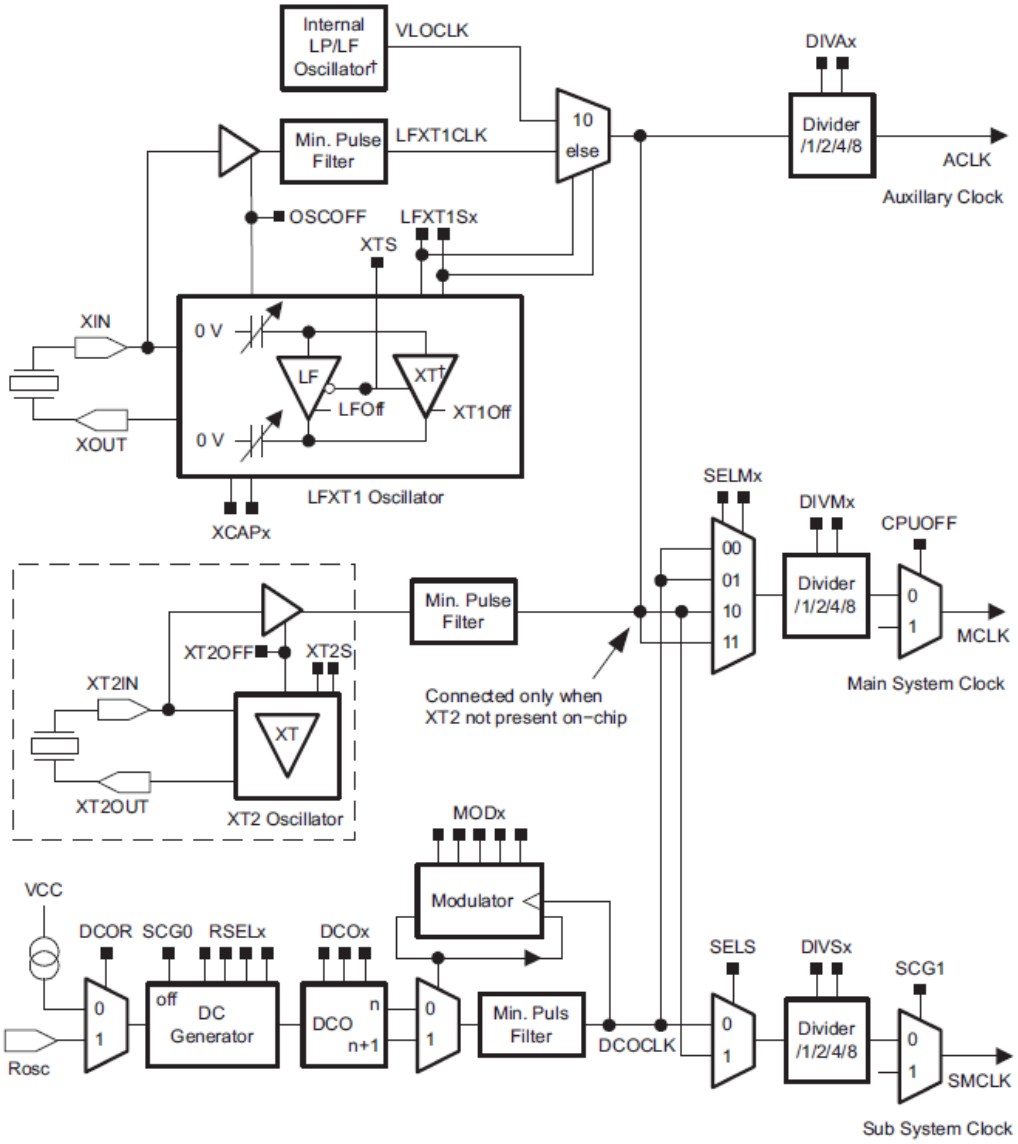
```
/*****************************************************************************
 * Lab02: definición de nombres para acceder a registros mapeados en memoria
 * Micro: MSP430G2211
 *****************************************************************************/
#include <msp430.h>

// Declaraciones para sustituir a P1SEL, P1DIR y P1OUT
#define P1_SEL (*(unsigned char *)0x0026)
#define P1_DIR (*(unsigned char *)0x0022)
#define P1_OUT (*(unsigned char *)0x0021)

void main(void)
{      unsigned int i;

       WDTCTL = WDTPW + WDTHOLD;  // Detener el watchdog

       // Configura el Puerto P1 como E/S (con P1.0 maneja el led rojo)

       P1_SEL = 0x00;     // P1 completo como E/S
       P1_DIR = 0x01;     // P1.0 salida; el resto entrada
       P1_OUT = 0x00;     // Apaga el led controlado por P1.0

       while(1)
       {
              for(i=0; i<45833; ++i);   // Retardo de aprox. 0.25 seg
              for(i=0; i<45833; ++i);   // Retardo de aprox. 0.25 seg
              P1_OUT ^= 0x01;           // Cada 0.5 seg conmuta el estado del led
                                        // controlado por P1.0
       }
}
```

Lab. 3: Configurando las distintas señales de reloj del micro.

El MSP430 presenta distintas opciones para configurar las tres señales de reloj:

- MCLK: reloj principal, que gobierna la ejecución de instrucciones.
- SMCLK: reloj secundario, que puede ser configurado para distintos periféricos
- ACLK: reloj auxiliar, que también puede ser configurado para distintos periféricos

La siguente figura, extraída de la guía de usuario de la familia del MSP430G2211, muestra las distintas opciones de configuración. Se pueden observar las señales (por ej.: VLOCLK, DCOCLK, etc.) y los parámetros de configuración (por ej.: DIVAx, SELMx, etc.).



Luego del reset MCLK se obtiene del DCO, cuya configuración es RSEL=7 y DCO=3, lo que debería dar una frecuencia aproximada de 1.1 MHz (en realidad puede variar entre 0.8 y 1.5 MHz).
En el presente Lab vamos a configurar MCLK a una frecuencia aproximada de 7.8 MHz y SMCLK a 1/8 de la anterior, a partir del DCO. Por otra parte, ACLK se obtendrá del oscilador interno de baja frecuencia y baja potencia (VLOCLK: ~ 12 KHz). Para ello se usarán los siguientes parámetros de configuración:

- LFXT1S2 (10_b)
- DIVA0
- DCO13
- RSEL15
- MOD0
- SELM3
- DIVM0
- SELS0
- DIVS0

```
/*******************************************************************************
 * Lab03: configuración de las señales de reloj MCLK, SMCLK utilizando
 *        el DCO, y ACLK usando el oscilador local de baja frecuencia (VLO)
 * Micro: MSP430G2211
 ******************************************************************************/

#include <msp430.h>

void main(void)
{
      unsigned int i, j;

      WDTCTL = WDTPW + WDTHOLD;  // Detener el watchdog

      // Configura el DCO para generar aprox. 7.8 MHz (RSELx=13, DCOx=3, MODx=0)
      // MCLK = DCOCLK -- SMCLK = DCOCLK/8 -- ACLK = VLOCLK (aprox. 12 KHz).
      DCOCTL = DCO1 | DCO0;
      BCSCTL1 = RSEL3 | RSEL2 | RSEL0;
      BCSCTL2 = DIVS1 | DIVS0;        // SMCLK = DCOCLK / 8
      BCSCTL3 = LFXT1S1;              // ACLK = VLOCLK

      // Configura el Puerto P1 como E/S (con P1.0 maneja el led rojo)

      P1SEL = 0x00;    // P1 completo como E/S
      P1DIR = 0x01;    // P1.0 salida; el resto entrada
      P1OUT = 0x00;    // Apaga el led controlado por P1.0

      while(1)
      {
            for(i=0; i<10; ++i)             // Repite 10 veces un retardo de
                  for(j=0; j<65000; ++j);   // aproximadamente 0.05 seg

            P1OUT ^= 0x01;     // Cada 0.5 seg conmuta el estado del led rojo
      }
}
```

Lab. 4: Atención de interrupciones.

Sobre la base del programa del Lab. 3, se configura el puerto P1.3, que recibe la señal proveniente del pulsador, para generar una interrupción cada vez que se oprima el pulsador (el pulsador oprimido se lee como un valor cero en P1.3).
En el programa se puede observar la configuración del puerto P1.3, la habilitación general de interrupciones y la función que trabaja como rutina de atención de interrupción (cada vez que ésta se ejecuta, cambia el estado del led verde controlado desde el puerto P1.6 y se modifica la frecuencia de parpadeo del led rojo –más rápido cuando el led verde está encendido–).

```c
/*****************************************************************************
 * Lab04: Configuración del puerto P1.3 para generar una interrupción por
 *        flanco descendente.Dentro de la rutina de interrupción se
 *        se conmuta el estado del led azul.
 *        Obs.: el led rojo cambia de estado cada 0.5 seg aproximadamente.
 *        Cuando se oprime el pulsador conectado a P1.3 se produce una
 *        interrupción que, al ser atendida, se usa para conmutar el estado
 *        del led verde (P1.6) y se modifica la frecuencia de parpadeo
 *        del led rojo.
 * Micro: MSP430G2211
 *****************************************************************************/
#include <msp430.h>

volatile int n;

void main(void)
{
	unsigned int i, j;

	WDTCTL = WDTPW + WDTHOLD;  // Detener el watchdog

	// Configura el DCO para generar aprox. 7.8 MHz (RSELx=13, DCOx=3, MODx=0)
	// MCLK = DCOCLK -- SMCLK = DCOCLK/8 -- ACLK = VLOCLK (aprox. 12 KHz).
	DCOCTL = DCO1 | DCO0;
	BCSCTL1 = RSEL3 | RSEL2 | RSEL0;
	BCSCTL2 = DIVS1 | DIVS0;
	BCSCTL3 = LFXT1S1;

	// Configura P1 como E/S (P1.0: led rojo / P1.3: pulsador / P1.6: led azul)
	P1SEL = 0x00;  // Todo P1 como E/S
	P1DIR = 0x41;  // P1.0 y P1.6 salida; el resto entrada
	P1OUT = 0x08;  // Apaga leds conectados a P1.0 y P1.6 – Activa pull-up P1.3
	P1IE  = 0x08;  // Habilita interrupciones para P1.3
	P1IES = 0x08;  // Config. P1.3 p/generar interrupción por flanco descendente

	// Habilitación general de interrupciones del micro. Esta macro genera la
	// instrucción EINT, que pone a 1 el bit GIE (global interrupt enable) del
	// registro SR (status register)
	_enable_interrupts();

	// Contador del bucle exterior inicialmente en 10
	// Cuando se enciende el led verde se reduce a 5
	n = 10;

	while(1)
	{
		for(i=0; i<n; ++i)              // Repite n veces un retardo de
			for(j=0; j<65000; ++j);   // aproximadamente 0.05 seg

		P1OUT ^= BIT0;      // Cada 0.5 seg conmuta el estado del led rojo,
		                    //  controlado por P1.0
	}
}
```

```
//-------------------------------------------------
// Rutina de atención de interrupción del puerto P1
//-------------------------------------------------
#pragma vector=PORT1_VECTOR
__interrupt void P1_Interrupt(void)
{
      if( P1IFG & BIT3 )  // Verifica la causa de interrupción (en este ejemplo no
                          // es necesario)
      {
            P1OUT ^= BIT6;      // Conmuta el estado del led controlado por P1.6

            P1IFG &= ~BIT3;     // Reseta IFG para P1.3

            n=(P1OUT & BIT6)?5:10;// Modifica frecuencia de parpadeo del led rojo
      }
}
```

Lab. 5: configuración y uso del Timer A2, usando interrupciones.

Sobre la base del programa del Lab. 3, ahora se usará la señal ACLK (~ 12 KHz) para controlar el Timer A2 en modo ascendente. El objetivo es generar una interrupción cada 0.5 seg, a fin de conmutar el estado del led rojo (P1.0) con esa periodicidad.

```
/*****************************************************************************
 * Lab05: Se usa la rutina de interrupción del puerto P1.3 para modificar
 *        la frecuencia de parpadeo del led rojo (P1.0).
 *        Se configura el Timer A2 para generar una interrupción cada 0.5 seg
 *        aproximadamente. Dentro de la rutina de atención de interrupción
 *        del Timer A2 se conmuta el estado del led verde (P1.6).
 *        El Timer A2 se mueve en base a ACLK, que se obtiene de VLOCLK / 4
 *        (aproximadamente 3 KHz).
  * Micro: MSP430G2211
 *****************************************************************************/

#include <msp430.h>

volatile int n;

void main(void)
{
      unsigned int i, j;

      WDTCTL = WDTPW + WDTHOLD;  // Detener el watchdog

      // Configura el DCO para generar aprox. 7.8 MHz (RSELx=13, DCOx=3, MODx=0)
      // MCLK = DCOCLK -- SMCLK = DCOCLK/8 -- ACLK = VLOCLK (aprox. 12 KHz) / 4.
      DCOCTL = DCO1 | DCO0;
      BCSCTL1 = RSEL3 | RSEL2 | RSEL0 | DIVA_2; // RSELx = 13 // Div ACLK: 4
      BCSCTL2 = DIVS_3;                         // SMCLK = DCO / 8
      BCSCTL3 = LFXT1S1;                        // ACLK = VLOCLK / DIVA

      // Configura P1 como E/S (P1.0: led rojo / P1.3: pulsador / P1.6: led azul)

      P1SEL = 0x00;  // Todo P1 como E/S
      P1DIR = 0x41;  // P1.0 y P1.6 salida; el resto entrada
      P1OUT = 0x08;  // Apaga leds conectados a P1.0 y P1.6 – Activa pull-up P1.3
      P1IE  = 0x08;  // Habilita interrupciones para P1.3
      P1IES = 0x08;  // Config. P1.3 p/generar interrupción p/flanco descendente
```

```
        // Configura Timer A para generar una interrupción cada 0.5 seg aprox.
        // Considerando que ACLK=3 KHz, se requieren 1500 cuentas para 0.5 seg
        TACCR0 = 1500;
        TACTL = TASSEL_1 | MC_1 | TAIE;

        // Habilitación general de interrupciones. Esta macro genera la instrucción
        // EINT, que pone a 1 el bit GIE (global interrupt enable) del registro SR
        // (status register)
        _enable_interrupts();

        // Contador del bucle exterior inicialmente en 10
        // Presionando el botón P1.3 se modifica n a 5, 10 o 20
        n = 10;

        while(1)
        {
              for(i=0; i<n; ++i)              // Repite n veces un retardo de
                    for(j=0; j<65000; ++j);   // aproximadamente 0.05 seg

              P1OUT ^= BIT0;    // Conmuta estado del led rojo, controlado por P1.0
        }
}

//-------------------------------------------------------
// Rutina de atención de interrupción del puerto P1
//-------------------------------------------------------

#pragma vector=PORT1_VECTOR
__interrupt void P1_Interrupt(void)
{
        if( P1IFG & BIT3 )    // Verifica la fuente de interrupción (en este ejemplo
                              // no sería necesario, ya que hay una sola entrada
                              // configurada para generar interrupciones)
        {
              P1IFG &= ~BIT3;        // Reseta IFG para P1.3
              n = (n<20)? n*2 : 5;  // Modifica frecuencia de parpadeo del led rojo
        }
}

//-------------------------------------------------------
// Rutina de atención de interrupción del Timer A
//-------------------------------------------------------

#pragma vector=TIMERA1_VECTOR
__interrupt void TimerA1_Interrupt(void)
{
        if( TAIV == 0x0a )         // Verifica origen de la interrupción
              P1OUT ^= BIT6;      // Conmuta el estado del led controlado por P1.6
}
```

Lab. 6:

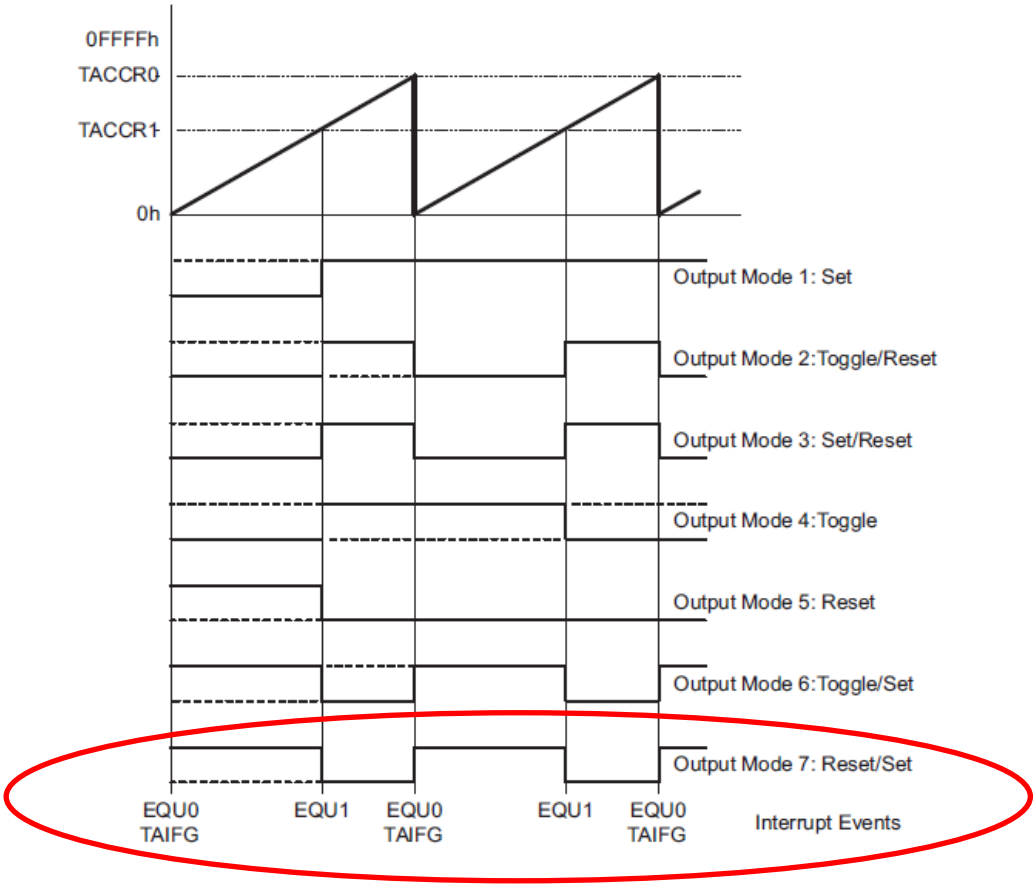
A diferencia del Lab. 5, en este caso el parpadeo del led verde (P1.6) no se mediante instrucciones en la rutina de interrupción del timer, sino que se configura el timer para conmutar el estado de la salida OUT1 en cada vencimiento de la cuenta, y se configura P1.6 para reflejar el estado de OUT1.

```c
/*****************************************************************************
 * Lab06: Se configura el Timer A2 para hacer conmutar el estado de P1.6,
 *        inicialmente cada 0.5 seg.
 *        En la rutina de atención de interrupción de P1.3 se modifica la
 *        frecuencia de parpadeo del led verde entre 0.25 seg, 0.5 seg y 1 seg.
 *        El Timer A2 se mueve en base a ACLK, que se obtiene de VLOCLK / 4
 *        (aproximadamente 3 KHz).
  * Micro: MSP430G2211
 *****************************************************************************/

#include <msp430.h>

volatile int n, m=1500;

void main(void)
{
      unsigned int i, j;

      WDTCTL = WDTPW + WDTHOLD;  // Detener el watchdog

      // Configura el DCO para generar aprox. 7.8 MHz (RSELx=13, DCOx=3, MODx=0)
      // MCLK = DCOCLK -- SMCLK = DCOCLK/8 -- ACLK = VLOCLK (aprox. 12 KHz).
      DCOCTL = DCO1 | DCO0;
      BCSCTL1 = RSEL3 | RSEL2 | RSEL0 | DIVA_2; // RSELx = 13 // Div. para ACLK: 4
      BCSCTL2 = DIVS_3;                         // SMCLK = DCO / 8
      BCSCTL3 = LFXT1S1;                        // ACLK = VLOCLK / DIVA

      // Configura P1
      P1SEL = 0x40;  // P1.6 como OUT1, el resto como E/S
      P1DIR = 0x41;  // P1.0 y P1.6 como salida; el resto entrada
      P1OUT = 0x08;  // Apaga leds conectados a P1.0 y P1.6 – Activa pull-up P1.3
      P1IE  = 0x08;  // Habilita interrupciones para P1.3
      P1IES = 0x08;  // Config. P1.3 p/generar interrupción por flanco descend.

      // Config. Timer A para generar una interrupción cada 0.5 seg aprox.
      // Considerando que ACLK=3 KHz, se requieren 1500 cuentas para lograr 0.5 s
      TACCR0 = 1500;
      TACTL = TASSEL_1 | MC_1;  // Modo ascendente y comparación
      TACCTL1 = OUTMOD_4;       // Conmutar OUT1

      // Habilitación general de interrupciones del micro. Esta macro genera la
      // instrucción EINT, que pone a 1 el bit GIE (global interrupt enable) del
      // registro SR (status register)
      _enable_interrupts();

      // Contador del bucle exterior inicialmente en 10
      // Presionando el botón P1.3 se modifica m (500-1000-1500)

      while(1)
      {
            if( P1IN & BIT3 )
            {
                  for(i=0; i<10; ++i)           // Repite 10 veces un retardo de
                        for(j=0; j<65000; ++j); // aproximadamente 0.05 seg

                  P1OUT ^= BIT0;          // Conmuta el estado del led rojo (P1.0)
            }
      }
}
```

```
//-------------------------------------------------------
// Rutina de atención de interrupción del puerto P1
//-------------------------------------------------------

#pragma vector=PORT1_VECTOR
__interrupt void P1_Interrupt(void)
{
      if( P1IFG & BIT3 )  // Verifica la fuente de interrupción (en este ejemplo
                          // no sería necesario, ya que hay una sola entrada
                          // configurada para generar interrupciones)
      {
            P1IFG &= ~BIT3;           // Reseta IFG para P1.3
            // Modifica frecuencia de parpadeo del led verde
            if(m >= 3000)
                  m = 750;
            else
                  m *= 2;
            TACCR0 = m;
      }
}
```

Lab. 7: PWM

Efectuando algunas modificaciones al código del Lab. 6, se puede lograr que la salida OUT1 (P1.6) produzca una onda rectangular cuya frecuencia se controla mediante el valor de TACCR0 y cuyo ciclo de trabajo se controla mediante el valor de TACCR1, según se puede observar en la figura de más abajo.



```c
/*****************************************************************************
 * Lab07: Se configura el Timer A2 para hacer conmutar el estado de P1.6 a
 *        aprox. 500 Hz, con un ciclo de trabajo proporcional al valor de TACCR1.
 *        En la rutina de atención de interrupción de P1.3 se modifica el
 *        ciclo de trabajo de OUT1 (4% - 12% y 91%).
 * Micro: MSP430G2211
 *****************************************************************************/
#include <msp430.h>

volatile int m;

void main(void)
{
      unsigned int i, j;

      WDTCTL = WDTPW + WDTHOLD;  // Detener el watchdog

      // Configura el DCO para generar aprox. 7.8 MHz (RSELx=13, DCOx=3, MODx=0)
      // MCLK = DCOCLK -- SMCLK = DCOCLK/8 -- ACLK = VLOCLK (aprox. 12 KHz).
      DCOCTL = DCO1 | DCO0;
      BCSCTL1 = RSEL3 | RSEL2 | RSEL0;  // RSELx = 13 // Divisor para ACLK: 1
      BCSCTL2 = DIVS_3;                 // SMCLK = DCO / 8
      BCSCTL3 = LFXT1S1;                // ACLK = VLOCLK / DIVA

      // Configura P1 (P1.0: led rojo / P1.3: pulsador / P1.6: led azul)
      P1SEL = 0x40;  // P1.6 como OUT1, el resto como E/S
      P1DIR = 0x41;  // P1.0 y P1.6 como salida; el resto entrada
      P1OUT = 0x08;  // Apaga leds conectados a P1.0 y P1.6 – Activa pull-up P1.3
      P1IE  = 0x08;  // Habilita interrupciones para P1.3
      P1IES = 0x08;  // Config. P1.3 p/generar interrup. por flanco descendente

      // Configura Timer A para generar una señal de 500 Hz en OUT1.
      // Considerando que ACLK=12 KHz, se requieren 24 cuentas para lograr 500 Hz
      TACCR0 = 24;
      TACCR1 = m = 1;          // Inicialmente el ciclo de trabajo es del 4 %
      TACTL = TASSEL_1 | MC_1; // Modo ascendente y comparación
      TACCTL1 = OUTMOD_7;      // Acción sobre OUT1: Reset-Set

      // Habilitación general de interrupciones del micro. Esta macro genera la
      // instrucción EINT, que pone a 1 el bit GIE (global interrupt enable) del
      // registro SR (status register)
      _enable_interrupts();

      // Queda en un bucle infinito sin hacer nada.
      while(1);
}
//-------------------------------------------------------
// Rutina de atención de interrupción del puerto P1
//-------------------------------------------------------
#pragma vector=PORT1_VECTOR
__interrupt void P1_Interrupt(void)
{
      if( P1IFG & BIT3 )        // Verifica que la interrupción provenga de P1.3
      {
            P1IFG &= ~BIT3;     // Reseta IFG para P1.3
            // Modifica ciclo de trabajo de OUT1 (y la intensidad del led verde)
            if(m==1)      m=3;  // 12 %
            else if(m==3) m=11; // 91 %
            else          m=1;  // 4 %
            TACCR1 = m;
      }
}
```

Lab. 09: Uso de un cristal para generar la señal de reloj

En este caso, se modifica el programa del Lab. 08, a fin de utilizar el cristal de 32768 Hz conectado a XIN y XOUT, para generar la señal de reloj ACLK.
Se usa ACLK para generar una señal de 1024 Hz en OUT1 (P1.6), cuyo ciclo de trabajo se modifica para variar la intensidad luminosa del led verde (P1.6).

Se aprovecha para mostrar algunas buenas prácticas de programación, como el uso de funciones separadas para realizar las distintas tareas, y el uso de variables const sobre memoria flash, a fin de ahorrar memoria RAM.

```
/*****************************************************************************
 * Lab09: Genera ACLK a partir del cristal de 32768 Hz.
 *        Configura el Timer A2 para hacer conmutar el estado de P1.6 a
 *        aprox. 1024 Hz, con un ciclo de trabajo proporcional al valor de TACCR1.
 *        En la rutina de atención de interrupción de P1.3 se modifica el
 *        ciclo de trabajo (3% - 12.5% - 50% - 99.9%).
 * Micro: MSP430G2211
 *****************************************************************************/
#include <msp430.h>

void InitClock(void);
void InitPorts(void);
void InitTimer(void);

const unsigned int taccr1[4] = {1,4,16,31};

volatile int taccr1_index = 0;

void main(void)
{
      WDTCTL = WDTPW + WDTHOLD;  // Detener el watchdog

      InitClock();
      InitPorts();
      InitTimer();

      // Habilitación general de interrupciones del micro. Esta macro genera la
      // instrucción EINT, que pone a 1 el bit GIE (global interrupt enable) del
      // registro SR (status register)
      _enable_interrupts();

      while(1); // Bucle infinito
}
//-------------------------------------------------------
// Rutina de atención de interrupción del puerto P1
//-------------------------------------------------------
#pragma vector=PORT1_VECTOR
__interrupt void P1_Interrupt(void)
{
      if( P1IFG & BIT3 )  // Verifica que la interrupción provenga de P1.3
      {
            P1IFG &= ~BIT3;           // Reseta IFG para P1.3

            // Modifica ciclo de trabajo de OUT1, y con ello
            // la intensidad luminosa del led verde (P1.6)
            taccr1_index = ++taccr1_index % 4;
            TACCR1 = taccr1[ taccr1_index ];
      }
}
```

```
//-------------------------------------------------------------------------------
// Configura el DCO para generar aprox. 7.8 MHz (RSELx=13, DCOx=3, MODx=0)
// MCLK = DCOCLK -- SMCLK = DCOCLK/8 -- ACLK = LFXT1CLK (32768 Hz).
//-------------------------------------------------------------------------------
void InitClock(void)
{
      DCOCTL = DCO1 | DCO0;
      BCSCTL1 = RSEL3 | RSEL2 | RSEL0 ;  // RSELx = 13 // Divisor para ACLK: 1
      BCSCTL2 = DIVS_3;                  // SMCLK = DCO / 8
      BCSCTL3 = LFXT1S_0;                // ACLK = LFXT1CLK / DIVA
}

//-------------------------------------------------------------------------------
// Configura P1 como E/S (P1.0: led rojo / P1.3: pulsador / P1.6: led azul)
//-------------------------------------------------------------------------------
void InitPorts(void)
{
      P1SEL = 0x40;  // P1.6 como OUT1, el resto como E/S
      P1DIR = 0x41;  // P1.0 y P1.6 como salida; el resto entrada
      P1OUT = 0x08;  // Apaga leds conectados a P1.0 y P1.6 – Activa pull-up P1.3
      P1IE  = 0x08;  // Habilita interrupciones para P1.3
      P1IES = 0x08;  // Config. P1.3 p/generar interrup. por flanco descendente
}

//-------------------------------------------------------------------------------
// Configura Timer A para generar una señal de 500 Hz en OUT1
// Considerando que ACLK=32768 Hz, se requieren 32 cuentas para lograr 1024 Hz
//-------------------------------------------------------------------------------
void InitTimer(void)
{
      TACCR0 = 32;                       // ACLK / 32 = 1024 Hz
      TACCR1 = taccr1[ taccr1_index ];   // Ciclo de trabajo inicial (
      TACTL = TASSEL_1 | MC_1;           // Modo ascendente y comparación
      TACCTL1 = OUTMOD_7;                // Acción sobre OUT1: Reset-Set
}
```

Lab. 10: Medición de tiempo a partir de la frecuencia del cristal

Modificando la función `InitTimer()` del Lab. 09 por la que se indica a continuación, se produce la conmutación de OUT1 a una frecuencia de 2 Hz, por lo que se podrá apreciar que el led parpadea exactamente cada 1 seg.

```
void InitTimer(void)
{
      TACCR0 = 16384;                    // ACLK / 16384 = 2 Hz
      TACCR1 = 0;
      TACTL = TASSEL_1 | MC_1;           // Modo ascendente y comparación
      TACCTL1 = OUTMOD_4;                // Acción sobre OUT1: conmutar
}
```

Obs.: se deberá eliminar también el código de la rutina de interrupción de P1.3 que modifica el valor de TACCR1.

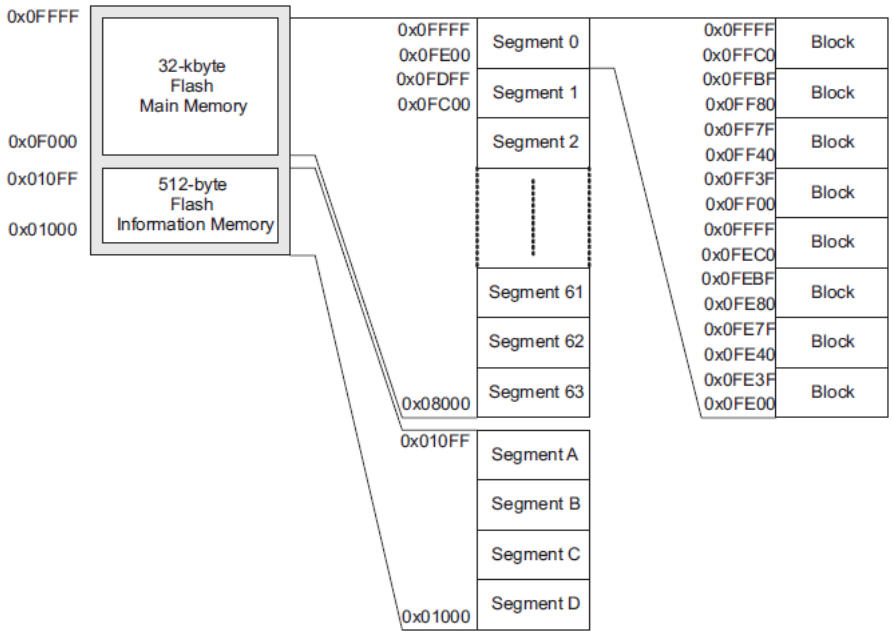
Lab. 11: Escritura en memoria flash

La memoria flash de los micros MSP430 se encuentra dividida en dos grandes secciones: memoria principal (de programa) y memoria de información. La ubicación en el mapa de

memoria y la extensión de cada una de estas áreas depende del modelo del micro. Por ejemplo:

	Memoria Principal		Memoria de Información	
	Tamaño	Ubicación	Tamaño	Ubicación
MSP430G2211	2 KB (4 x 512 B)	F800h – FFFFh	256 B (4 x 64 B)	1000h – 10FFh
MSP430G2231	2 KB (4 x 512 B)	F800h – FFFFh	256 B (4 x 64 B)	1000h – 10FFh
MSP430G2553	16 KB (32 x 512 B)	C000h – FFFFh	256 B (4 x 64 B)	1000h – 10FFh
MSP430G2452	8 KB (16 x 512 B)	E000h – FFFFh	256 B (4 x 64 B)	1000h – 10FFh
MSP430F149	~60 KB (2 x 256 B + 118 x 512 B)	1100h – FFFFh	256 B (4 x 64 B)	1000h – 10FFh

La siguiente figura muestra la organización de la memoria flash para un micro con 32 KB de memoria principal:



Para poder efectuar operaciones de escritura en la memoria flash hay que resolver las siguientes cuestiones:

- Inicializar el controlador de memoria flash
- Borrado de la memoria (en forma masiva o por segmento)
- Escritura (por bit, byte, palabra

```
/*******************************************************************************
 * Lab11: Ejemplo de manejo de la memoria flash para almacenar datos que
 *        deben ser no volátiles.
 *        Genera ACLK a partir del cristal de 32768 Hz y configura el Timer A2
 *        para hacer conmutar el estado de P1.6 a una frecuencia que puede
 *        ser modificada mediante el pulsador conectado a P1.3.
 *        El valor de la frecuencia de parpadeo del led se almacena en flash
 *        para que al reiniciarse el micro, comience con la última frecuencia
 *        con que haya trabajado.
 * Micro: MSP430G2211
 *******************************************************************************/
#include <msp430.h>

void InitClock(void);
void InitPorts(void);
```

```
void InitTimer(void);

void InitFlashCtl(void);
void flash_clr(unsigned int *);
void flash_ww(unsigned int *, unsigned int);

#define FRECUENCIA (*(unsigned int *)0x1000)

volatile unsigned int frecuencia;

//-------------------------
// Inicio del programa
//-------------------------
void main(void)
{
	if( (frecuencia=FRECUENCIA) > 32768 )
		frecuencia = 32768;

	WDTCTL = WDTPW + WDTHOLD;  // Detener el watchdog

	InitClock();
	InitPorts();
	InitTimer();
	InitFlashCtl();

	// Habilitación general de interrupciones del micro. Esta macro genera la
	// instrucción EINT, que pone a 1 el bit GIE (global interrupt enable) del
	// registro SR (status register)
	_enable_interrupts();

	while(1)
	{
		 if( frecuencia != FRECUENCIA )
		 {
			// Se ha producido un cambio en la frecuencia y
			// graba el nuevo valor en la flash

			flash_clr( &FRECUENCIA );            // borra el segmento

			flash_ww( &FRECUENCIA, frecuencia ); // escribe el nuevo valor
		 }
	}
}
//-----------------------------------------------------
// Rutina de atención de interrupción del puerto P1
//-----------------------------------------------------
#pragma vector=PORT1_VECTOR
__interrupt void P1_Interrupt(void)
{
	if( P1IFG & BIT3 )  // Verifica que la interrupción provenga de P1.3
	{
		P1IFG &= ~BIT3;           // Reseta IFG para P1.3

		// Modifica frecuencia de parpadeo del led verde (P1.6)
		if( frecuencia > 32768 || frecuencia < 4096 )
			frecuencia = 32768;
		else
			frecuencia = frecuencia >> 1;

		TACCR0 = frecuencia;
	}
}
```

```
//-------------------------------------------------------------------------------
// Configura el DCO para generar aprox. 7.8 MHz (RSELx=13, DCOx=3, MODx=0)
// MCLK = DCOCLK -- SMCLK = DCOCLK/8 -- ACLK =      LFXT1CLK (32768 Hz).
//-------------------------------------------------------------------------------
void InitClock(void)
{
      DCOCTL = DCO1 | DCO0;
      BCSCTL1 = RSEL3 | RSEL2 | RSEL0 ;  // RSELx = 13 // Divisor para ACLK: 1
      BCSCTL2 = DIVS_3;                  // SMCLK = DCO / 8
      BCSCTL3 = LFXT1S_0;                // ACLK = LFXT1CLK / DIVA
}

//-------------------------------------------------------------------------------
// Configura P1 como E/S (P1.0: led rojo / P1.3: pulsador / P1.6: led azul)
//-------------------------------------------------------------------------------
void InitPorts(void)
{
      P1SEL = 0x40;  // P1.6 como OUT1, el resto como E/S
      P1DIR = 0x41;  // P1.0 y P1.6 como salida; el resto entrada
      P1OUT = 0x08;  // Apaga leds conectados a P1.0 y P1.6 – Activa pull-up P1.3
      P1IE  = 0x08;  // Habilita interrupciones para P1.3
      P1IES = 0x08;  // Config. P1.3 p/generar interrup. por flanco descendente
}
//-------------------------------------------------------------------------------
// Configura Timer A para generar una onda cuadrada de la frecuencia
// Considerando que ACLK=32768 Hz, si TACCR0==16384 el led parpadea a 1 Hz
//-------------------------------------------------------------------------------
void InitTimer(void)
{
      TACCR0 = frecuencia;
      TACCR1 = 0;
      TACTL = TASSEL_1 | MC_1;          // Modo ascendente y comparación
      TACCTL1 = OUTMOD_4;               // Acción sobre OUT1: conmutar
}
//-------------------------------------------------------------------------------
// Escribe una palabra en la flash
//-------------------------------------------------------------------------------
void flash_ww( unsigned int *Data_ptr, unsigned int word )
{
  _disable_interrupt();
  FCTL3 = 0x0A500;  // Lock = 0
  FCTL1 = 0x0A540;  // WRT = 1
  *Data_ptr=word;   // program Flash word
  FCTL1 = 0x0A500;  // WRT = 0
  FCTL3 = 0x0A510;  // Lock = 1
  _enable_interrupt();
}
//-------------------------------------------------------------------------------
// Borra un segmento en la flash
//-------------------------------------------------------------------------------
void flash_clr( unsigned int *Data_ptr )
{
  _disable_interrupt();
  FCTL3 = 0x0A500;  // Lock = 0
  FCTL1 = 0x0A502;  // Erase = 1
  *Data_ptr=0;      // dummy write
  FCTL1 = 0x0A500;  // Erase = 0
  FCTL3 = 0x0A510;  // Lock = 1
  _enable_interrupt();
}
```

```
//-------------------------------------------------------------------------------
// Inicializa el controlador de memoria flash
//-------------------------------------------------------------------------------
void InitFlashCtl()
{
  FCTL2 = 0x0A500 | FSSEL_1 | 18;     // MCLK (~7.8 MHz) / 19 = 410 KHz
}
```