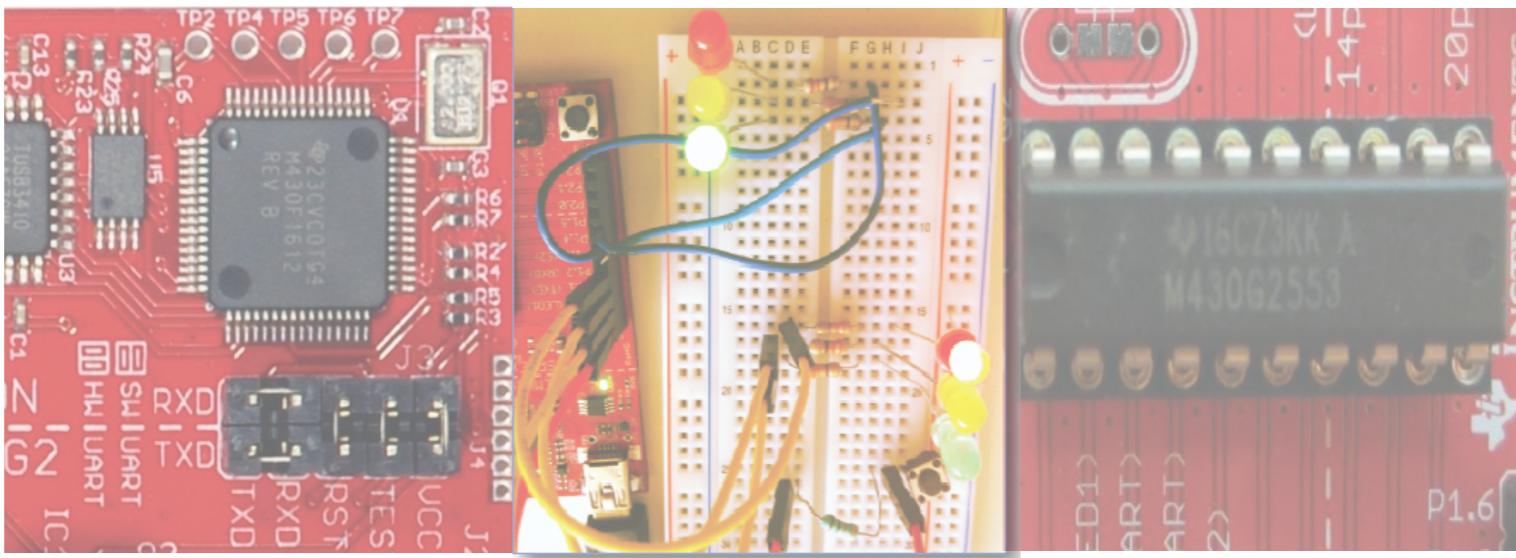




Manual de Laboratorio de Microprocesadores y Microcontroladores



M.en I. Eduardo Ramírez Sánchez

Ing. Gabriela Hernández Larios

Manual de Laboratorio de Microprocesadores y Microcontroladores

Tabla de contenidos

Prefacio	ii
Práctica 1:Retardo con LED externo y push-button	4
Práctica 2:Semáforos	10
Práctica 3:Lectura de una entrada digital y Configuración de resistencias internas	13
Práctica 4:Interrupciones de los puertos	21
Práctica 5:Convertidor Analógico-Digital (ADC10)	32
Práctica 6:Timer_A como temporizador	43
Práctica 7:Timer_A en modo comparación	55
Práctica 8:Timer_A en modo captura	67
Práctica 9:Comunicación Serial Síncrona: USCI en modo SPI	77
Práctica 10:Comunicación Serial Asíncrona:USCI en modo UART	92
Bibliografía y Mesografía	104

Prefacio

“La teoría sin la práctica es estéril, la práctica sin teoría es ciega.”

-Torres H.

Este libro cuenta con prácticas elaboradas para la materia de Laboratorio de Microprocesadores y Microcontroladores, usando el microcontrolador de Texas Instruments de bajo costo y bajo consumo MSP430G2553 LaunchPad ®.

Las iniciales MSP resultan de la abreviación de Mixed Signal Processor, lo cual indica que se trata de un procesador de señal mixta, que si bien permite el uso y generación de señales en forma digital, también sus periféricos pueden trabajar con señales analógicas.

El microcontrolador MSP430G2553®, es considerado como un microcontrolador muy competente ya que presenta características únicas, una de las principales que lo hacen sobresaliente en el mercado es que si bien es un microcontrolador de bajo costo es de 16 bits lo cual se traduce en una mejor precisión en las operaciones y/o aplicaciones a desarrollar.

Además este microcontrolador es el que menos energía consume en el mercado, incluso por debajo de los famosos PIC's.

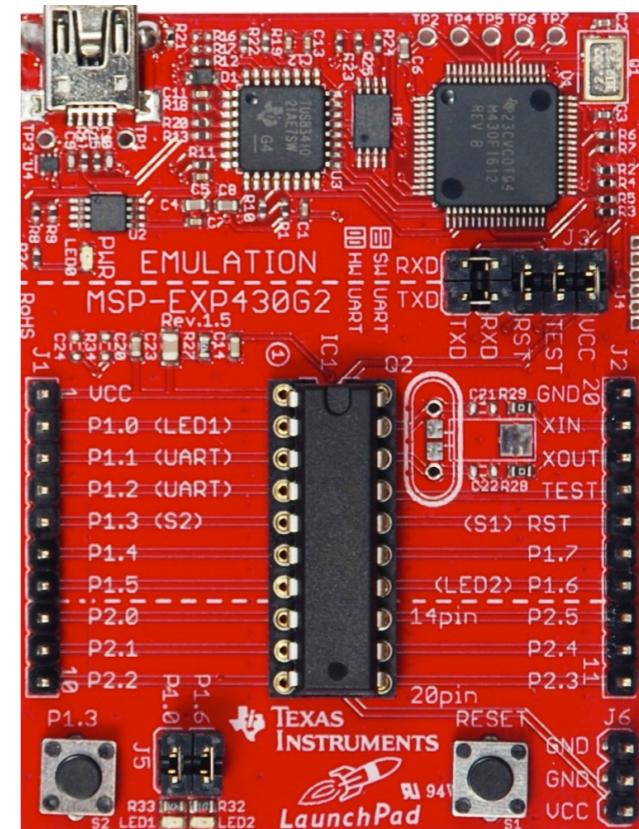
Este microcontrolador cuenta con una CPU RISC de 16 bits y una arquitectura Von Neuman y tiene un sistema de reloj muy flexible. Otra característica es que sus herramientas son muy fáciles de usar.

A continuación se enlistan otras de sus principales características:

- ◆ Rango de voltaje: 1.8V a 3.6 V.
- ◆ Ultra-bajo consumo de potencia.
- ◆ Dos Timer_A de 16 bits con tres registros de Captura/Comparación cada uno.
- ◆ Interface de Comunicación Serial Universal (USCI)
- ◆ Convertidor Analógico-Digital de 10 bits.

Para mayor detalle sobre sus características principales ver la hoja de especificación de datos que proporciona Texas Instruments.

<http://www.ti.com/lit/ds/symlink/msp430g2553.pdf>



Práctica 1

Retardo con LED externo y Push button.

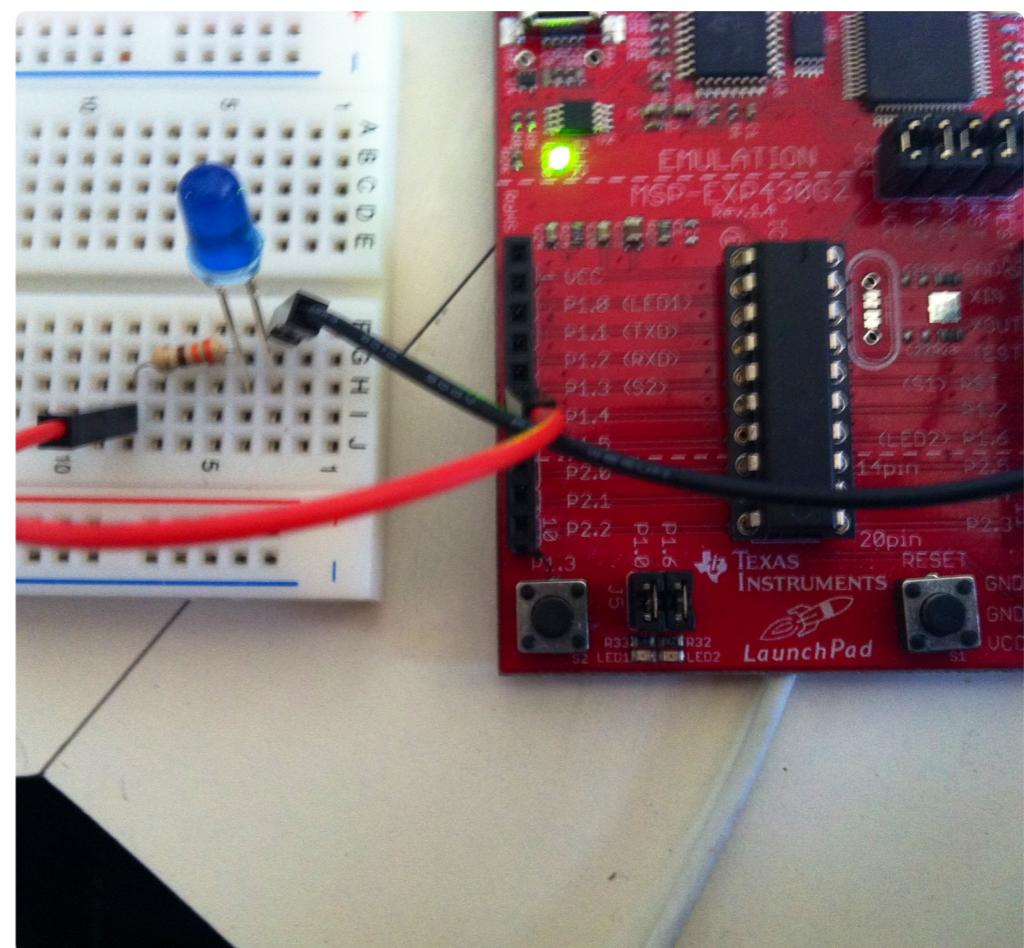
OBJETIVOS

1. Conocer el funcionamiento y configuración de los puertos digitales del MSP430G2553.
2. Aprender la conexión de la Tarjeta y un LED externo.
3. Conocer el funcionamiento de las subrutinas de retardo, y cómo obtener el tiempo de retardo requerido.
4. Aprender a ejecutar el programa paso a paso, así como depurarlo y ejecutarlo en una sola corrida.

MATERIALES

- 1 Microcontrolador MSP430G2553 LaunchPad
- 1 Placa protoboard.
- 1 LED
- 1 Resistencia de 220 ó 330 Ohms.

El desarrollo de esta práctica consiste en leer el push-button conectado al bit 3 del puerto 1 (pin 1.3), de modo que cuando éste esté presionado un LED externo encienda por 250ms, transcurrido este tiempo se apaga el LED. Posteriormente modificar el tiempo de encendido por 1 segundo. Hacer el programa en lenguaje assembly.



Metodología

LED

El diodo emisor de luz (LED), es un diodo que emite luz visible o invisible (infrarroja) cuando se energiza. En cualquier



unión p-n polarizada en directa se tiene dentro de la estructura y principalmente cerca de la unión, una recombinación de huecos y electrones.

Dicha recombinación ocupa que la energía procesada por los electrones libres se transforme a otro estado. En todas las uniones p-n semiconductoras una parte de esta energía se libera en forma de calor y otra en forma de fotones.

En los diodos de Si y Ge el mayor porcentaje de energía convertida durante la recombinación en la unión se disipa en forma de calor dentro de la estructura y la luz emitida es insignificante. Los materiales de los LEDs permiten que la mayor parte de dicha energía convertida se disipe en forma de fotones.

Los diodos construidos de GaAs emiten luz en una zona infrarroja (invisible) durante el proceso de recombinación en la unión p-n. Mediante otras combinaciones de elementos se puede generar luz visible. La tabla 1.1 proporciona la lista de semiconductores compuestos comunes, la luz que genera y el intervalo de potenciales de polarización directa de cada uno. 1

Tabla 1.1. Diodos Emisores de Luz

Color	Elemento combinados	Voltaje en directa típico [Volts]
Ámbar	AlInGaP	2.1
Azul	GaN	5.0
Verde	GaP	2.2
Naranja	GaAsP	2.0
Rojo	GaAsP	1.8
Blanco	GaN	4.1
Amarillo	AlInGaP	2.1

Subrutinas de retardo

Las subrutinas son una herramienta muy importante en lenguaje assembly, gracias a éstas se puede reducir el código, ya que cuando una pequeña parte del código se tenga que repetir varias veces se utiliza una subrutina. De esta manera se escribe la subrutina una sola vez y se puede mandar a llamar cuantas veces sea requerida.

La subrutina comienza con una etiqueta y termina con RET.

ETIQUETA

Código de subrutina

.

.

.

RET

Cuando se requiera la subrutina se llama ésta de la siguiente manera:

CALL #Subrutina

Una subrutina de retardo es de suma importancia para cuando se requiera cierta precisión en aplicaciones de tiempo, como encender un LED durante un determinado tiempo, generación de secuencias de luces como en los semáforos, para eliminar los rebotes de push-button y otros dispositivos, entre otras muchas aplicaciones.

La función principal de una subrutina de retardo es mantener ocupada a la CPU y que ésta consuma un cierto tiempo requerido.

Para el manejo de subrutinas de retardo es necesario conocer los ciclos de trabajo de cada instrucción, así como la frecuencia del microcontrolador, en este caso, se trabajará con la frecuencia de 1MHz.

Si la frecuencia es de 1MHz, entonces el periodo será de 1us.

A continuación se muestra una forma de realizar un retardo en código assembly.

```
;*****  
;                      SUBRUTINA DE RETARDO  
;*****  
Retardo    mov   #Data,R7  
Loop1     dec   R7  
          jnz   Loop1  
          ret
```

Los ciclos de trabajo de cada instrucción según su modo de direccionamiento se muestran a continuación.

mov #Data,R7	2 ciclos de trabajo
dec R7	2 ciclos de trabajo
jnz Loop1	2 ciclos de trabajo
ret	3 ciclos de trabajo

Debido a que **mov** y **ret** sólo se utilizan una vez en esta subrutina de retardo sólo suman 5 ciclos de trabajo, mientras que la instrucción **dec** y **jnz** se repiten según la cantidad que tenga constante, una ecuación para determinar el tiempo del retardo, o bien la variable **Data** requerida para tener un tiempo de retardo deseado es:

$$\text{Tiempo} = [5 + (\text{Data} * 4)] * \text{periodo}$$

O bien,

$$\text{Data} = [\text{Tiempo} / 4 * \text{periodo}] - 5$$

Donde:

Tiempo= Es el tiempo de la subrutina de retardo.

Data= Valor que se pone al principio de la subrutina entre 0-65530 (0xFFFFh) y determina el tiempo de retardo.

Periodo= Período del microcontrolador, en este caso 1.

Es importante mencionar, que el microcontrolador con el que se está trabajando es de 16 bits, por lo que sus registros son capaces de almacenar sólo 16 bits, debido a esto con esta ecuación el máximo retardo que se puede llevar a cabo es de 262.144ms, ya que el máximo valor para la variable **Data** es:

$$2^{16} - 1 = 65535d = 0xFFFF$$

Si se requieren llevar a cabo subrutinas de retardo mayores a 262.144ms se puede usar la subrutina de retardo y mandar a llamar ésta las veces que se requiera para obtener el tiempo deseado, lo cual genera un ciclo anidado.

Otra manera de hacer una subrutina de mayor tiempo, es introducir la instrucción **nop**, que no lleva a cabo ninguna operación y consume 2 ciclos de trabajo, este método es indicado si sólo se requieren unos micro segundos más, sin embargo, si se requiere un tiempo de retardo muy grande, este método no resulta práctico.

```
;*****  
;                               SUBRUTINA DE RETARDO  
;*****  
Retardo    mov   #Data,R7  
Loop1      nop  
           nop  
           dec   R7  
           jnz   Loop1  
           ret
```

Y la ecuación se modifica agregándole estos ciclos de trabajo de la siguiente manera:

$$\text{Tiempo} = [5 + (\text{Data} * (4 + (2 * n))) * \text{periodo}]$$

Donde n es el número de operaciones nop integradas.

Nota 1. Este tipo de retardo resultan particularmente aceptables cuando el microcontrolador tiene que realizar una tarea única. Si es el caso de que tenga que realizar un mayor número de tareas, se recomienda utilizar un timer (temporizador) el cual se revisará en prácticas posteriores.

Desarrollo de la práctica

1. Conectar el LED externo con el microcontrolador, tal como se muestra en la figura 2 y 3.

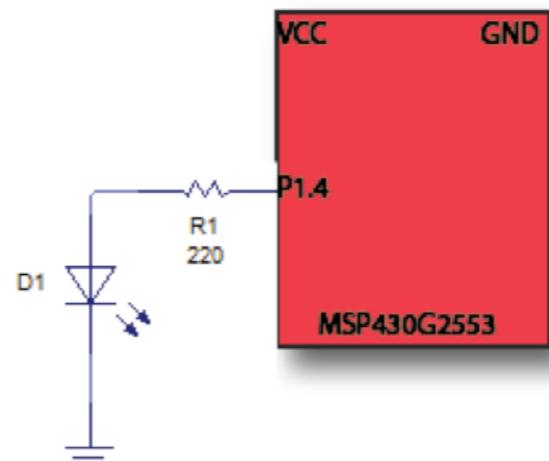


Figura 2. Diagrama de conexión del LED con el microcontrolador.

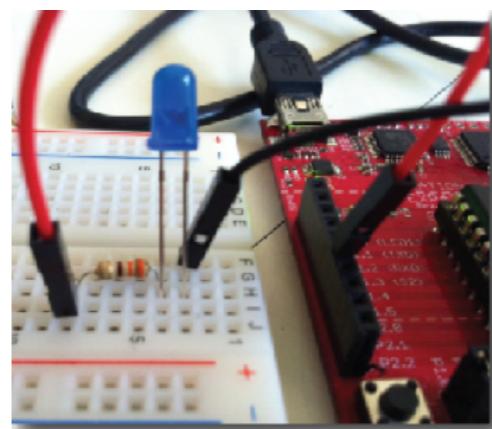


Figura 3. Conexión física del LED con el microcontrolador.

1. Robert L. Boylestad y Louis Nashelsky. "Electrónica: Teoría de Circuitos y Dispositivos Electrónicos". Editorial PEARSON. Edición 10ma. Pág. 42.

2. Cargar el siguiente programa en código assembly en el ambiente integrado, y ejecutarlo paso a paso, siguiendo las instrucciones del profesor. En este programa, la subrutina de retardo tiene un tiempo de retardo de 250ms.

;Programa que lee el push-button conectado a p1.3 del puerto 1, y enciende un LED externo por un determinado tiempo y después lo apaga.

```
.cdecls C,LIST,"msp430g2553.h"
.global RESET
.text

RESET    mov.w #8400h,SP          ;Inicialización del StackPointer
StopWDT  mov.w #WDTPW+WDTHOLD,&WDTCTL ;Detener el WatchDog
;*****
;           CONFIGURACIÓN DE PUERTO 1
;*****
bis.b #00010000b,&P1DIR      ; P1.3 entrada (push-button),
;                                ; P1.4 Salida.
;*****
;           PROGRAMA PRINCIPAL
;*****
Mainloop bit.b #00001000b,&P1IN ;Lee push-button conectado internamente
;                                ; a P1.3
        clr P1OUT
        jnc OnLED      ;Si está oprimido brinca a etiqueta
;OnLED.
        jmp Mainloop
OnLED    bis.b #00010000b,&P1OUT ;Enciende LED externo
        call #Ret250ms ;Llamado a subrutina Retardo
        bic.b #00010000b,&P1OUT ;Apaga LED externo
        jmp Mainloop      ;Brinca a etiqueta Mainloop para volver
;                                ; a preguntar por estado de push-button.
;*****
;           SUBRUTINA DE RETARDO
;En esta subrutina se genera un tiempo "muerto" deseado
;*****
Ret250ms mov #0xF41F,R7      ;Data=0xF41F para un tiempo de retardo de
;                                ;250mili-segundos.
Loop1   dec R7                ;Se decrementa el registro R7
        jnz Loop1      ;Permanece en el ciclo hasta que R7=0.
        Ret             ;Retorno de subrutina.

.sect ".reset"          ; MSP430 RESET Vector
.short RESET
.end
```

- 3.** Cambiar el tiempo de retardo por 1 segundo. Esto se logra mandando a llamar por cuatro veces la subrutina de retardo de 250ms. Por lo que no se obtendrá exactamente un segundo de retardo, si se requiere precisión, se deben de restar estos ciclos de trabajo añadidos a la variable data.

;Programa que lee el push-button conectado a p1.3 del puerto 1, y enciende un LED externo por un determinado tiempo y después lo apaga.

```
.cdecls C,LIST,"msp430g2553.h"
.global RESET
.text

RESET    mov.w #8400h,SP          ;Inicialización del StackPointer
StopWDT  mov.w #WDTPW+WDTHOLD,&WDTCTL ;Detener el WatchDog
;*****
;           CONFIGURACIÓN DE PUERTO 1
;*****
bis.b #00010000b,&P1DIR      ; P1.3 entrada (push-button),
;                                ; P1.4 Salida.
;*****
;           PROGRAMA PRINCIPAL
;*****
Mainloop bit.b #00001000b,&P1IN ;Lee push-button conectado internamente
;                                ; a P1.3
        clr P1OUT
        jnc OnLED      ;Si está oprimido brinca a etiqueta
;OnLED.
        jmp Mainloop
OnLED    bis.b #00010000b,&P1OUT ;Enciende LED externo
        call #Ret1s     ;Llamado a subrutina Retardo
        bic.b #00010000b,&P1OUT ;Apaga LED externo
        jmp Mainloop      ;Brinca a etiqueta Mainloop para volver
;                                ; a preguntar por estado de push-button.
;*****
;           SUBRUTINAS DE RETARDO
;Subrutina de retardo de 250 mili-segundos.
Ret250ms mov #0xf41F,R7      ;Data=0xf41F para un tiempo de retardo de
;                                ;250mili-segundos.
Loop1   dec R7                ;Se decrementa el registro R7
        jnz Loop1      ;Permanece en el ciclo hasta que R7=0.
        Ret             ;Retorno de subrutina.

;Subrutina de retardo de 1 segundo.
Ret1s    mov #0x04,R6      ;R6<-0x04
Loop2   call #Ret250ms ;Llamado a subrutina de retardo de 250ms.
        dec R6            ;Se decrementa R6.
        jnz Loop2      ;Si R6 no es cero permanece en la subrutina.
        Ret             ;Retorno de subrutina.

.sect ".reset"          ; MSP430 RESET Vector
.short RESET
.end
```

- 4.** Hacer el reporte correspondiente a la práctica 1.

Práctica 2

Semáforos

OBJETIVOS

1. Dominio de generación de distintas subrutinas de retardo.
2. Sincronización de encendido de LEDs.
3. Reforzar el tema de subrutinas de retardo y manejo de LEDs externos.

MATERIALES

1 Microcontrolador MSP430G2553
LaunchPad.

2 LEDs rojos.

2 LEDs amarillos.

2 LEDs verdes.

6 Resistencias de 220 o 330 ohms.

Placa protoboard.

En esta práctica se busca reforzar los conocimientos adquiridos en la práctica 1, sincronizando dos “prototipos” de semáforos de LEDs.

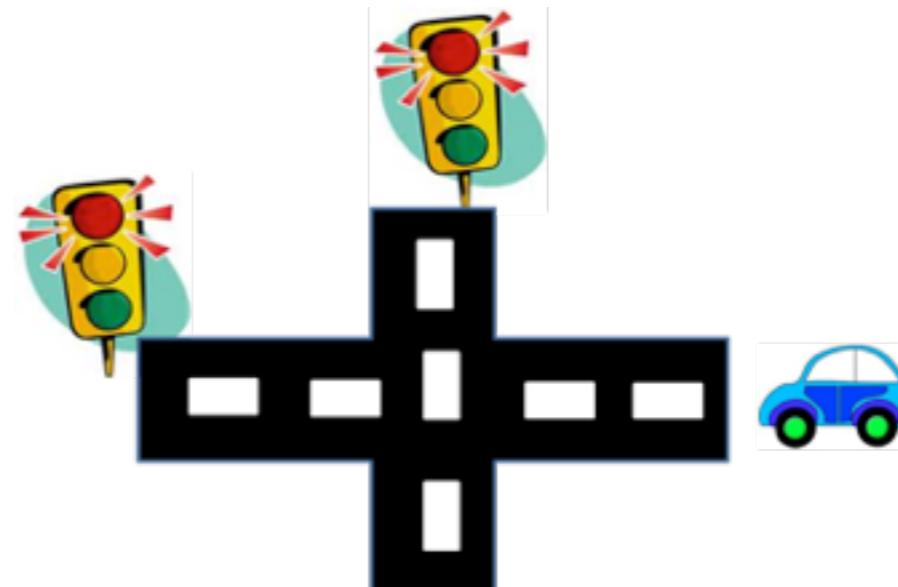


Figura 2.1. Esquemático ilustrativo de la práctica.

Desarrollo de la práctica

1. Conectar los LEDs de los “semáforos” al Puerto 1 del microcontrolador como se muestra en el diagrama de la figura 2.2.

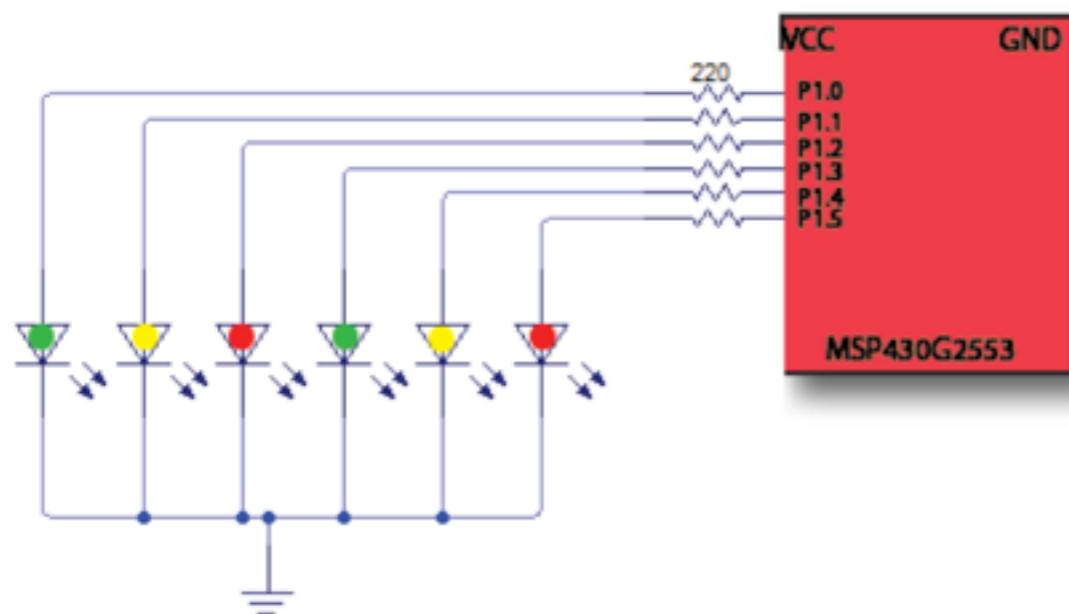


Figura 2.2. Diagrama de conexión de LEDs con el microcontrolador.

2. En base a la tabla 2.1, se llevó a cabo el programa en código Assembly, se observa que cuando se termina de ejecutar el estado cuatro se regresa al estado uno, es decir, se tiene un ciclo infinito.

Tabla 2.1. Configuración y sincronización de los semáforos.

PUERTO 1										
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	P1OUT (HEX)	Tiempo retardo	Edo
SEMÁFORO 1					SEMÁFORO 2					
		R1	A1	V1	R2	A2	V2			
0	0	1	0	0	0	0	1	0x21	2 s	1
0	0	1	0	0	0	1	0	0x22	1s	2
0	0	0	0	1	1	0	0	0x0C	2s	3
0	0	0	1	0	1	0	0	0x14	1s	4

3. Escriba y cargue el siguiente programa generado en código assembly, y posteriormente ejecute éste según las instrucciones del profesor. Una vez probado y ejecutado el programa cambie los tiempos de retardo según indique el profesor.

```

;Programa que sincroniza dos semáforos.

.cdecls C,LIST,"msp430g2553.h"
.global RESET
.text

RESET    mov.w #0400h,SP           ;Inicialización del StackPointer
StopWDT  mov.w #WDTPW+WDTHOLD,&WDTCTL ;Detener el WatchDog
;*****
;***** CONFIGURACIÓN DE PUERTO 1
;*****
;      bis.b #00111111b,&P1DIR      ;P1.0,P1.1,P1.2,P1.3,P1.4,P1.5
;                                ;como salidas.
;*****
;***** PROGRAMA PRINCIPAL
;*****
;      clr P1OUT                 ;Limpia puerto 1.
Semaforo  mov #0x21,&P1OUT          ;Estado 1.
          call #Ret2s              ;Llama a subrutina de retardo de 2s.
          mov #0x22,P1OUT          ;Estado 2.
          call #Ret1s              ;Llama a subrutina de retardo de 1s.
          mov #0x0c,&P1OUT          ;Estado 3.
          call #Ret2s              ;Llama a subrutina de retardo de 2s.
          mov #0x14,&P1OUT          ;Estado 4.
          call #Ret1s              ;Llama a subrutina de retardo de 1s.
          jmp Semaforo             ;Brinca a etiqueta Semaforo para
;                                ;regresar al Estado 1.
;*****
;***** SUBRUTINAS DE RETARDO
;*****
;Retardo de 250 mili-segundos.
Ret250ms  mov #0xf41F,R7          ;Data=0xf41F para un tiempo de retardo de
;                                ;250mili-segundos.
Loop1     dec R7                  ;Se decrementa el registro R7
          jnz Loop1               ;Permanece en el ciclo hasta que R7=0.
          Ret                   ;Retorno de subrutina.

;Retardo de 1 segundo.
Ret1s     mov #0x04,R6            ;R6<-0x04
Loop2     call #Ret250ms         ;Llamado a subrutina de retardo de 250ms.
          dec R6                  ;Se decrementa R6.
          jnz Loop2               ;Si R6 no es cero permanece en el ciclo.
          Ret                   ;Retorno de subrutina.

```

```

;Retardo de 2 segundos.
Ret2s     mov #0x02,R5            ;R6<-0x04
Loop3     call #Ret1s             ;Llamado a subrutina de retardo de 250ms.
          dec R5                  ;Se decrementa R6.
          jnz Loop3               ;Si R6 no es cero permanece en el ciclo.
          ret

.sect ".reset"                ;MSP430 RESET Vector
.short RESET
.end

```

4. Hacer el reporte correspondiente a la práctica 2.

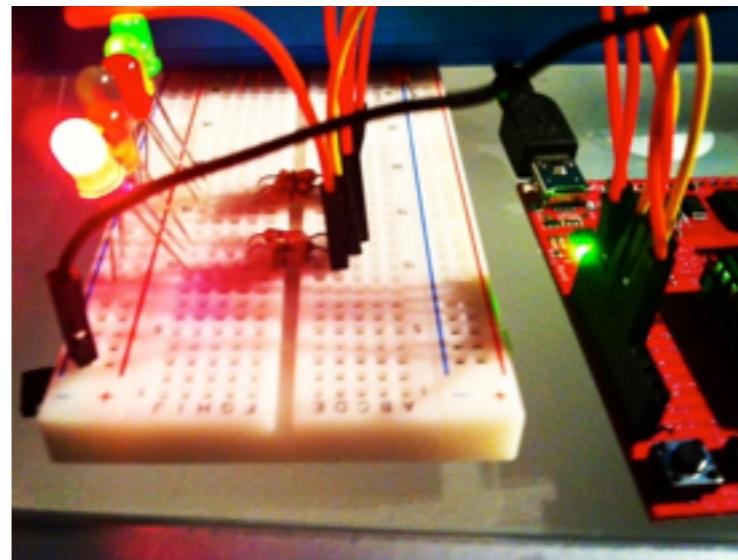


Figura 2.3. Conexión física de los LEDs y el microcontrolador

Práctica 3

Lectura de una entrada digital y configuración de resistencias internas pull-up, pull-down

OBJETIVOS

1. Aprender el funcionamiento y configuración de las entradas digitales de los puertos.
2. Conocer las configuraciones y conexiones al introducir un push-button, switch o sensor externo.
3. Aprender a configurar la resistencia pull-up o pull-down interna de los puertos.
4. Leer una entrada y dado el estado de ésta realizar distintas acciones.

MATERIALES

- 1 Microcontrolador MSP430G2553 Launch-Pad.
- 1 Placa protoboard.
- 2 LEDs rojos.
- 2 LEDs amarillos.
- 2 LEDs verdes.
- 6 resistencias de 220 o 330 ohms.
- 1 push-button o switch.
- 1 resistencia de 33k-ohms.

Una vez comprendido el uso de las salidas digitales de los puertos, el siguiente paso es aprender a trabajar con las entradas digitales, es por ello que en esta práctica se introduce un push-button externo (o switch). Se lee el estado de entrada producido por el push-button, y según esté oprimido o sin oprimir se realizan ciertas acciones. Nuevamente se hace uso de los “semáforos” que se utilizaron en la práctica anterior.

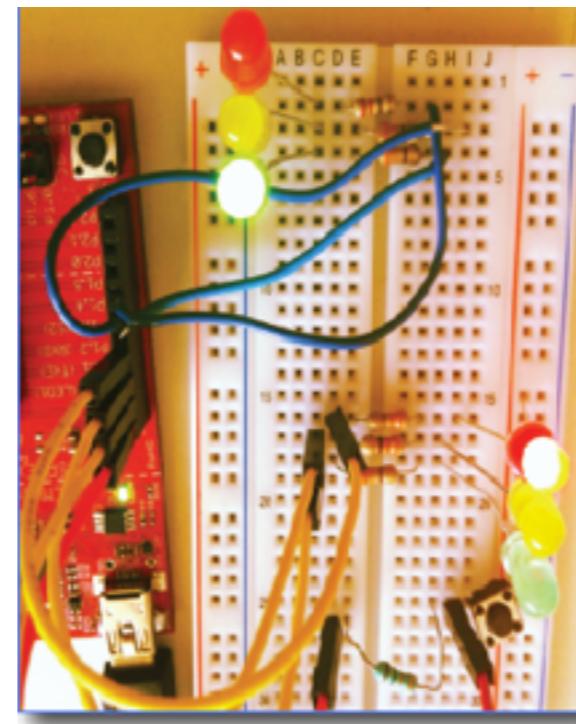


Figura 3.1. Componentes externos utilizados en la práctica 3 y su conexión.

Metodología

Lectura de una entrada

Primeramente el bit del puerto a utilizar se debe de configurar como entrada en el registro PxDIR, poniendo un cero en dicho registro según el bit del puerto (puerto 1 o puerto 2) que se quiera configurar como entrada.

Una vez que se haya realizado el programa y no se hayan encontrado errores, éste está listo para ejecutarse.

Cuando se ejecuta el programa es posible conocer el estado del puerto de entrada. En la ventana superior derecha del Code Composer se pueden checar, las variables, las expresiones y los registros, se activan los registros como se muestra en la figura 3.1.

Una vez, que se está en la pestaña de registros se busca Port_1_2, dentro de Port_1_2 se localiza P1IN y se despliegan todos los bits de P1IN como se muestra en la figura 3.2.

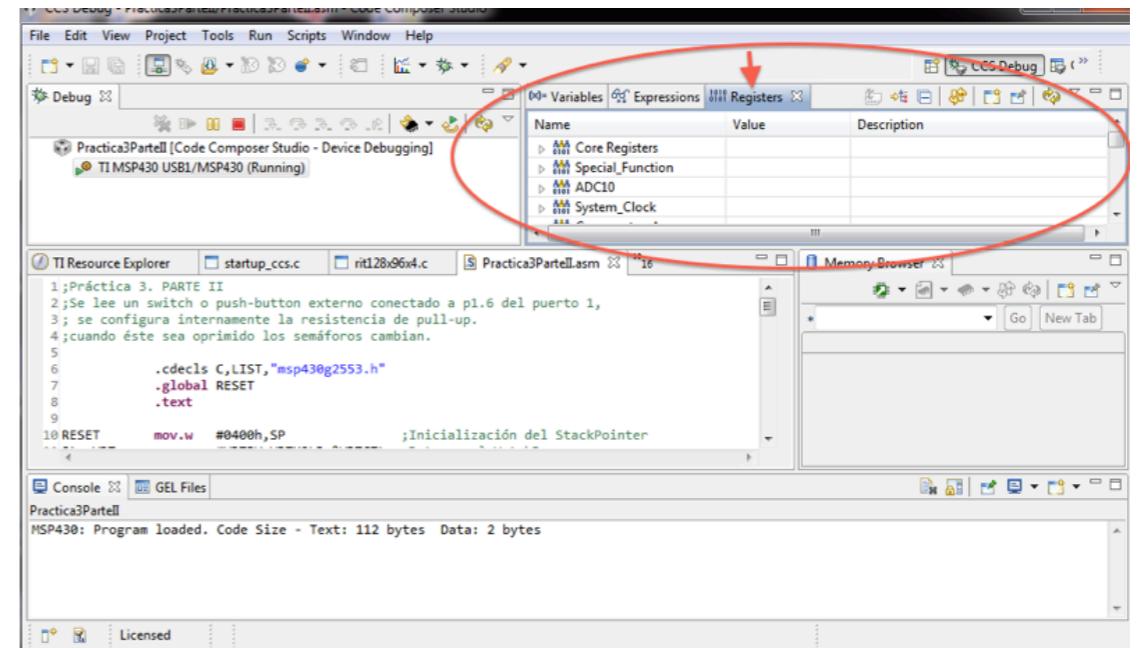


Figura 3.1. Lectura de una entrada.

Name	Value	Description
Comparator_A		
Flash		
Port_1_2		
P1IN	0x0C	Memory Mapped Register: Port 1 Input
P7	0	P7
P6	0	P6
P5	0	P5
P4	0	P4
P3	1	P3

Figura 3.2. Lectura de una entrada.

Si se está utilizando, por ejemplo el bit 6 del puerto 1 (P1.6), y su estado de entrada es un “uno lógico”, en el bit 6 del puerto 1, aparecerá un 1 como se muestra en la figura 3.3.

Name	Value	Description
Comparator_A		
Flash		
Port_1_2		
P1IN	0x4C	Memory Mapped Register: Port 1 Input
P7	0	P7
P6	1	P6
P5	0	P5
P4	0	P4
P3	1	P3

Figura 3.3. Lectura de entrada en “1” Lógico.

Si ahora el estado de la entrada cambia por un “cero lógico”, en el bit 6 del puerto 1, aparecerá un 0 como se muestra en la figura 3.4.

Name	Value	Description
Comparator_A		
Flash		
Port_1_2		
P1IN	0x0C	Memory Mapped Register: Port 1 Input
P7	0	P7
P6	0	P6
P5	0	P5
P4	0	P4
P3	1	P3

Figura 3.4. Lectura de entrada en “0” Lógico.

Configuración de la “resistencia interna” pull-up del microcontrolador.

La configuración básica de la resistencia de pull-up externa se muestra a continuación:

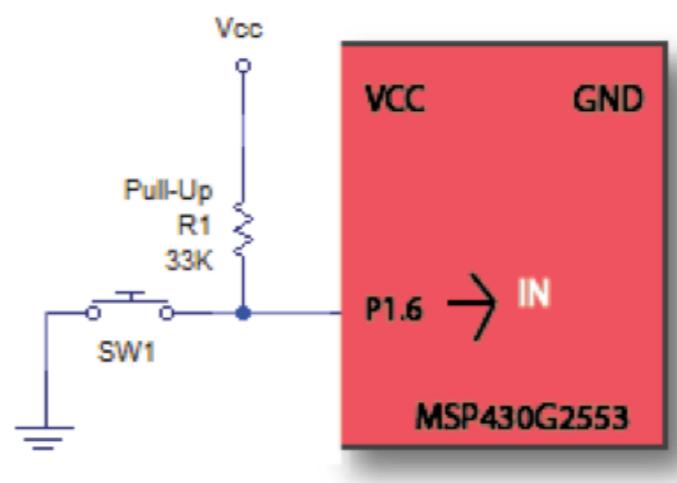


Figura 3.2.
Configuración de
resistencia pull-up
externa.

La resistencia de pull-up mantiene la entrada en uno lógico (Vcc) cuando el push-button no está oprimido (es decir, está abierto), cuando se oprime el push-button, se cierra éste y la entrada está en 0 lógico (Tierra). Por lo tanto la entrada se conoce como activa-bajo, ya que al oprimir el botón la entrada va de Vcc a Tierra, es decir, de 1 lógico a cero lógico.

Existe una corriente desperdiciada que va de la resistencia del pull-up a tierra cuando el push-button es presionado. Esta corriente se reduce teniendo una resistencia de pull-up grande (típicamente 33k-ohms), sin embargo el sistema se vuelve sensible al ruido.

Debido a que esta configuración es muy común, la mayoría de los microcontroladores tienen resistencias internas de pull-up, de este modo se reduce el número de componentes externos requeridos.

La configuración de la resistencia pull-up interna se muestra en la siguiente figura.

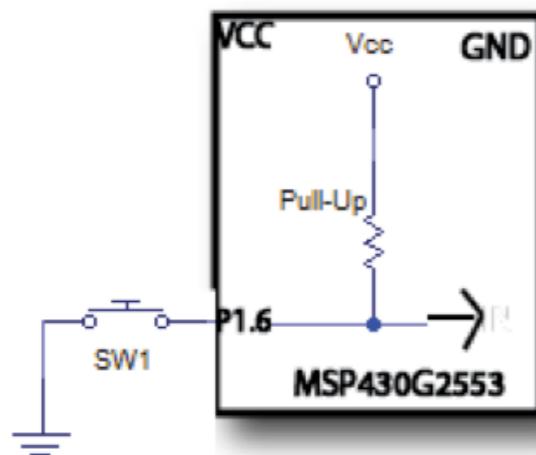
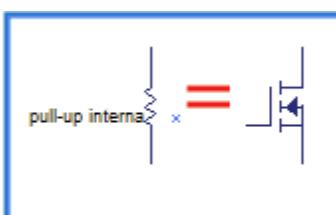


Figura 3.3. Configuración de resistencia pull-up interna.



NOTA: En realidad la resistencia pull-up o pull-down interna del microcontrolador es un transistor.

En el microcontrolador MSP430G2553 las resistencias internas de pull-up y pull-down pueden ser activadas configurando algunos bits en los registros PxREN, PxOUT y PxDIR como se muestra en tabla 3.1.

Tabla 3.1. Configuración de entrada/ salida de puertos.

PxDIR	PxREN	PxOUT	Configuración E/S
0	0	X	Entrada
0	1	0	Entrada configurada con Resistencia pull-down interna.
0	1	1	Entrada configurada con Resistencia pull-up interna.
1	X	X	Salida

Así, por ejemplo si se requiere evitar el uso de componentes externos (resistencias), se puede utilizar el puerto P1.6 del puerto 1 como entrada y configurar su “resistencia interna” como pull-up como se muestra a continuación.

En el Registro **P1DIR** se debe de poner un 0 en el bit 6. Por ejemplo, mov.b #00000000b,&P1DIR en el caso que todos los bits fuesen entradas.

En el registro **P1REN** se debe poner un 1 en el bit 6, por ejemplo, mov.b #01000000,&P1REN, en el caso que sólo el bit 6 se requiera con resistencia de pull-up interna.

Y finalmente cada que se utilice **P1OUT** no olvidar poner un uno en el bit 6, por ejemplo, mov.b #01000000,&P1OUT, en el caso de que sólo se requiera para el bit 6.

NOTA: En el caso que se utilicen los bits del puerto 1 sugeridos en el desarrollo de la práctica, es decir, P1.0, P1.1, P1.2, P1.3, P1.4, P1.5 como salidas y P1.6 como entrada es importante mencionar que se deben de retirar los jumpers conectados en J5 como se muestra en la figura 3.4. Ya que estos jumpers conectan a P1.0 con el LED1 y a P1.6 con el LED2, evitando cualquier tipo de interferencia.

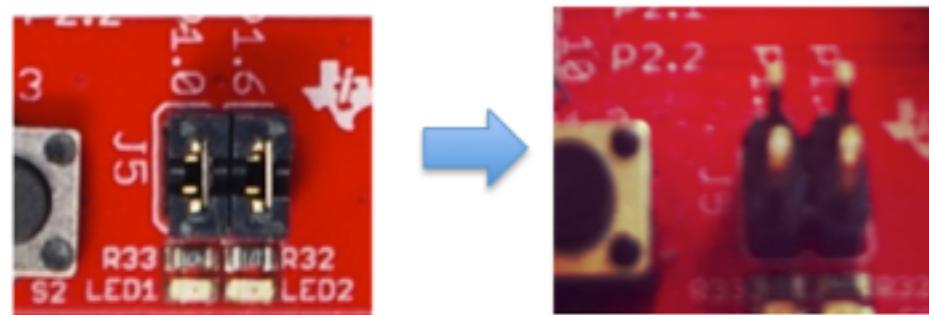


Figura 3.4. Diagrama ilustrativo que indica cuáles jumpers se deben de retirar.

Desarrollo de la práctica

PARTE I. Resistencia de pull-up externa.

1.1 Conectar los LEDs externos y push button externo con el microcontrolador como se muestra en el siguiente esquemático.

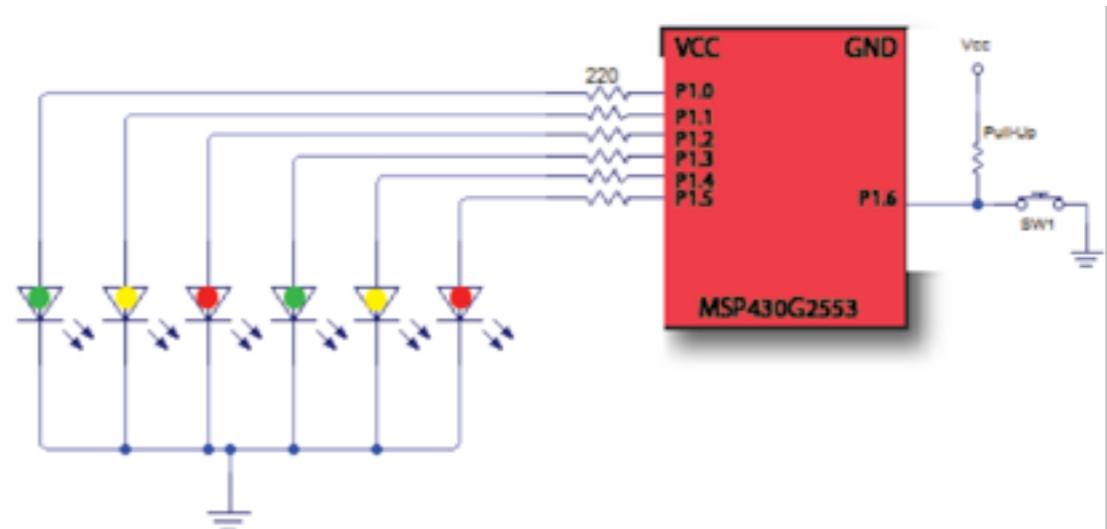


Figura 3.5.Conexión de los elementos externos con el microcontrolador.

1.2 Hacer el programa en código assembly. Dadas las siguientes indicaciones.

Inicialmente el **semáforo uno (principal)** estará en **verde** y el **semáforo dos** estará en **rojo**, y permanecerán así, al menos que el push-button se oprima. Cuando se presione el botón el **semáforo uno (principal)** pasará por el **amarillo**, posteriormente al **rojo** y después regresará a su estado inicial (**verde**), mientras que el **semáforo dos** pasará al **verde** y posteriormente al **amarillo** para regresar a su estado inicial (**rojo**). A continuación se muestran imágenes ilustrativas del desarrollo de la práctica.

Estado inicial:

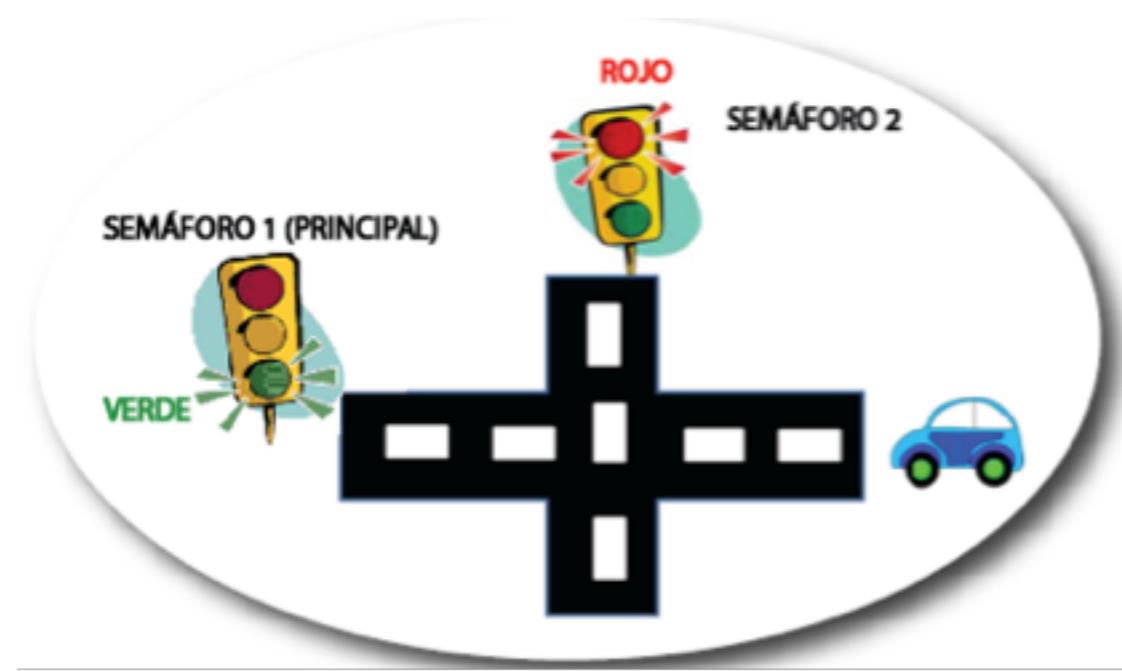


Figura 3.6. Esquemático ilustrativo del estado inicial.

Al oprimir el push-button o switch externo:

Los semáforos cambiarán como se muestra a continuación.

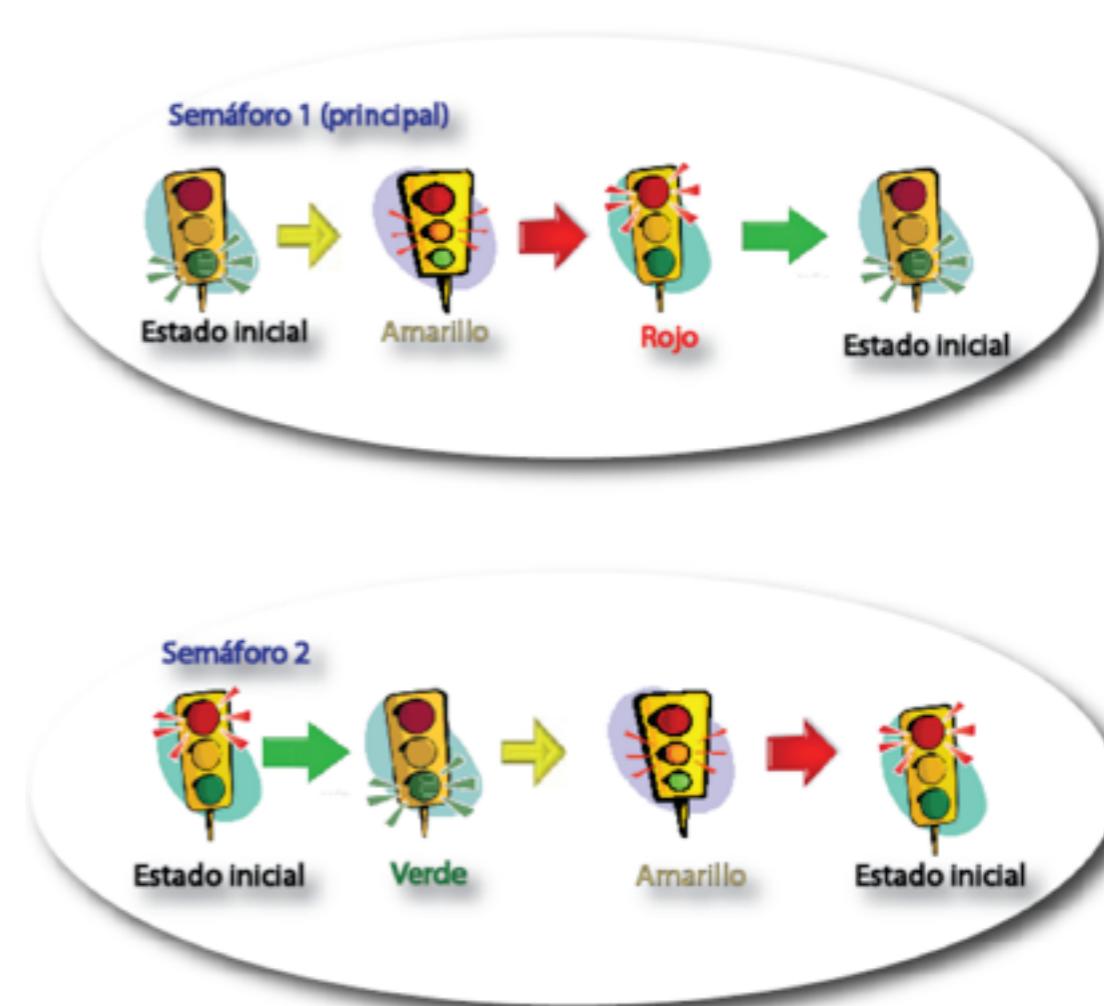


Figura 3.7. Esquemático ilustrativo de los estados de los semáforos al oprimir el push button o switch externo.

A continuación se muestran las tablas que corresponden a los estados de los semáforos, y pueden servir de referencia al realizar el programa.

Tabla 3.2. Estado inicial

PUERTO 1									
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	P1OUT (HEX)	Estado
SEMÁFORO 1				SEMÁFORO 2					
R1 A1 V1				R2 A2 V2					
0	0	0	0	1	1	0	0	0x0C	Inicial

Cuando se oprime el botón, los estados de los semáforos cambian como se muestra en la tabla 3.3.

Tabla 3.3. Estados del puerto 1 cuando se oprime el botón.

PUERTO 1										Estado
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	P1OUT (HEX)	Tiempo retardo	Estado
SEMÁFORO 1				SEMÁFORO 2						
R1 A1 V1				R2 A2 V2						
0	0	0	1	0	1	0	0	0x14	2 s	1
0	0	1	0	0	0	0	1	0x21	4s	2
0	0	1	0	0	0	1	0	0x22	2s	3
Regresar a Estado Inicial										

No olvidar configurar P1.0, P1.1, P1.2, P1.3, P1.4, P1.5 como salidas digitales y P1.6 como entrada digital.

1.3 Una vez, que se ha probado el correcto funcionamiento del programa, desconecte la resistencia externa de pull-up, oprima el push-button, y anote sus observaciones.

PARTE II. Resistencia de pull-up interna.

2.1 Una vez retirada la resistencia de pull-up externa, corroborar que la conexión sea como se muestra en la figura 3.8

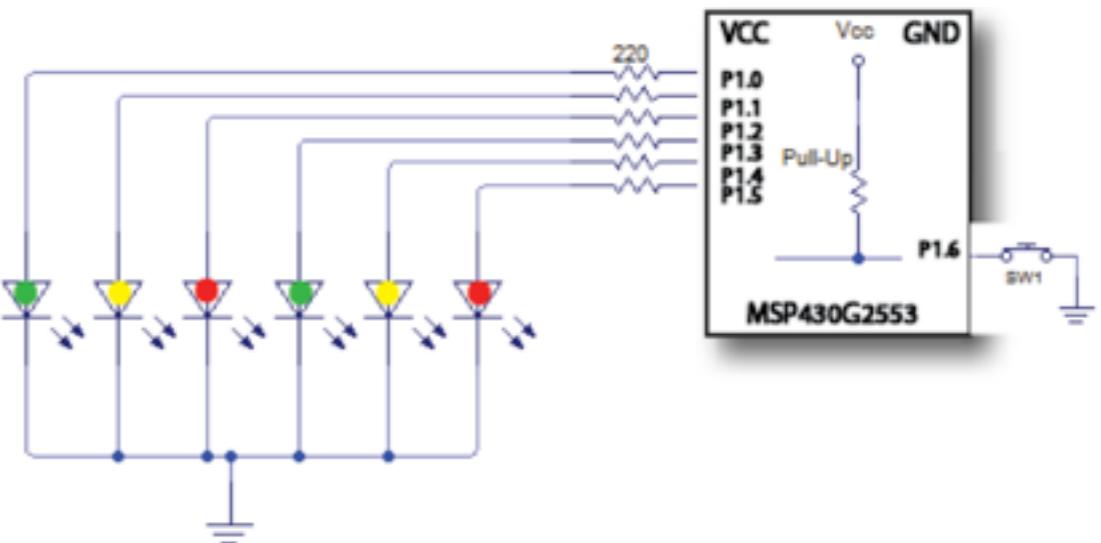


Figura 3.8. Conexión de los elementos externos con el microcontrolador, cuando se configura la resistencia de pull-up interna.

2.2 Modificar el programa anterior, configurando la resistencia de pull-up interna como se menciona en la sección de metodología. Es decir, en el Registro **P1DIR** del puerto 1 se debe de poner un 0 en el bit 6, los bits del 0-5 son 1's ya que son las salidas a los LEDs.

En el registro **P1REN** se debe poner un 1 en el bit 6. Y en **P1OUT** también se debe de poner un '1' en el bit 6 para que sea la resistencia de pull-up la que se configure.

Tabla 3.4. Estado Inicial, resistencia pull-up interna configurada.

PUERTO 1										
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	P1OUT (HEX)	Estado	
SEMÁFORO 1				SEMÁFORO 2						
				R1	A1	V1	R2	A2	V2	
0	1	0	0	1	1	0	0	0	0x4C	Inicial

Como se mencionó en la parte de metodología, es necesario en el registro P1OUT que el bit 6 siempre sea uno para permitir el correcto funcionamiento de la resistencia de pull-up interna. Tal como se muestra en la tabla 3.5.

Tabla 3.5. Estados del puerto uno, una vez que se presionó el botón, resistencia pull-up interna configurada.

PUERTO 1										
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	P1OUT (HEX)	Tiempo retardo	Estado
SEMÁFORO 1				SEMÁFORO 2						
				R1	A1	V1	R2	A2	V2	
0	1	0	1	0	1	0	0	0x54	2 s	1
0	1	1	0	0	0	0	1	0x61	4s	2
0	1	1	0	0	0	1	0	0x62	2s	3
Regresar al Estado Inicial										

2.3 Observa la diferencia entre la tabla 3.2 y 3.4 y la tabla 3.3 y 3.5.

2.4 Hacer reporte correspondiente a la práctica 3.

Práctica 4

Interrupciones de los puertos

OBJETIVOS

1. Introducirse en el ambiente de las interrupciones.
2. Conocer el funcionamiento y configuración de las interrupciones de los puertos.
3. Conocer la dirección de los vectores de interrupción de los puertos.
4. Conocer la prioridad de las interrupciones según su vector.

MATERIALES

1 Microcontrolador MSP430G2553 LaunchPad.

1 Placa protoboard.

2 LEDs rojos.

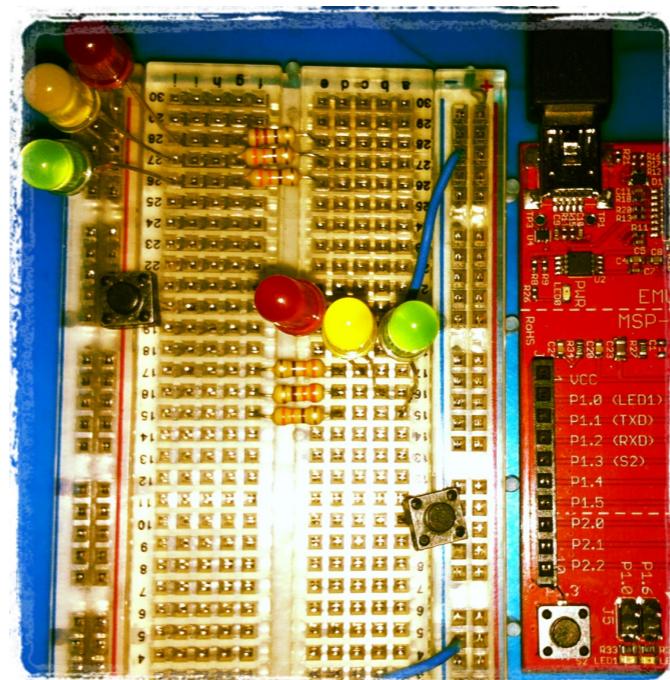
2 LEDs amarillos.

2 LEDs verdes.

6 resistencias de 220 o 330 ohms.

2 push-button (o switch).

Una vez que se han dominado las entradas y salidas digitales de los puertos, es momento de introducirnos en el tema de interrupciones, empezando por las interrupciones de los puertos.



Para ello, se volverán a utilizar los dos “semáforos” de LEDs utilizados en las prácticas anteriores y dos push-buttons externos. Los semáforos funcionarán normalmente, hasta que un push-button se oprima, y se genere una interrupción. Si se oprime el push-button o switch 1 (principal) hará que el semáforo 1 (principal) se ponga inmediatamente en verde y permanezca ahí por unos segundos más y el semáforo 2 en rojo.

Por otra parte si se oprime el push-

button o switch 2 hará que el semáforo 2 se ponga inmediatamente en verde permaneciendo ahí por unos segundos más y el semáforo 1 (principal) en rojo. Si se llegasen a oprimir los dos push-buttons al mismo tiempo, primero se efectuará la acción correspondiente al push-button 1 (principal) y posteriormente la acción correspondiente al push-button 2, de este modo se le da prioridad al semáforo principal.

Metodología

Las interrupciones son generalmente generadas por hardware, aunque también se pueden generar por software. Generalmente indican que un evento ha ocurrido y que requiere una pronta respuesta.

Cuando una interrupción ocurre el procesador se detiene de lo que estaba haciendo guardando cierta información (el contenido del Program Counter (PC) y del Status Register (SR)), para regresar después en donde se detuvo el programa una vez que se haya ejecutado la Rutina de Servicio de Interrupción (ISR (Interrupt Service Routine)).

Una **Rutina de Servicio de Interrupción (ISR)** es como una subrutina pero ésta es llamada por hardware (generalmente) en cualquier momento impredecible, en lugar de ser llamada por software de forma ordenada.

Las interrupciones pueden ser generadas por la mayoría de los módulos periféricos y algunos de la base del MCU como el generador del reloj. Cada interrupción tiene una bandera, la cual se activa cuando la condición de interrupción ocurre. Por ejemplo, el registro PxIFG indica en cual bit del puerto x ocurrió la interrupción.

Las interrupciones en este microcontrolador tienen dos funciones:

1. Indicar que un evento ha ocurrido y que se requiere una respuesta urgente.
2. Despertar al procesador (CPU) de un estado de bajo consumo o bien apagar el CPU, esta aplicación es muy importante particularmente en este microcontrolador, ya que una de sus principales características es precisamente su bajo consumo de potencia.

Una interrupción se puede utilizar para realizar tareas que tienen mayor prioridad que el código principal en un preciso momento. También una interrupción nos ayuda a evitar el famoso “polling” (preguntar por el estado de una entrada o por el valor de un registro infinitamente).

¿Qué pasa cuando una interrupción se ha generado?

Los siguientes pasos indican lo que sucede cuando una interrupción ocurre.

1. Si el CPU está activo, la instrucción en la que se encuentra se completa. Si el CPU está inactivo el MCLK se enciende, es decir, despierta el procesador.

- 2.** El PC (Program Counter), el cual está apuntado a la siguiente instrucción, se empuja al stack (se guarda).
- 3.** El Registro de estados (Status Register (SR)) también se empuja al stack (se guarda).
- 4.** Si muchas interrupciones están esperando, la interrupción con mayor prioridad es seleccionada.
- 5.** La Bandera de petición de interrupción se limpia automáticamente, sólo cuando los vectores tiene una sola fuente, cuando no es así es necesario que en la ISR se limpie esta bandera.
- 6.** El Registro de Estados (Status Register (SR)) se limpia, lo cual tiene dos efectos: Primero, las futuras interrupciones “maskable” son deshabilitadas porque el bit de GIE es limpiado, las interrupciones non-maskable permanecen activas. Segundo, se termina con el modo de bajo consumo de potencia.
- 7.** El vector de Interrupción es cargado en el Program Counter (PC) y el CPU empieza a ejecutar la Rutina de servicio de Interrupción (ISR) en dicha dirección.

Esta secuencia consume seis ciclos de reloj, en el MSP430G2553 antes de que la ISR comience.

Configuración de las Interrupciones de los Puertos

El MSP430G2553 cuenta con dos puertos (Puerto 1 con 8 bits, y Puerto 2 con 6 bits), por lo que se cuentan con 10 fuentes de interrupción.

Cada bit es configurable de manera independiente, si se desea que sólo un pin del puerto 1 admita interrupciones se puede hacer, o bien también se puede que todos los bits del puerto 1 y puerto 2 admitan interrupciones.

Para configurar una interrupción a un bit del puerto es necesario que éste esté configurado como entrada, así como habilitar el bit de las interrupciones globales (GIE) y configurar las interrupciones locales.

Además de los registros de puerto que se han manejado anteriormente: PxIN, PxOUT, PxREN, PxDIR y PxSEL, hay tres registros más encargados de la configuración de las interrupciones.

PxIFG, en este registro se activa la bandera de petición de interrupción según el bit donde se generó dicha interrupción.

PxIES, en este registro se selecciona el flanco (de bajada o de subida) por el cual la interrupción se generará.

PxIE, en este registro se habilitan las interrupciones de cada bit del puerto (puerto 1 ó puerto 2).

Es importante mencionar el hecho de que las interrupciones se generan mediante una transición de niveles lógicos, es decir, de 0 a 1 (flanco de subida) o bien de 1 a 0 (flanco de bajada), no con niveles estáticos.

Una vez que se ha configurado el bit (en el cual queremos que se habilite su interrupción), como entrada, así como su resistencia “interna” de pull-up o pull-down en caso de utilizar un switch o push-button externo, se configura el flanco por el cual la interrupción será generada, en el registro PxIES.

Tabla 4.1. Registro PxIES

PxIES (P1IES ó P2IES) Interrupt Edge Select Registers (Registros de Selección de Flanco de Interrupción)							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
BIT7-BIT0 0= Interrupción generada por flanco de subida (transición de “0” lógico a “1” lógico). 1= Interrupción se genera por flanco de bajada (transición de “1” lógico a “0” lógico).							

Cuando se genera un cambio de estado bajo (0) a estado alto (1) se genera un flanco de subida, por el contrario cuando el cambio se genera de estado alto (1) a estado bajo (0), se genera un flanco de bajada.



Si se configura, por ejemplo, la entrada con la resistencia interna de pull-up, el estado de entrada del bit siempre será 1 lógico y al presionar el switch externo se producirá un 0 lógico, por lo que la interrupción se generará por flanco de bajada.

Una vez que se configuró este registro, es importante antes de empezar con el programa principal borrar todas las banderas de petición de interrupción del puerto o de los puertos.

Una **bandera de interrupción** es un bit que indica que existe interrupción pendiente (1) o no existe interrupción pendiente (0). En el caso de las interrupciones de los puertos el registro PxIFG (P1IFG ó P2IFG) tiene la bandera de petición de interrupción de cada bit de cada puerto.

Tabla 4.2. Registro PxIFG

PxIFG (P1IFG ó P2IFG) Interrupt Flag Registers (Registros de Banderas de Interrupción)							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
BIT7-BIT0 0= No hay interrupción pendiente.							
1= Hay interrupción pendiente.							

Antes de permitir cualquier interrupción por primera vez es necesario borrar la bandera correspondiente al bit que se configurará con interrupción , ya que en ocasiones suele estar en 1 y nos produciría un comportamiento no adecuado, como mandar a llamar la ISR (Rutina de Servicio de Interrupción) en un momento en el cual no es requerida. Este registro se puede limpiar con la instrucción clr.b.

clr.b &P1IFG o

clr.b &P2IFG

Cabe mencionar que las banderas de petición de Interrupción de los puertos al ser múltiples no son borradas automáticamente cuando la interrupción ocurre, sino que hay que borrarlas por software en la rutina de servicio de interrupción (ISR) en el registro PxIFG de lo contrario, el microcontrolador no tendrá la respuesta deseada.

Por ejemplo, si ocurrió una interrupción en el bit 6 del Puerto 1 (P1.6) una vez que se está en la rutina de servicio de Interrupción (ISR) se borra esta bandera con la instrucción bic.b, sino se borra esta bandera, permanecerá siempre en la rutina de servicio de interrupción (ISR).

bic.b #BIT6,&P1IFG ó

bic.b #01000000b,&P1IFG

Una vez que se ha configurado el flanco (de subida o bajada) por el cual ocurrirá la interrupción y se ha limpiado la bandera al principio del programa sólo resta habilitar la interrupción del bit que se desea utilizar, mediante el registro PxIE (P1IE ó P2IE según el puerto con el que se vaya a trabajar) y hacer la Rutina de Servicio de Interrupción que se quiere ejecutar cuando la interrupción ocurra.

El registro PxIE sirve para seleccionar los bits que generarán interrupciones y los que no las generarán.

Tabla 4.3. Registro PxIE

PxIE (P1IE ó P2IE) Interrupt Enable Registers							
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
BIT7-BIT0 0= Interrupción inhabilitada.							
1= Interrupción habilitada.							

Por ejemplo, si se desea que los bits 6 y 7 del puerto 1 generen interrupciones, se pueden habilitar con la instrucción bis.b.

bis.b #BIT7+BIT6,&P1IE ó bis.b #11000000b,&P1IE

Una vez que se han habilitado las interrupciones locales de los puertos u otros periféricos, es muy importante, habilitar las interrupciones globales, esto se puede hacer con dos instrucciones:

EINT ó bis #GIE,SR

Vectores de las Interrupciones de los Puertos

Una vez que se han configurado las interrupciones y son permitidas, el siguiente paso es definir su vector correspondiente, para posteriormente definir la rutina de servicio de interrupción en la dirección del vector. La siguiente tabla proporcionada por el fabricante muestra los vectores correspondientes a cada interrupción así como su prioridad, en un cuadrado rojo se encuentran los vectores de interrupción para los puertos 1 y 2, es decir, la dirección donde se efectúa la Rutina de Servicio de Interrupción (ISR) así como su prioridad.

Tabla 4.4 Fuente de interrupción, Banderas y Vectores.

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range ⁽¹⁾	PORIFG RSTIFG WDTIFG KEYV ⁽²⁾	Reset	0FFF Eh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG ⁽²⁾⁽³⁾	(non)-maskable (non)-maskable (non)-maskable	0FFF Ch	30
Timer1_A3	TA1CCR0 CCIFG ⁽⁴⁾	maskable	0FFF Ah	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG ⁽²⁾⁽⁴⁾	maskable	0FFF 8h	28
Comparator_A+	CAIFG ⁽⁴⁾	maskable	0FFF 6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF 4h	26
Timer0_A3	TA0CCR0 CCIFG ⁽⁴⁾	maskable	0FFF 2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG ⁽⁵⁾⁽⁴⁾	maskable	0FFF 0h	24
USCI_A0/USCI_B0 receive USCI_B0 I2C status	UCA0RXIFG, UCB0R0IFG ⁽²⁾⁽⁵⁾	maskable	0FFE Eh	23
USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	UCA0TXIFG, UCB0TXIFG ⁽²⁾⁽⁶⁾	maskable	0FFE Ch	22
ADC10 (MSP430G2x53 only)	ADC10IFG ⁽⁶⁾	maskable	0FFE Ah	21
			0FFE 8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE 6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE 4h	18
			0FFE 2h	17
			0FFE 0h	16

Esta tabla se puede encontrar en la página once, del manual de usuario proporcionado por Texas Instruments.

<http://www.ti.com/lit/ds/symlink/msp430g2553.pdf>

Se observa que para el Puerto 1 el vector de interrupción está en la dirección 0FFE4h, y para el Puerto 2 en la dirección 0FFE6h. Además de que tiene una mayor prioridad la interrupción del Puerto 2 que la del Puerto 1, es decir, dado que ocurra una interrupción en el Puerto 2 al mismo tiempo que en el Puerto 1, primero se atenderá la interrupción generada en el Puerto 2 y posteriormente la generada en el Puerto 1.

Los vectores de interrupción se definen después del vector de RESET al final del programa en código assembly. Como se muestra a continuación.

```
;***** Vectores de interrupción *****
;***** .sect ".reset" ; MSP430 RESET Vector
;***** .short RESET
;***** .sect ".int02"
;***** .short ISR_P1
;***** .sect ".int03"
;***** .short ISR_P2
;***** .end
```

Se observa que en lugar de poner la dirección del vector de interrupción del Puerto 1 (0FFE4h) se pone “.int02”, y del puerto 2 (0FFE6h) se pone “.int03”, esto se realiza de esta manera en lenguaje assembly.

Desarrollo de la práctica

- Realiza la conexión de los LEDs y push-buttons externos como se muestra en el diagrama 4.1.

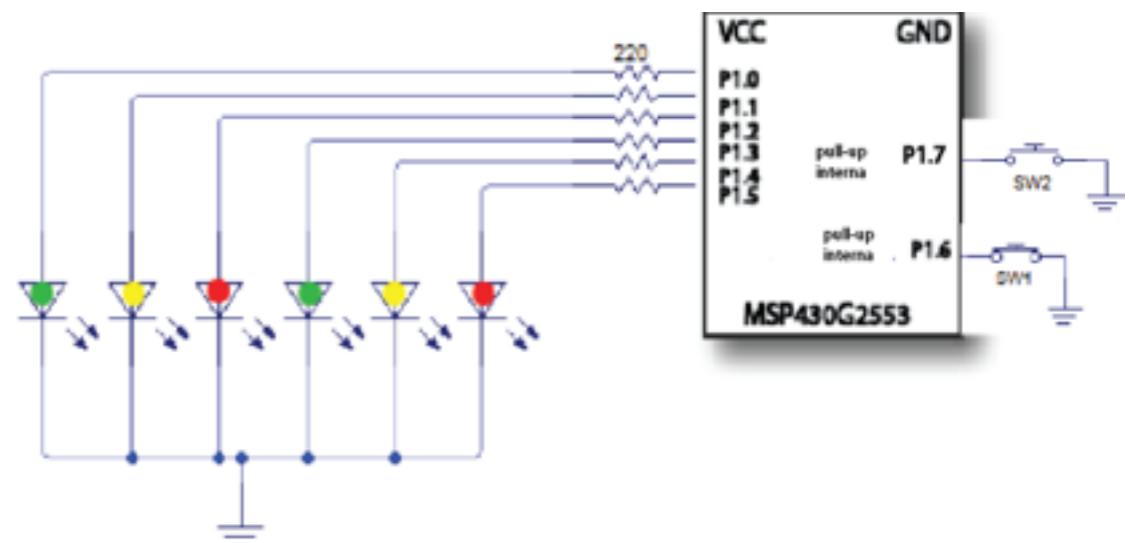


Figura 4.1. Diagrama de conexión con los elementos externos.

- El programa en código Assembly se realizó dadas las siguientes indicaciones:

- Se utilizaron los bits 5-0 del puerto 1 para los LEDs de los dos “semáforos”.
- El bit 6 del puerto 1 se configura como entrada con resistencia interna de pull-up y se habilita su interrupción, ya que funge como el switch 1 o push-button 1.

-- El bit 7 del puerto 1 también se configura también como entrada con resistencia interna de pull-up y se habilita su interrupción, ya que funge como el switch 2 o push-button 2.

-- Inicialmente los semáforos funcionan normalmente como se muestra en la Tabla 4.1.

Tabla 4.1. Configuración y sincronización de los semáforos.

PUERTO 1											
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	P1OUT (HEX)	Tiempo retardo	Estado	
SW2	SW1	SEMÁFORO 1			SEMÁFORO 2						
		R1	A1	V1	R2	A2	V2				
1	1	1	0	0	0	0	1	0xE1	2 s	1	
1	1	1	0	0	0	1	0	0xE2	1s	2	
1	1	0	0	1	1	0	0	0xCC	2s	3	
1	1	0	1	0	1	0	0	0xD4	1s	4	

-- Cuando el push-button 1 (SW1) se oprime el puerto 1 pasa inmediatamente al Estado 3, es decir el semáforo 1 se pone en verde y el semáforo 2 en rojo por 3 segundos.

Tabla 4.2. Estado del puerto 1 cuando el SW1 se oprime.

PUERTO 1											
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	P1OUT (HEX)	Tiempo retardo	Estado	
SW2	SW1	SEMÁFORO 1			SEMÁFORO 2						
		R1	A1	V1	R2	A2	V2				
1	1	0	0	1	1	0	0	0xCC	3s	3	

-- Cuando el push-button 2 (SW2) se oprime el puerto 1 pasa de manera inmediata al Estado 1, es decir el semáforo 1 se pone en rojo y el semáforo 2 en verde por 3 segundos.

Tabla 4.3. Estado del puerto 1 cuando el SW2 se oprime.

PUERTO 1											
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	P1OUT (HEX)	Tiempo retardo	Estado	
SW2	SW1	SEMÁFORO 1			SEMÁFORO 2						
		R1	A1	V1	R2	A2	V2				
1	1	1	0	0	0	0	1	0xE1	0	0	1

3. A continuación se detalla paso por paso el programa efectuado.

-- Primeramente se pone la cabecera, en donde se cargan los archivos del microcontrolador con el que se trabaja, se inicializa el stack pointer (SP) y se detiene al perro guardián.

```
.cdecls C,LIST,"msp430g2553.h"
.global RESET
.text
RESET    mov.w   #0400h,SP           ;Inicialización del StackPointer
StopWDT  mov.w   #WDTPW+WDTHOLD,&WDTCTL ;Detener el WatchDog
```

-- Después se configura el puerto 1. Los bits 5-0 del puerto 1 se configuran como salidas y los bits 7 y 6 del puerto 1 como entradas .

Además a estos bits de entrada se les configuran sus resistencias internas de pull-up (porque se utilizan push-buttons (o switches externos), y se habilitan sus interrupciones por flanco de bajada. En este paso también se habilitan las interrupciones globalmente.

```
*****
        CONFIGURACIÓN DE PUERTO 1
*****
    mov.b #00111111b,&P1DIR      ;P1.0,P1.1,P1.2,P1.3,P1.4,P1.5 como
                                ;salidas, P1.6 y P1.7 como entradas.
    mov.b #11000000b,&P1OUT      ;Configuración para resistencias
                                ;internas de pull-up, limpia salidas.
    bis.b #11000000b,&P1REN      ;Configuración de resistencia pull-up
                                ;interna de P1.6 y P1.7
    bis.b #BIT7+BIT6,&P1IES      ;Interrupciones en P1.6 y P1.7
                                ;por flanco de bajada.
    bic.b #BIT7+BIT6,&P1IFG      ;Limpia banderas de interrupción del
                                ;puerto 1.
    bis.b #BIT7+BIT6,&P1IE      ;Habilita las Interrupciones de
                                ;puerto en P1.6 y P1.7
    mov    #GIE,SR              ;Se habilitan interrupciones globales.
```

-- Enseguida de la configuración del puerto 1 se escribe el programa principal, en el cual los semáforos funcionan normalmente de acuerdo a la tabla 4.1.

```
*****
        PROGRAMA PRINCIPAL
*****
Estado1  mov    #0xE1,&P1OUT      ;Estado 1.
        call   #Ret2s
        ;Llama a subrutina de retardo de 2s.
Estado2  mov    #0xE2,P1OUT      ;Estado 2.
        call   #Ret1s
        ;Llama a subrutina de retardo de 1s.
Estado3  mov    #0xCC,&P1OUT      ;Estado 3.
        call   #Ret2s
        ;Llama a subrutina de retardo de 2s.
Estado4  mov    #0xD4,&P1OUT      ;Estado 4
        call   #Ret1s
        ;Llama a subrutina de retardo de 1s.
        jmp   Estado1
        ;Brinca a etiqueta Semaforo para
        ;regresar al Estado 1.
```

-- Se recomienda que antes de la Rutina de Servicio de Interrupción (ISR) se pongan las subrutas. A continuación se tienen las subrutas de retardo vistas en las prácticas anteriores.

```
;-----
;          SUBRUTINAS DE RETARDO
;-----

;Retardo de 250 mili-segundos.
Ret250ms  mov    #0xf41F,R7
          ;Data=0xf41F para un tiempo de retardo de
          ;250mili-segundos.
          ;Se decrementa el registro R7
          ;Permanece en el ciclo hasta que R7 sea cero
          ;Retorno de subrutina.

Loop1     dec    R7
          jnz   Loop1
          ret

;Retardo de 1 segundo.
Ret1s     mov    #0x04,R6
Loop2     call   #Ret250ms
          dec    R6
          jnz   Loop2
          ret

;Retardo de 2 segundos.
Ret2s     mov    #0x02,R5
Loop3     call   #Ret1s
          dec    R5
          jnz   Loop3
          ret

;R6<-0x04
;Llamado a subrutina de retardo de 250ms.
;Se decrementa R6.
;Si R6 no es cero permanece en la subrutina.

;R6<-0x04
;Llamado a subrutina de retardo de 250ms.
;Se decrementa R6.
;Si R6 no es cero permanece en la subrutina.
```

-- Posteriormente se encuentra la Rutina de Servicio de Interrupción (ISR), en la cual primeramente se pregunta por cual “switch (o push-button)” sucedió la interrupción, según el resultado realiza las acciones para el switch 1 o para el switch 2, se incluye un retardo de 3 segundos dentro de la Rutina de Servicio de Interrupción, ya que no se recomienda mandar a llamar otras subrutas estando en la ISR porque el programa no funcionará adecuadamente. Se observa que se borran por software las banderas de petición de interrupción utilizadas.

Finalmente se colocan los vectores de reset y de la interrupción del puerto 1 (.int02 localizado en la dirección 0xFFE4).

```
;*****
;          Vectores de interrupción
;*****
.sect    ".reset"           ; MSP430 RESET Vector
.short   RESET

.sect    ".int02"            ;Vector de Interrupción Puerto 1
.short   ISR_P1              ;ISR_P1 nombre de la rutina de
                             ;servicio de interrupción (ISR).

.end
```

4. A continuación se muestra el código del programa completo, cópialo y ejecútalo.

```
;      Laboratorio de microprocesadores y microcontroladores
;      PRÁCTICA 4. Interrupciones del Puerto

.cdecls C,LIST,"msp430g2553.h"
.global RESET
.text

RESET    mov.w  #8400h,SP      ;Inicialización del StackPointer
StopWDT  mov.w  #WDTPW+WDTHOLD,&WDTCTL ;Detener el WatchDog
;*****
;          CONFIGURACIÓN DE PUERTO 1
;*****
mov.b  #00111111b,&P1DIR ;P1.0,P1.1,P1.2,P1.3,P1.4,P1.5 como
                           ;salidas, P1.6 y P1.7 como entradas.
mov.b  #11000000b,&P1OUT ;Configuración para resistencias
                           ;internas de pull-up, limpia salidas.
bis.b  #11000000b,&P1REN ;Configuración de resistencia pull-up
                           ;interna de P1.6 y P1.7
bis.b  #BIT7+BIT6,&P1IES ;Interrupciones en P1.6 y P1.
                           ;por flanco de bajada.
bic.b  #BIT7+BIT6,&P1IFG ;Limpia banderas de interrupción del
                           ;puerto 1.
bis.b  #BIT7+BIT6,&P1IE  ;Habilita las Interrupciones de
                           ;puerto en P1.6 y P1.7
mov     #GIE,SR             ;Se habilitan interrupciones globales.
```



```
;*****
;          PROGRAMA PRINCIPAL
;*****

Estado1  mov   #0xE1,&P1OUT      ;Estado 1.
         call  #Ret2s        ;Llama a subrutina de retardo de 2s.
Estado2  mov   #0xE2,P1OUT      ;Estado 2.
         call  #Ret1s        ;Llama a subrutina de retardo de 1s.
Estado3  mov   #0xCC,&P1OUT      ;Estado 3.
         call  #Ret2s        ;Llama a subrutina de retardo de 2s.
Estado4  mov   #0xD4,&P1OUT      ;Estado 4
         call  #Ret1s        ;Llama a subrutina de retardo de 1s.
         jmp   Estado1       ;Brinca a etiqueta Semaforo para
                           ;regresar al Estado 1.

;*****
;          SUBRUTINAS DE RETARDO
;*****
```

```
;Retardo de 250 mili-segundos.
Ret250ms mov   #0xf41F,R7      ;Data=0xf41F para un tiempo de retardo de
                           ;250mili-segundos.
Loop1    dec   R7             ;Se decrementa el registro R7
         jnz   Loop1        ;Permanece en el ciclo hasta que R7=0.
         Ret
```

```
;Retardo de 1 segundo.
Ret1s    mov   #0x04,R6        ;R6<-0x04
Loop2    call  #Ret250ms     ;Llamado a subrutina de retardo de 250ms.
         dec   R6             ;Se decrementa R6.
         jnz   Loop2        ;Si R6 no es cero permanece en el ciclo.
         ret
```

```
;Retardo de 2 segundos.
Ret2s    mov   #0x02,R5        ;R6<-0x04
Loop3    call  #Ret1s        ;Llamado a subrutina de retardo de 250ms.
         dec   R5             ;Se decrementa R6.
         jnz   Loop3        ;Si R6 no es cero permanece en el ciclo.
         ret
```

```

;***** Rutina de servicio de Interrupción (ISR) *****
;
;***** ISR_P1 *****

ISR_P1    and.w #0x00FF,&P1IFG      ;Se enmascara sólo a 8 bits.
          bit.b #BIT6,&P1IFG      ;¿Interrupción por push-button 1?
          ;Mayor prioridad para switch 1.

          jnz  switch_1           ;Si lo es brinca a etiqueta switch_1.
          bit.b #BIT7,&P1IFG      ;¿Interrupción por push-button 2?
          jnz  switch_2           ;Si lo es brinca a etiqueta switch_2.
          bic.b #BIT7+BIT6,&P1IFG   ;Limpia banderas de interrupción
          jmp  Fin                 ;Sino suceden ninguna de las condiciones
                           ;anteriores, sale de la ISR.

switch_1   bic.b #BIT6,&P1IFG      ;Limpia Bandera de interrupción de P1.6
          mov   #0xCC,&P1OUT       ;Se pone en verde semáforo 1 y en rojo
                           ;semáforo 2.

Ret3s     mov   #0x12,R10        ;Retardo de 3 segundos.
Ret250ms2  mov   #0xf41F,R7        ;Data=0xf41F para un tiempo de retardo de
                           ;250mili-segundos.

Loop12    dec   R7             ;Se decrementa el registro R7
          jnz  Loop12
          dec   R10
          jnz  Ret250ms2         ;Permanece en el ciclo hasta que R10=0.
          jmp  Fin
          bic.b #BIT7,&P1IFG      ;Limpia bandera de interrupción de P1.7.
          mov   #0xE1,&P1OUT       ;Se pone e verde semáforo 1 y en rojo
                           ;semáforo 2.

          jmp  Ret3s              ;Retardo de 3 segundos.
          reti                     ;Retorno de interrupción.

;***** Vectores de interrupción *****
;

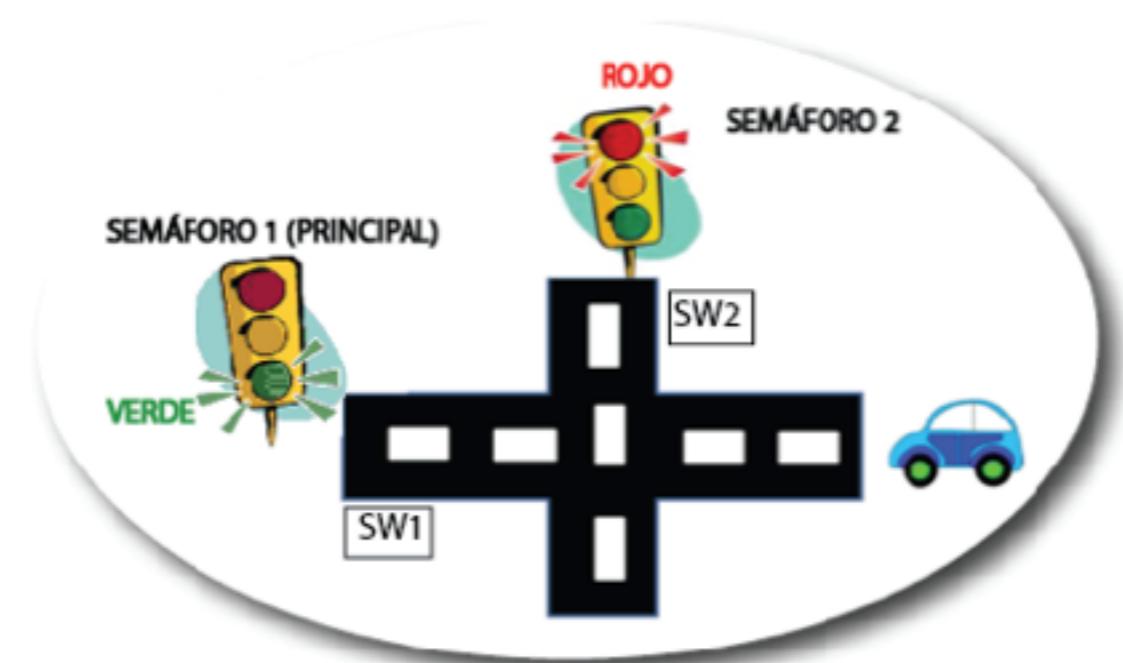
.sect ".reset"           ; MSP430 RESET Vector
.short RESET

.sect ".int02"            ;Vector de Interrupción Puerto 1
.short ISR_P1             ;ISR_P1 nombre de la rutina de
                           ;servicio de interrupción (ISR).

.end

```

5. Realizar reporte correspondiente a la práctica 4.



Práctica 5

Convertido Analógico-Digital (ADC10)

OBJETIVOS

1. Aprender los conceptos básicos del Convertidor Analógico-Digital.
2. Manejar los registros del ADC10 con los que se trabaja.
3. Conocer el funcionamiento y configuración del Convertidor Analógico-Digital.
4. Crear una aplicación utilizando el Convertidor Analógico-Digital.

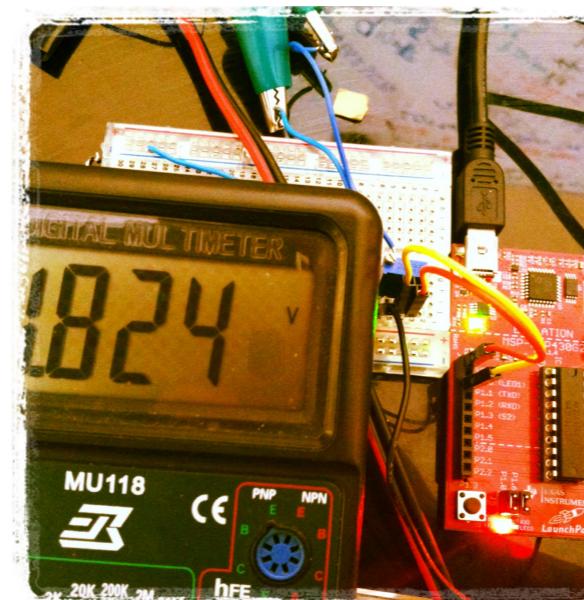
MATERIALES

- 1 Microcontrolador MSP430G2553.
- 1 Trimpot de 10k-ohms.*
- 4 LEDs.
- 4 Resistencias de 220 ó 330 ohms.

*Trimpot: Es una resistencia variable de precisión, ideal para pruebas con convertidores A/D. Su ajuste se realiza con un destornillador.

Se ha trabajado ya anteriormente con entradas y salidas digitales, así como interrupciones y configuración de resistencias internas de pull-up y pull-down del microcontrolador.

Un periférico muy importante en cualquier microcontrolador es el Convertidor Analógico-Digital (ADC). Por lo que en esta práctica 5 se abordará el tema, analizando los principales conceptos de este convertidor.



En esta práctica mediante un potenciómetro de precisión (TRIMPOT) se muestra el funcionamiento del convertidor analógico-digital. Este trimpot se conecta a una entrada (configurada como analógica) del microcontrolador, cuando la entrada tiene un valor superior a 0.5Vcc el LED rojo conectado al bit 0 del puerto 1 (P1.0) enciende y el LED verde conectado al bit 7 del puerto 1 (P1.7) se apaga, y cuando el valor es inferior a 0.5Vcc el LED verde conectado al pin P1.7 enciende. Y el LED rojo se apaga.

Metodología

¿Qué es un Trimpot?

Un trimpot es un potenciómetro de ajuste. El cual a diferencia de los potenciómetros comunes tiene una mayor estabilidad y precisión ya que tiene un número definido de vueltas. Y para variar su valor resistivo se necesita un desarmador pequeño o una perilla que en algunos viene incluida.

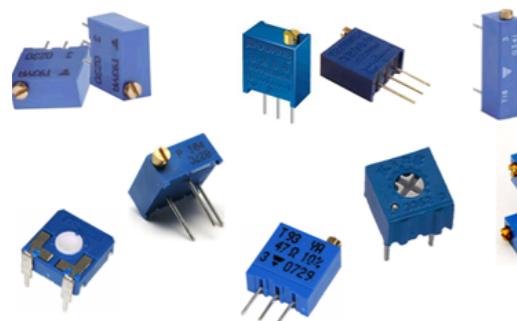


Figura 5.1. Trimpots.

Aspectos Generales del Convertidor Analógico-Digital (ADC)

El objetivo de un ADC es convertir una entrada analógica (como un voltaje en función del tiempo), en un valor binario de modo que el procesador pueda manejarla. La entrada al ser analógica es una función continua en el tiempo $V(t)$ lo que significa que el voltaje V puede tener cualquier valor permitido dentro de un rango y cambiar en cualquier momento como función del tiempo.

En contraste la salida del convertidor $V[n]$ es una secuencia de valores binarios. Normalmente la entrada es muestreada en intervalos regulares de tiempo T_s , por lo que su naturaleza continua se ve afectada. A continuación se presenta la definición de conceptos importantes relacionados con el ADC.

Exactitud. Este término hace alusión a qué tan cerca una medición está de su valor “real”, es decir el producido por un sistema ideal. Este concepto es fácil de definir mas no de medir.

Resolución o precisión. Este concepto se refiere al número de distintos valores de salida que una medición puede dar. Este término también se puede emplear para referirse a un cambio en la entrada que corresponde al mínimo cambio en la salida, 1bit.

El Convertido analógico-digital (ADC) con el que cuenta el MSP430G2553 es de 10 bits, lo cual significa que su salida es un valor binario de 10 bits, el cual puede ser representado por

$$2^{10} = 1024 \text{ valores distintos.}$$

También es necesario conocer el rango del voltaje de entrada para determinar la resolución de dicha entrada.

Por ejemplo, suponiendo que el rango es desde 0 a Escala plena, y el Voltaje de escala plena es 3 Volts. Entonces un cambio de un bit en la salida corresponde a el cambio de $(3V)/1024=3mV$ en la entrada.

Esto se conoce como Bit Menos Significativo (LSB por sus iniciales en inglés) y es otra forma de definir resolución. Por lo que se puede decir que el ADC convierte su entrada con una precisión de 3mV.

La conversión de una entrada continua a una salida discreta se conoce como cuantización. Para conocer el valor de la entrada de voltaje en bits, es decir, para saber el valor de la salida discretizada, dada una entrada continua, se defina la Función de transferencia de un N-Bit ADC .

$$N_{ADC} = \text{nint}\left(2^N \frac{V_{IN}}{V_{FS}}\right)$$

La función, nint() da el entero más cercano de su argumento.

En el caso del convertidor ADC que maneja este microcontrolador N es 10 bits, así por ejemplo si el voltaje de entrada es igual a 1.2 Volts, y el voltaje a plena escala es 3V, la entrada discretizada en bits es:

$$N_{ADC} = \text{nint}\left(2^{10} \frac{1.2V}{3V}\right) = \text{nint}(409.6) = 410\text{decimal} = 199\text{hexadecimal}$$

Otro elemento a destacar del convertidor A/D que maneja este microcontrolador es que La conversión puede ser “disparada” (trigger) por software o por el timer A.

Características del Convertidor Analógico-Digital (ADC10) del MSP430G2553.

- Convertidor de 10 bits.
- 6 canales externos.
- Vcc y temperatura interna.
- Velocidad:200Ksps.
- Reloj de conversión puede ser seleccionado.
- Referencia interna o externa.
- Disparo por Timer A.
- Cuenta con Interrupción.
- Controlador de transferencia de datos (DTC).

Registros del Convertidor Analógico-Digital (ADC) del MSP430G2553

Estos registros se encuentran explicados a detalle en el capítulo ADC10 de la Guía de Usuario del microcontrolador, proporcionada por Texas Instruments..

1. ADC10SA: ADC10 Data Transfer Start Address (Dirección de Inicio para Transferencia de Datos). Dirección: 01BCh

15	14	13	12	11	10	9	8
ADC10SAx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10SAx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0

Bits 15-1 ADC10 Dirección de Inicio. Escribir en el registro ADC10SA es necesario para iniciar la conversión.

Bit 0 No se usa. Sólo de lectura y siempre se lee 0.

2. ADC10MEM. ADC10 Conversion-Memory Register. (Registro de Memoria de Conversión). Dirección: 1B4h

Los 16 bits (15-0 bits) de este registro son de sólo lectura y dan la salida discretizada de la conversión, es decir, cuando la conversión se ha efectuado el valor de dicha conversión se guarda en este registro.

3. ADC10CTL1. ADC10 Control Register 1. (Registro de Control 1 del ADC10). Dirección: 1B2h

NOTA: Este registro solo puede ser modificado cuando ENC=0.

En este registro se configuran algunos parámetros del ADC10, como la selección del canal de entrada, la fuente de muestreo, la fuente del reloj, si se desea dividir el reloj, así como el modo de conversión que se requiera, entre otros.

15	14	13	12	11	10	9	8
		INCHx		SHSx		ADC10DF	ISSH
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
		ADC10DIVx		ADC10SELx		CONSEQx	ADC10BUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0
Can be modified only when ENC = 0							

INCHx

BITS 15-12

Selección del canal de entrada. Estos bits seleccionan el canal para la conversión simple, o el canal de mayor prioridad para la secuencia de conversiones.

0000	A0
0001	A1
0010	A2
0011	A3
0100	A4
0101	A5
0110	A6
0111	A7
1000	V_{oREF+}
101	V_{oREF+}/V_{oREF-}
110	sensor de temperatura.
1011	$(V_{cc} - V_{ss})/2$

SHSx	Bits 11-10	Fuente de Muestreo.
	00	ADC10SCbit
	01	Timer_A_UT1
	10	Timer_A.OUT0
	11	Timer_A.OUT2

ADC10DF	Bit 9	Formato del dato. 0→ Binario común. 1→ Complemento a 2.
ISSH	Bit 8	Inversión de señal de entrada. 0→La señal de entrada muestreada no es invertida. 1→La señal de entrada muestreada es invertida.

ADC10DIVx	Bits 7-5	ADC10 Divisor de reloj
	000	/1
	001	/2
	010	/3
	011	/4
	100	/5
	101	/6
	110	/7
	111	/8

ADC10SELx	Bits 4-3	ADC10 Selección de la fuente de reloj.
	00	ADC10OSC
	01	ACLK
	10	MCLK
	11	SMCLK

CONSEQx	Bits 2-1	Selección del modo de conversión
	00	Un solo canal-una sola conversión.
	01	Secuencia de canales.
	10	Repetir un solo canal.
	11	Repetir una secuencia de canales.

ADC10BUSY BIT0
ADC10 Ocupado. Este bit indica que la operación de conversión o muestreo se está realizando.
0→La operación no está activa.
1→ Una secuencia, una muestra o una conversión se está efectuando.

4. ADC10CTL0. ADC10 Control Register 0 (Registro de Control 0 del ADC10). Dirección: 1B0h

Este registro también sólo puede ser modificado con ENC=0.

En este registro se configuran los parámetros restantes del ADC10, como la selección de referencia, el tiempo de muestreo, velocidad de muestreo, en este registro se habilita la conversión análoga-digital, también se habilita en éste las interrupción del ADC, es decir, la interrupción cuando la conversión se ha generado. También en este registro se activa el bit para dar inicio a la conversión.

15	14	13	12	11	10	9	8
	SREFx		ADC10SHTx	ADC10SR	REFOUT	REFBURST	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	
7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ENC = 0

SREFx Bits 15-13 Selección de referencia.

000	$V_{R+} = V_{CC}$ y $V_{R-} = V_{SS}$
001	$V_{R+} = V_{REF+}$ y $V_{R-} = V_{SS}$
010	Checar hoja de especificación de datos proporcionada por el fabricante.
011	
100	
101	
11	
111	

ADC10SHTx Bits 12-11 ADC10 tiempo de muestreo.

00	4xADC10CL s
01	8xADC10CLKs
10	16xADC10CLKs
11	64xADC10CLKs

ADC10SR Bit 10 ADC10 Velocidad de muestreo. Este bit selecciona el buffer de referencia para la máxima velocidad de muestreo. Activando ADC10SR se reduce el consumo actual del buffer de referencia.
0→ El Buffer de referencia soporta **arriba** de 100Ksps
1→ El buffer de referencia soporta más de 50ksps

REFOUT	Bit 9	Salida de referencia. Checar hoja de especificación de datos. 0→ Referencia de salida desactivada. 1→ Referencia de salida activada.
REFBURST	Bit 8	Referencia burst . 0→ Buffer de referencia activado continuamente. 1→ Buffer de referencia activado sólo durante el muestreo-y-conversión.
MSC	Bit 7	Múltiple muestreo y conversión. Válido sólo para modos repetidos o de secuencias. 0→ El muestreo necesita un flanco de subida de la señal SHI para disparar (trigger) cada muestreo-y-conversión. 1→ El primer flanco de subida de la señal el temporizador (timer) de muestreo, muestreos y conversiones se automáticamente tan pronto como conversión se haya completado.
REF2_5V	Bit 6	Voltaje de referencia generado. REFON debe de estar activado. 0→ 1.5V 1→ 2.5V
REFON	Bit 5	Activa generador de referencia. 0→ Referencia desactivada. 1→ Referencia activada.
ADC10ON	Bit 4	Activa ADC10 0→ ADC10 deshabilitado. 1→ ADC10 habilitado.
ADC10IE	Bit 3	Interrupción de ADC10 0→ Habilita interrupción. 1→ Deshabilita interrupción

ADC10IFG	Bit 2	ADC10 Bandera de interrupción. Este bit se activa si el registro ADC10MEM es cargado con el resultado de la conversión. Este bit es automáticamente reseteado cuando la petición de interrupción es aceptada, también se puede borrar por software. 0→ No existe interrupción pendiente. 1→ Interrupción pendiente.
ENC	Bit 1	Habilita conversión. 0→ ADC10 deshabilitado. 1→ ADC10 habilitado.
ADC10SC	Bit 0	ADC10 Inicio de conversión. ADC10SC y ENC deben ser activados juntos en una sola instrucción. ADC10SC se resetea automáticamente. 0→ No empieza muestreo-y conversión. 1→ Empieza muestreo y conversión.

Operación básica del ADC10

Los siguientes tres pasos son necesarios para llevar acabo una sola conversión con el ADC10.

Paso 1. Configurar el ADC10, habilitando los bits según la aplicación y el requerimiento en los registros, además habilitar el bit de ADC10ON. Durante esta operación el bit ENC debe de estar deshabilitado, ya que muchos de los bits en los registros ADC10CTL0 y ADC10CTL1 pueden ser modificados sólo cuando ENC=0.

Paso 2. Ahora sí, habilitar el bit ENC para permitir la conversión. Nuevamente se realza la importancia de que esta operación no se debe de efectuar cuando se realice la configuración del ADC10.

Paso 3. Habilitar la conversión, ya sea activando el bit ADC10SC o por un flanco desde el Timer_A.

Los últimos dos pasos deben repetirse para cada conversión, lo cual requiere deshabilitar (limpiar) y prender nuevamente el bit ENC. La razón de llevar a cabo estos dos pasos en cada conversión es porque el ADC10 puede ser disparado directamente desde el Timer_A no sólo por software. Por lo que una nueva conversión no debe de empezar hasta que la antigua haya sido procesada y la conversión se reactiva desactivando y activando el bit ENC.

En cuanto al disparo por software se tiene más flexibilidad. En este caso la primera conversión puede ser disparada encendiendo o activando los bits ENC y ADC10SC en una sola instrucción:

bis.w #ENC+ADC10SC,&ADC10CTL0.

Las siguientes conversiones pueden ser disparadas habilitando solamente el bit ADC10SC sin desactivar y activar ENC.

Desarrollo de la práctica

1. Llevar a cabo la conexión del Trimpot con el MSP430G2553 como se muestra en el siguiente diagrama.

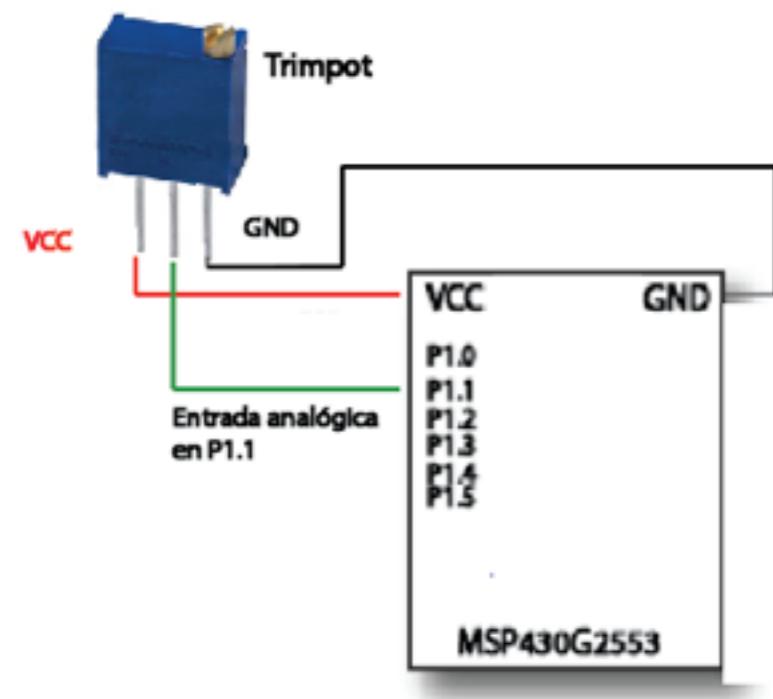


Figura 5.2. Diagrama de conexión.

2. El programa a ejecutarse en esta práctica realiza lo siguiente:

-- Configura el convertidor (ADC10) con un tiempo de muestreo de 16xADC10CLOKs (comúnmente se realiza con este tiempo). Se habilita la interrupción del convertidor y se elige el canal 1 (P1.1) como entrada analógica.

-- Al inicio del programa principal se activa la opción de bajo consumo de potencia, sale de este modo cuando la interrupción del convertidor se lleve a cabo, es decir, cuando la conversión haya concluido.

-- Si el voltaje a plena escala es el del microcontrolador, es decir, 3.6 volts el registro ADC10MEM tendrá el valor hexadecimal 3FF (1023 decimal). Una vez que se ha efectuado la conversión se pregunta si el valor del registro ADC10MEM es menor que 0.5Vcc es decir que 1FF hexadecimal, si es menor brinca a la etiqueta verde y se enciende el LED Verde (LED2) del microcontrolador el cual mediante el jumper J5 está conectado al bit 6 del puerto 1 (p1.6) y se apaga el LED Rojo (LED1) del microcontrolador conectado al bit 0 del puerto 1 (p1.0) mediante el jumper 5.

-- Entonces, si la entrada analógica tiene un valor **inferior** a 1.8Volts equivalente a 0x1FF (producido por el divisor de tensión que genera el potenciómetro) se encenderá el LED Verde (LED2) y se apagará el LED Rojo (LED1) como se muestra en la siguiente figura.



Figura 5.3. Voltaje de entrada analógica (entrada en P1.1) inferior a Vcc/2 ($3.6/2=1.8$ volts). Enciende LED Verde.

-- Y si la entrada analógica tiene un valor **superior** a 1.8Volts, equivalente a 0x1FF (producido por el divisor de tensión que genera el potenciómetro) se encenderá el LED Rojo (LED1) y se apagará el LED Verde (LED2) como se muestra en la siguiente figura.

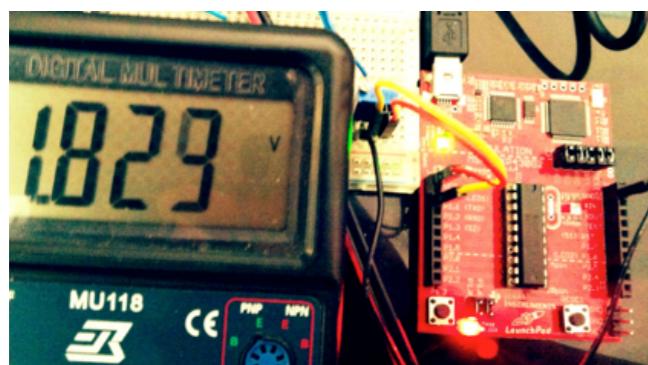


Figura 5.4. Voltaje de entrada analógica (entrada en P1.0) inferior a Vcc/2 ($3.6/2=1.8$ volts). Enciende LED Rojo.

3. Copiar el código en lenguaje Assembly que se proporciona a continuación.

```
*****  
.cdecls C,LIST, "msp430g2553.h"  
.global RESET  
;  
.text  
;Inicio de Programa  
;  
RESET mov.w #400h,SP ;Inicialización del StackPointer  
StopWDT mov.w #NDTPW+WDTHOLD,&WDTCTL ;Detiene Perro Guardian.  
;  
Configuración del ADC10  
;Se configura con el tiempo de muestreo 16xADC10CLKs, se habilita el convertidor  
;se habilita la interrupción del convertidor y se elige al Canal 1 (P1.1) para  
;entrada analógica.  
;  
SetupADC10 mov.w #ADC10SHT_2+ADC10ON+ADC10IE,&ADC10CTL0 ;16x tiempo de muestreo  
;Habilita ADC10  
;Habilita interrupción  
mov.w #INCH_1, &ADC10CTL1 ;Selecciona P1.1 como entrada analógica  
;  
SetupP1 bis.b #01000001b,&P1DIR ;Configura P1.1 como entrada y  
;P1.0 y P1.6 como salida.
```

Programa Principal	
Mainloop	bis.w #ENC+ADC10SC,&ADC10CTL0 ; Comienzo del muestreo y conversión. bis.w #CPUOFF+GIE,SR ;Se configura el modo de bajo consumo bic.b #BIT0+BIT6,&P1OUT ;Se limpia la salida de P1.0 y P1.6. cmp.w #01FFh,&ADC10MEM ; Pregunta si ADC10MEM = A1 > 0.5AVcc jlo Verde ;Si A1 NO es mayor a 0.5Vcc brinca a etiqueta Verde. Si A1 es mayor a 0.5Vcc efectúa lo correspondiente para encender LED rojo.
Rojo	bis.b #000000001b,&P1OUT ;Enciende LED rojo del microcontrolador conectado a P1.0. bic.b #010000000b,&P1OUT ;Apaga LED verde del microcontrolador conectado a P1.6. jmp Fin ;Brinca a etiqueta fin para volver a comenzar.
Verde	bis.b #01000000b,&P1OUT ;Enciende LED verde del microcontrolador conectado a P1.6. bic.b #000000001b,&P1OUT ;Apaga LED rojo del microcontrolador conectado a P1.0.
Fin	jmp Mainloop ;Brinca nuevamente al ciclo principal.

ADC10_ISR; Rutina de servicio de interrupción.	
	bic.w #CPUOFF,0(SP) ;Salida del modo de bajo consumo reti ;Retorno de interrupción ;
Interrupt Vectors	
.reset	RESET ; MSP430 RESET Vector ;
.int05	ADC10_ISR ; ADC10 Vector ;

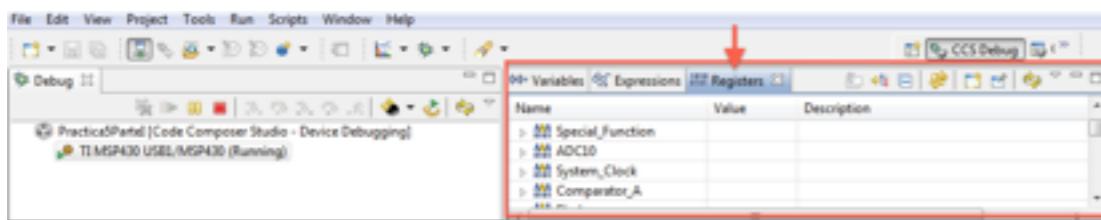
4. Ejecuta el programa bajo las siguientes instrucciones:

4.1 Presionar el botón de debugger

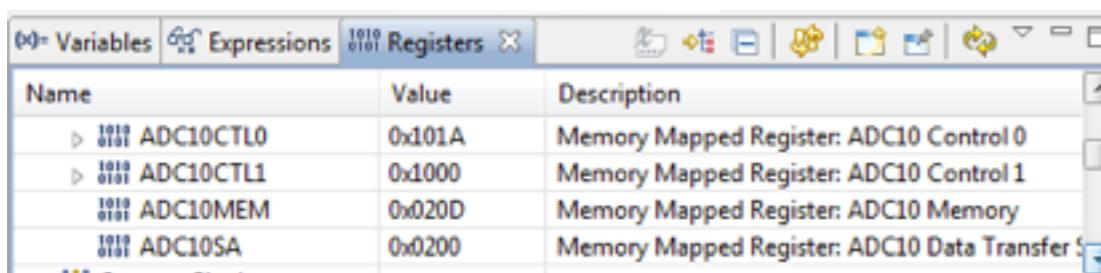
4.2 Presionar el botón Resume

4.3 Posteriormente presionar botón Suspend .

4.4 En la ventana superior derecha seleccionar pestaña Registers.



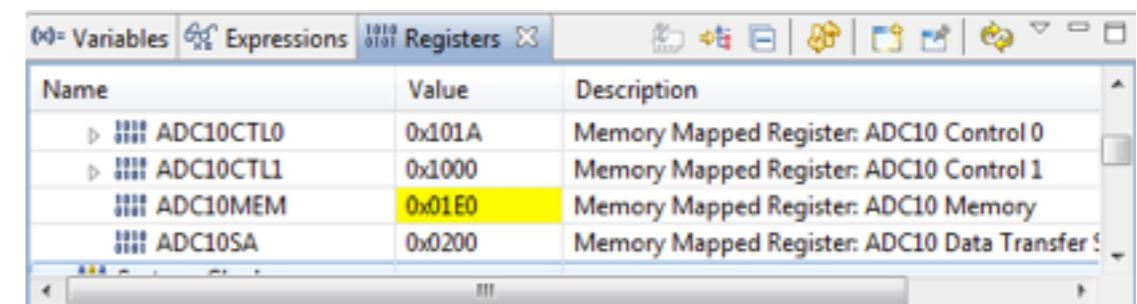
4.5 Buscar etiqueta ADC10 y abrir, dentro de ésta están todos los registros correspondientes al convertidor analógico-digital, y sus valores.



Name	Value	Description
ADC10CTL0	0x101A	Memory Mapped Register: ADC10 Control 0
ADC10CTL1	0x1000	Memory Mapped Register: ADC10 Control 1
ADC10MEM	0x020D	Memory Mapped Register: ADC10 Memory
ADC10SA	0x0200	Memory Mapped Register: ADC10 Data Transfer

4.6 Observar que el registro ADC10MEM ya tiene un valor de la conversión efectuada. Mover el potenciómetro y volver a presionar botón Resume y luego botón Suspend y observar el valor de la conversión en ADC10MEM, si el valor de este registro es mayor a 0x01FF el LED rojo debe de estar encendido y si el valor del registro ADC10MEM es menor que 0x01FF el LED verde debe de estar apagado.

4.7 Cada que se mueva el potenciómetro (Trimpot) se cambiará el valor del registro ADC10MEM y el recuadro se pondrá amarillo por unos segundos.



Name	Value	Description
ADC10CTL0	0x101A	Memory Mapped Register: ADC10 Control 0
ADC10CTL1	0x1000	Memory Mapped Register: ADC10 Control 1
ADC10MEM	0x01E0	Memory Mapped Register: ADC10 Memory
ADC10SA	0x0200	Memory Mapped Register: ADC10 Data Transfer

5. Modificar el código para que ahora se realicen las siguientes acciones:

5.1. Conectar cuatro LEDs externos.

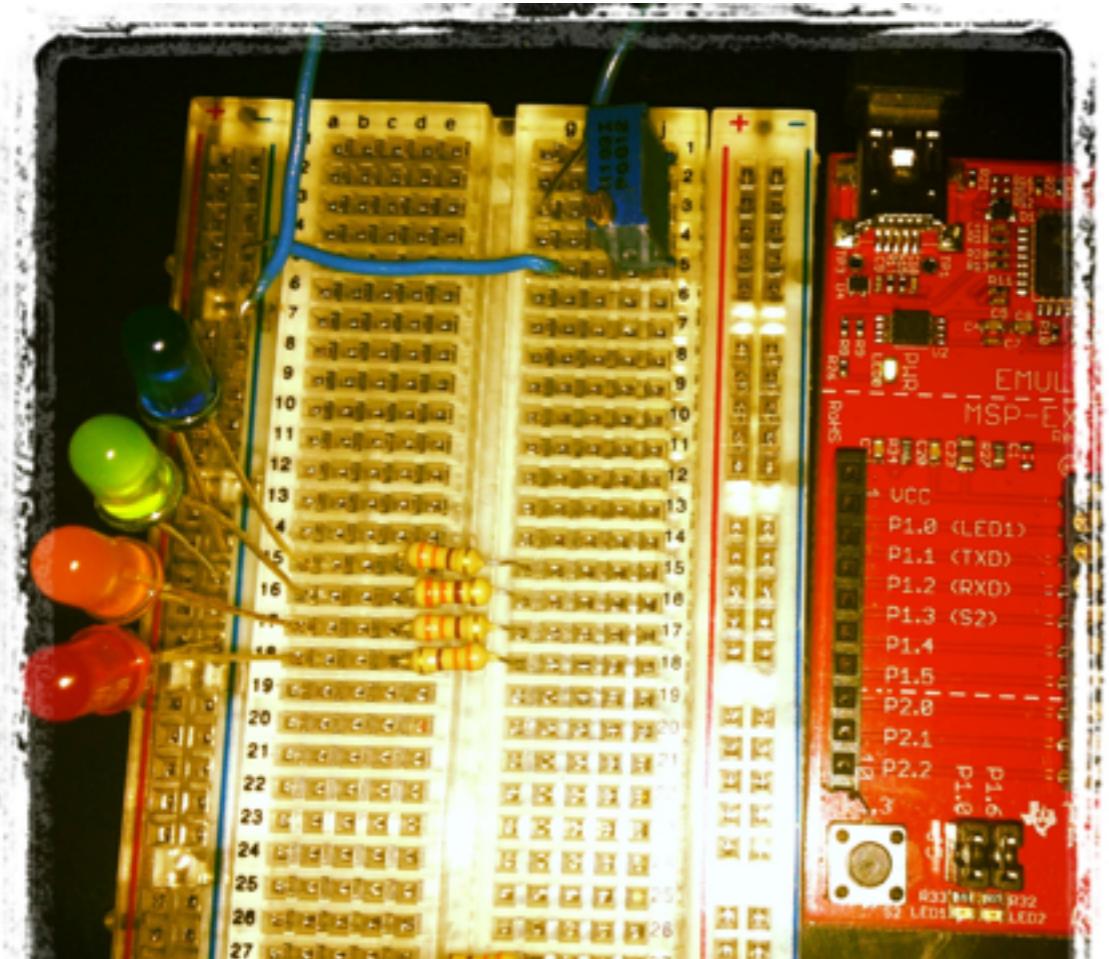
5.2. Cuando el voltaje de entrada es menor a 0.25Vcc pero mayor a 0 se enciende **LED1**.

Cuando el voltaje de entrada es menor a 0.5Vcc pero mayor a 0.25Vcc se enciende **LED2**.

Cuando el voltaje de entrada es menor a 0.75Vcc pero mayor a 0.5Vcc de enciende **LED3**.

Cuando el voltaje de entrada es menor a 1Vcc pero mayor a 0.75Vcc se enciende el **LED4**.

5.3 La rutina de servicio de interrupción ISR permanece igual.



6. Hacer reporte correspondiente a la práctica 5.

Práctica 6

Timer_A como temporizador.

OBJETIVOS

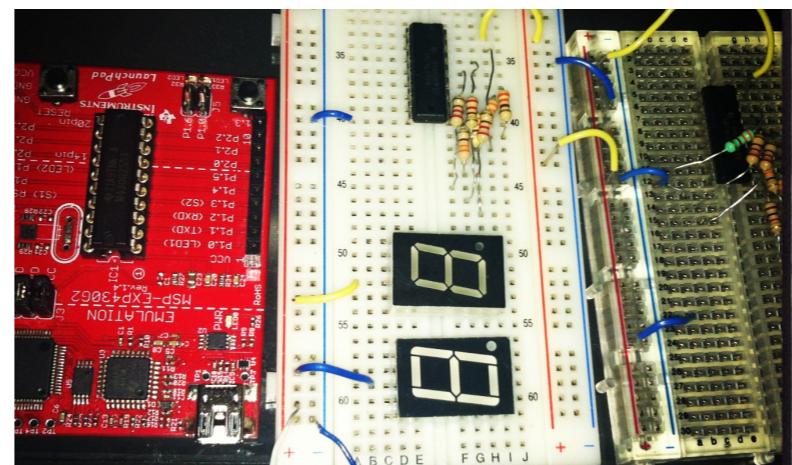
1. Aprender los conceptos básicos del Timer_A.
2. Conocer los modos de cuenta con los que trabaja el Timer_A.
3. Conocer los registros del Timer A necesarios para configurarlo como simple temporizador.
4. Conocer el funcionamiento y configuración del Timer A como temporizador.

MATERIALES

- 1 Microcontrolador MSP430G2553.
- 2 Decodificadores 74LS47 ó 74LS48.
- 2 Displays de 7 segmentos de ánodo común, ó de cátodo común (Según se haya elegido el decodificador).
- 14 Resistencias de 220 ó 330 ohms.

NOTA: Los decodificadores 74LS47 son para los displays de ánodo común, y los decodificadores 74LS48 son para los displays de cátodo común.

Otro periférico de gran utilidad con el que cuenta este microcontrolador es el Timer_A o Temporizador_A. El MSP430G2553 cuenta con dos timers: Timer0_A que tiene tres canales (0,1 y 2) y el Timer1_A el cual también cuenta con tres canales (0,1 y 2). Ambos Timers trabajan de la misma manera, y sus registros se configuran igual. Los timers se utilizan para medir y generar intervalos de tiempo precisos, generar distintas bases específicas de tiempo, también como contador de eventos, para generar PWM a distintas frecuencias y capturar señales. Si se requiere tener un tiempo exacto y preciso en lugar de utilizar una subrutina de retardo se debe de utilizar el Timer A. Debido a todas las funciones que maneja este periférico, el Timer_A se estudia en tres prácticas.



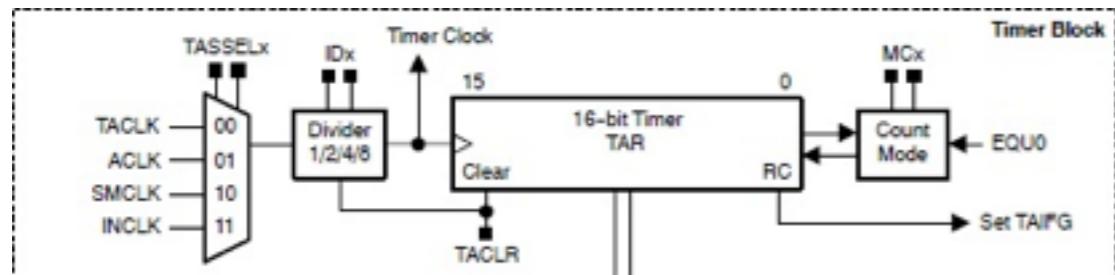
En esta práctica se maneja el Timer_A como temporizador para generar una base de tiempo de un segundo, y crear un contador de segundos de 0-99, es decir, cada un segundo se aumenta el contador hasta llegar a 99 segundos, el resultado se despliega en dos displays de siete segmentos. Esto puede ser la base de un reloj digital, para hacer un reloj digital se recomienda desplegar el resultado en un LCD en lugar de Displays de 7 segmentos, ya que es más barato y práctico.

Metodología

TIMER A como temporizador

El timer no es más que un contador el cual se utiliza para distintos fines y aplicaciones, no tiene un concepto directo con el tiempo, es tarea del programador establecer una relación entre el valor del contador y el tiempo y esto depende esencialmente de la frecuencia del reloj para el Timer.

Lo primero a analizar es el diagrama de bloque del Timer A.



Al principio del bloque se tiene un multiplexor con cuatro entradas y una salida controlado por TASSELx, estas entradas son las distintas fuentes de reloj que puede manejar el Timer_A:

TACLK es un reloj externo.

ACLK es reloj interno y usualmente lento, típicamente de 32KHz.

SMCLK es el reloj interno de mayor frecuencia en este microcontrolador es de 1MHz.

INCLK también es un reloj externo.

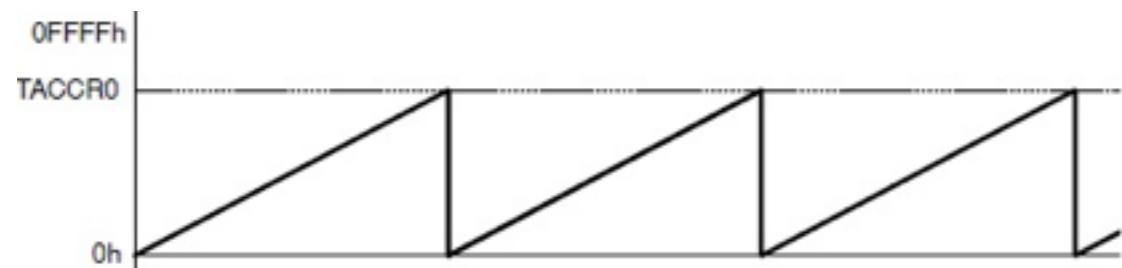
Después de la selección de reloj hay un divisor de frecuencias controlado por IDx el cual divide la señal entrante de reloj en 1, 2, 4 ó 8. Después esta señal ya seleccionada y dividida pasa a alimentar al contador/temporizador de 16 bits, almacenando la cuenta en un registro llamado TAR, TAR incrementa su valor en cada flanco de subida del reloj. Este contador/temporizador también es controlado por los bits MCx los cuales se encargan de controlar **el modo de cuenta** del contador/temporizador.

Se observa que tanto el divisor de frecuencia como el timer cuentan con un bit llamado TACLR el cual se encarga de resetear los valores tanto de TAR como del divisor de frecuencias IDx. Finalmente se tiene una sola salida TAIFG, la cual es la bandera de interrupción y se activará según se haya configurado el temporizador o timer.

Modos de cuenta del TIMER A

1. Modo Ascendente (Modo Up).

En este modo el timer contará desde cero hasta el valor almacenado en TACCR0, y posteriormente se reinicia la cuenta y continúa de nuevo hasta llegar al valor de TACCR0. Es decir, la cuenta en el registro TAR se reinicia una vez que haya alcanzado el valor almacenado en el registro TACCR0. Este modo de cuenta se aprecia mejor en la siguiente imagen.



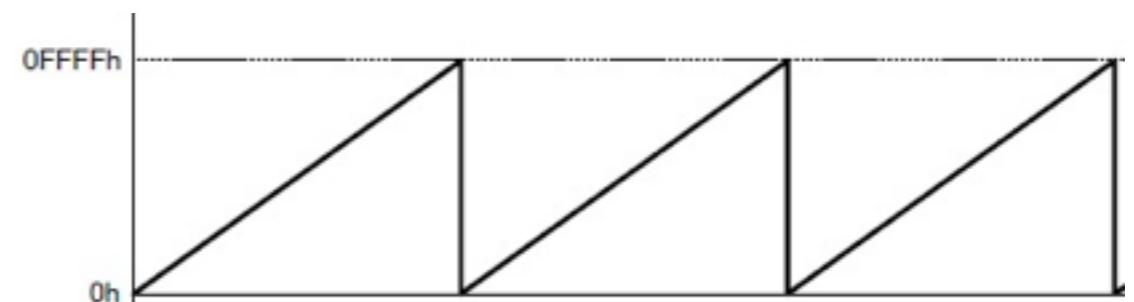
La interrupción se genera en el momento en el que la cuenta en TAR cambia de el valor de TACCR0 a 0, es decir, cuando el registro TAR se reinicia. Por ejemplo si TASSEL es 10h se elige el reloj SMCLK y posteriormente ponemos 11h en IDx dividimos el reloj en 8 por lo que queda una frecuencia de 125KHz que alimenta al timer.

Si se desea que se genere una interrupción cada 0.5 segundos deben de transcurrir 62,500 pulsos de la señal de

reloj elegida, porque $(62,500 \times (1/125000\text{Hz}))=0.5$ segundos, por lo tanto el registro TACCR0 se carga con el valor de 62,500, y se configura MCx con 01h para definir el modo ascendente ó modo Up. Los bits correspondientes a TASSELx, IDx y MCx se encuentran en el registro TACTL.

2. Modo Continuo

En este modo el Timer contará desde 0 hasta el máximo valor permitido por TAR que es 0xFFFF y después regresa a cero y vuelve a comenzar la cuenta.

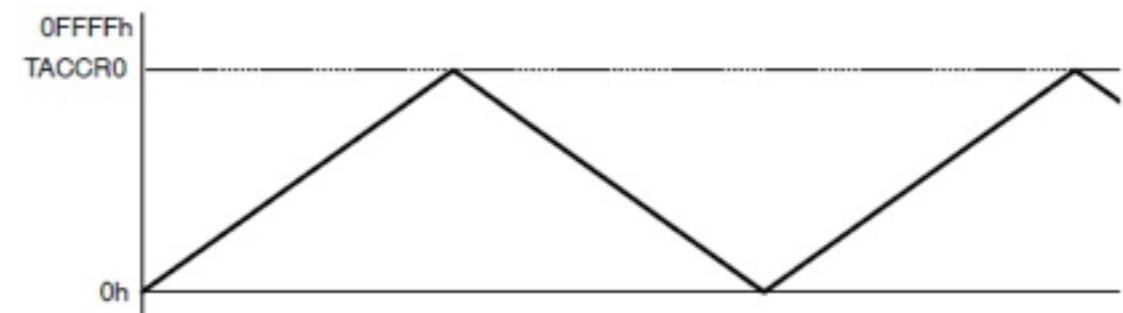


La interrupción se genera en el momento que la cuenta en TAR pasa de FFFFh a 0h

Si utilizamos el reloj SMCLK sin dividirlo tenemos una frecuencia de 1MHz, lo que genera un periodo de 1us. Con este modo continuo se generaría la interrupción cada 65.535ms porque (FFFFh=65535decimal) y $65535 \times 1\text{us} = 65.535\text{ms}$.

3. Modo Ascendente/Descendente (Modo Up/Down)

En este modo el Timer cuenta desde cero hasta el valor de TACCR0 de forma ascendente y después de manera descendente de TACCR0 hasta cero. Por lo que el periodo será el doble del valor almacenado en TACCR0. Esto es útil para generar pulsos simétricos.



En este modo la interrupción (TAIFG se activa) se genera cuando la cuenta en el registro TAR está de forma descendente y pasa de 0x0001 a 0x0000.

Registros del TIMER A

El timer A cuenta con cinco registros:

- 1 **TACTL:** Timer_A Control (Control del Timer_A).
2. **TAR:** Timer_A Counter(Contador del Timer_A).
3. **TACCRx:** Timer_A Capture/Compare Register (Registro de Captura /Comparación del Timer_A). Donde x puede ser 0, 1 ó 2 según el canal del timer que se esté configurando.

4. **TACCTLx:** Timer_A Capture/Compare Control Register

Para fines de utilizar el Timer como simple temporizador se escribe en TACCR0 el número de cuentas al que tiene que llegar el registro TAR, y en TACCTLx sólo se habilita la interrupción. En las prácticas subsecuentes se explicaran estos dos registros. Donde x puede ser 0, 1 ó 2 según el canal del timer que se esté configurando.

5. **TAIVx:** Timer_A Interrupt Vector Register (Registro de Vector de Interrupción del Timer_a).

Todos estos registros se explican a detalle en la guía de usuario proporcionada por Texas Instruments en el apartado de Timer A. Es importante mencionar que el MSP430G2553 no cuenta con Timer_B. A continuación se muestran los registros a utilizar en esta práctica.

1 **TACTL:** Timer_A Control (Control del Timer_A).

La configuración de este registro afecta a los tres canales de los dos timers. Para configurar el Timer0_A3 el registro es **TA0CTL** y para configurar el Timer1_A3 el registro es **TA1CTL**, por default TACTL hace alusión a configuración del Timer0_A. En este registro de control se configura la fuente de reloj, el divisor de frecuencias, el modo de cuenta e interrupciones del Timer

Registro TACTL:

15	14	13	12	11	10	9	8
Unused							
rw-(0)							
7	6	5	4	3	2	1	0
IDx		MCx		Unused	TACLR	TAIE	TAIFG
rw-(0)							

Unused BITs 15-10

TASSELx BITs 9-8 Timer_A Selección de Fuente de Reloj

00	TACLK
01	ACLK
10	SMCLK
11	INCLK

IDx. BITS 7-8 Divisor de frecuencias

00	/1
01	/2
10	/4
11	/8

MCx BITs 5-4. Modo de cuenta

00	Modo detenido. El timer está parado.
01	Modo Ascendente. El timer cuenta ascendente hasta TACCR0
10	Modo Continuo. El timer cuenta ascendente hasta 0xFFFF.
11	Modo Ascendente/Descendente. El timer cuenta ascendente hasta TACCR0 y después descendente hasta 0x0000.

TACLR BIT 2 Limpiar Timer A

Activando este bit, se reinicia la cuenta en TAR, se borra el divisor de frecuencias. Este bit se resetea automáticamente y siempre se lee como cero.

TAIE BIT 1 Timer_A Interrupción Habilitada.

0	Interrupción deshabilitada
1	Interrupción habilitada.

TAIFG BIT0 Timer_A Bandera de interrupción.

0	No hay Interrupción pendiente
1	Hay interrupción pendiente

2. TAR: Timer_A Counter(Contador del Timer_A).

Este registro es incrementado o decrementado por cada flanco ascendente de la señal cuadrada de reloj, es decir, según la fuente de reloj seleccionada será el tiempo en que se incremente o decremente este registro. Este registro es el más importante de el Timer A , en el cual se puede escribir ó leer.

Registro TAR

15	14	13	12	TARx	11	10	9	8
rw-(0)								
7	6	5	4	3	2	1	0	
				TARx				
rw-(0)								

TARx **BITs 15-0 Timer_A Registro.** El registro TAR es la cuenta del Timer_A.

Interrupciones del TIMER A

Las interrupciones del Timer0_A3 (A3 porque son 3 canales 0, 1 y 2) en la tabla de vectores de todas las interrupciones enmascaradas tiene reservado dos vectores, el #24 y el #25. El vector #24 corresponde a 3 banderas de interrupción: TA0CCR1 CCIFG, TA0CCR2 CCIFG y TAIFG. El vector #25 sólo refiere a la bandera TA0CCR0 CCIFG.

Del mismo modo las interrupciones del Timer1_A3 en la tabla de vectores de todas las interrupciones enmascaradas tiene reservado dos vectores, el #28 y el #29. El vector #28 corresponde a 3 banderas de interrupción: TA1CCR1 CCIFG, TA1CCR2 CCIFG y TAIFG. El vector #29 sólo refiere a TA1CCR0 y CCIFG.

A continuación se muestra la tabla de vectores.

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range ⁽¹⁾	PORIFG RSTIFG WDTIFG KEYV ⁽²⁾	Reset	0FFF Eh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG ⁽²⁾⁽³⁾	(non)-maskable (non)-maskable (non)-maskable	0FFF Ch	30
Timer1_A3	TA1CCR0 CCIFG ⁽⁴⁾	maskable	0FFF Ah	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG ⁽²⁾⁽⁴⁾	maskable	0FFF B h	28
Comparator_A+	CAIFG ⁽⁴⁾	maskable	0FFF 6 h	27
Watchdog Timer+	WDTIFG	maskable	0FFF 4 h	26
Timer0_A3	TA0CCR0 CCIFG ⁽⁴⁾	maskable	0FFF 2 h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG (3) ⁽⁴⁾	maskable	0FFF 0 h	24
USCI_A0/USCI_B0 receive USCI_B0 I2C status	UCA0RXIFG, UCB0RXIFG ⁽²⁾⁽⁵⁾	maskable	0FFE Eh	23
USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	UCA0TXIFG, UCB0TXIFG ⁽²⁾⁽⁶⁾	maskable	0FFE Ch	22
ADC10 (MSP430G2x53 only)	ADC10IFG ⁽⁴⁾	maskable	0FFE Ah	21
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE 6 h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7 ⁽²⁾⁽⁴⁾	maskable	0FFE 4 h	18
			0FFE 2 h	17
			0FFE 0 h	16

Pero... ¿Cómo se puede identificar cuál bandera de interrupción se activó en el vector #24 y #28, ya sea TACCR1 CCIFG, o TACCR2 CCIFG, o bandera de desbordamiento TAIFG?

Sucede que existe un registro llamado **TAIV**, cuyo valor permite conoce cual de las 3 interrupciones ha sucedido.

TAIV

TAIV significa Timer0_A3 Interrupt Vector Value (Valor de Vector de Interrupción), y tiene la propiedad de que cuando se produce una interrupción este registro se carga con un valor según la bandera de interrupción que se activó.

Por ejemplo, suponiendo que la bandera de TAIFG fue la que originó la interrupción el registro TAIV se carga con el valor 0x0A, y después de que el compilador invoque a la función de interrupción del vector #24 del Timer0_A3, se debe de consultar dentro de la Rutina de Servicio de Interrupción (ISR) el valor del registro TAIV, para determinar el tipo de interrupción, en este caso TAIV tendría un valor de 0x0A, después de esta lectura el registro se resetea automáticamente. A continuación se muestra la tabla de los valores de los vectores de interrupción.

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	-	
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2 ⁽¹⁾	TACCR2 CCIFG	
06h	Reserved	-	
08h	Reserved	-	
0Ah	Timer overflow	TAIFG	
0Ch	Reserved	-	
0Eh	Reserved	-	Lowest

Configuración básica del TIMER A

Para generar una base de tiempo o un intervalo de tiempo la configuración del Timer_A resulta sencilla.

Primeramente se debe de configurar el Registro TACTL con la fuente de reloj, el divisor de frecuencia, el modo de cuenta deseado, y las interrupciones. En este registro también se

se limpia el Timer_A. Por ejemplo si trabaja con el Timer0_A y se desea que la fuente de reloj sea interna y con la máxima frecuencia se elige el reloj SMCLK poniendo un 2 en los bits de TASSELx y si éste se quiere dividir entre 8 se pone un 3 en los bits de IDx, si se quiere trabajar con el modo cuenta ascendente/descendente (Up/Down) se requiere poner un 3 en los bits MCx y además se habilitan interrupciones. Todo esto se puede hacer en una sola instrucción:

mov.w TASSEL_2+ID_3+MC_3,&TA0CTL (En Assembly)

TA0CTL=TASSEL_2+ID_3+MC_3 (En C).

Esto nos da un reloj de 1MHz dividido entre 8, es decir 125KHz, y la cuenta se realiza de modo ascendente / descendente.

Posteriormente se carga en el registro **TA0CCR0** el número de cuentas que se quiere obtener para tener un tiempo específico. En el ejemplo anterior se configuró el modo Up/ Down, en este modo el periodo de tiempo es el doble del cargado en TA0CCR0, suponiendo que cargamos a TA0CCR0 con 62500, con este modo de cuenta el registro TAR cuenta ascendente hasta 62500 y luego descendente hasta 0.

El periodo de tiempo es el doble en TA0CCR0 con el modo ascendente/descendente por lo tanto tenemos un periodo total de: $(2 * (62500 * (1/125\text{KHz}))) = 1$ segundo. Es decir cada un segundo se reinicia la cuenta y se genera interrupción (si es que ésta es habilitada).

Si se tuviera un modo de cuenta ascendente el periodo de tiempo sería: $(62500 * (1/125\text{KHz})) = 0.5$ segundos.

Finalmente se habilita la interrupción para que cada que TAR llegue al valor cargado en el registro TA0CCR0 se lleve a cabo dicha interrupción. Esto se hace de la siguiente manera:

mov.w CCIE,&TA0CCTL0; (En Assembly)

TA0CCTL0=CCIE; (En C)

Con esta configuración se efectuará una interrupción cada segundo, dentro de la rutina de servicio de interrupción se pone la tarea que se desea realizar cada determinado tiempo, en este ejemplo 1 segundo.

En las dos prácticas siguientes se estudiará a más detalle los registros **TACCRx** y **TACCTLx**.

Hardware Externo

Decodificador 74LS47 y Display 7 segmentos ánodo común.

El decodificador recibe en su entrada un número binario que será visualizado en el display de 7 segmentos ánodo común, por lo que el decodificador tiene 7 salidas, una para cada segmento del display.

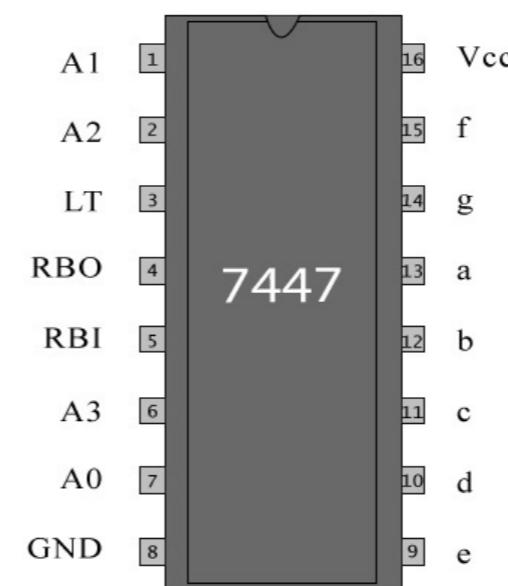


Figura 6.1. Decodificador 74LS47 para display de siete segmentos ánodo común.

Entradas: A0,A1,A2,A3. Estas entradas las tomaremos del microcontrolador.

Salidas: f,g,a,b,c,d,e. Estas salidas van a cada segmento del display.

El decodificador nos permite tener una entrada binaria desde 0000 hasta 1001 (esta entrada la tomaremos del microcontrolador) y desplegar el dígito en un display de siete segmentos.

TABLA 6.1. Entradas y salidas.

ENTRADAS BINARIAS	SALIDAS DESPLEGADAS EN EL DISPLAY 7 SEGMENTOS.
A0 A1 A2 A3	
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

Para más detalles de este circuito integrado buscar la hoja de especificación de datos.

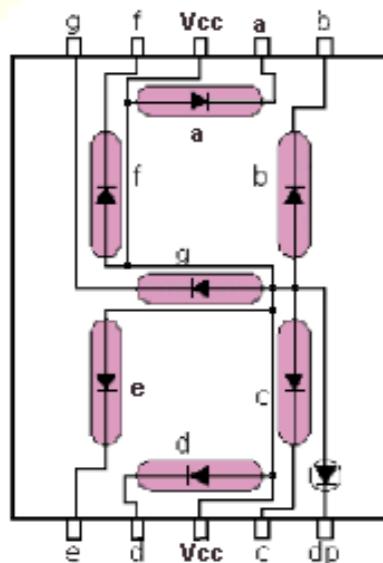
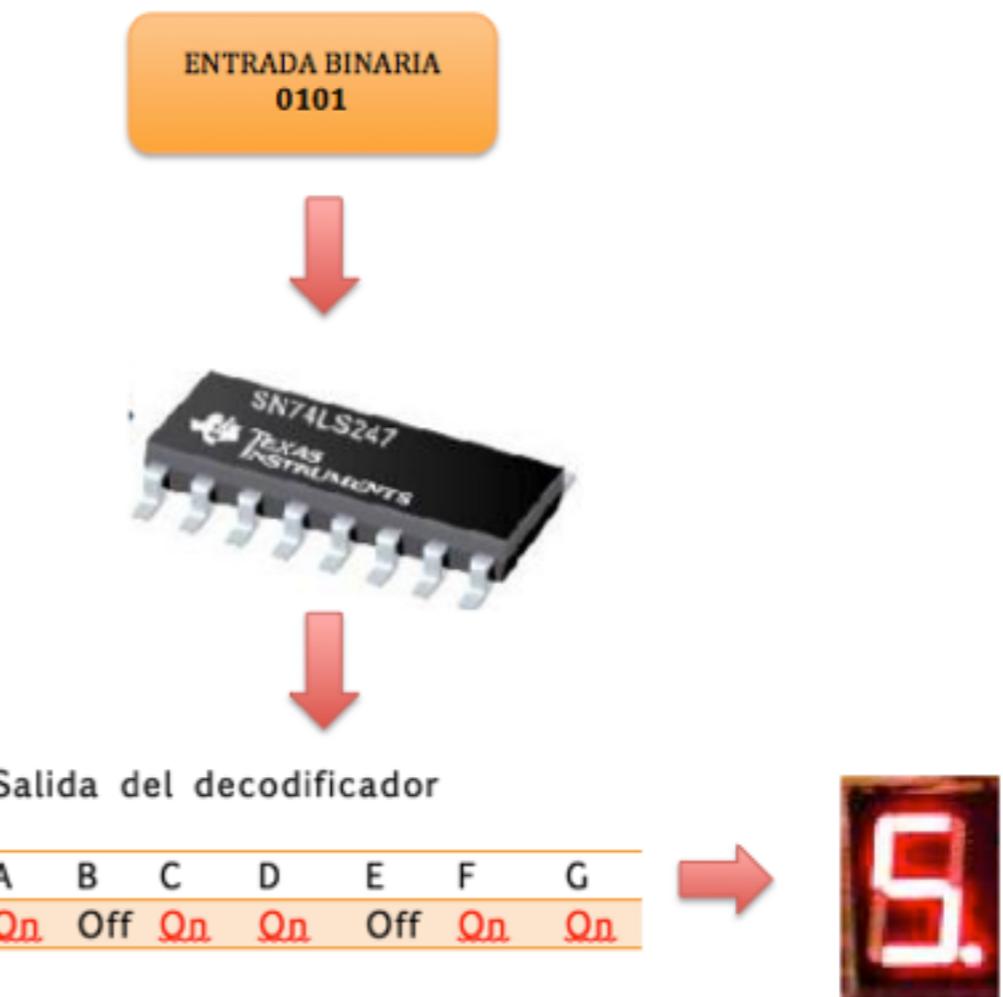


Figura 6.2. Display de 7 segmentos ánodo común.

Las salidas del decodificador (74LS47) son las entradas de cada segmento de este display, según la letra del pin del display se conecta al pin del decodificador.

Se llama de ánodo común porque tienen Vcc en común, y para que cada segmento encienda se debe de llevar a éste a tierra, y así se cierre el circuito.

Por ejemplo si tenemos una entrada binaria 0101 (5 en decimal) se desplegará en el display de 7 segmentos ánodo común el 5, con la ayuda del decodificador 74LS47, como se muestra en el siguiente diagrama.



Decodificador 74LS48 y Display 7 segmentos cátodo común.

El decodificador 74LS48 y el display de 7 segmentos de cátodo común, tienen el mismo funcionamiento que el decodificador de 74LS47 y el display de ánodo común. La diferencia es que el display de cátodo común, tienen en común la tierra y cada segmento que se quiera activar se necesita conectar a Vcc para cerrar el circuito.

El decodificador 74LS48 tiene los mismos pines ubicados en el mismo lugar que el 74LS47.

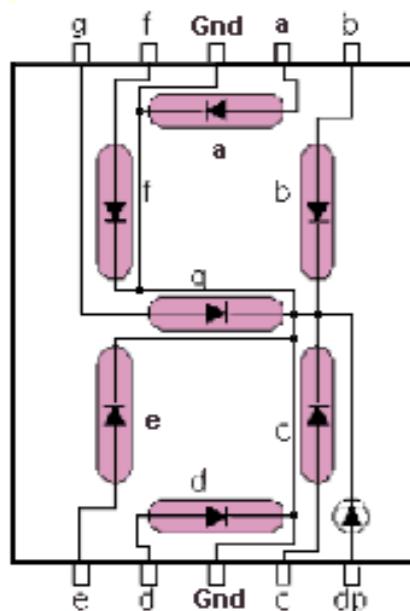


Figura 6.3. Display 7 segmentos cátodo común.

Las salidas del decodificador (74LS48) son las entradas de cada segmento de este display, según la letra del pin del display se conecta al pin del decodificador.

El desarrollo de la práctica se explicará tomando en cuenta al decodificador 74LS47 y el display de ánodo común. Si se cuenta con el decodificador 74LS48 y display de cátodo común, modificar la conexión de acuerdo a lo explicado anteriormente o bien a la hoja de especificación de datos.

Desarrollo de la práctica

En esta práctica se desarrollará un contador de 0-99, cada que pase un segundo aumentará la cantidad del contador. La base de tiempo de un segundo se realiza con Timer0_A0, es decir, con el canal 0 del Timer0_A. En esta práctica se utiliza los bits 0, 1, 2 y 3 del puerto 1 para contar las unidades (0-9), por lo que p1.0, p1.1, p1.2 y p1.3 se configuran como salidas. Estas salidas son utilizadas como entradas binarias al decodificador 74ls47 y las salidas de este decodificador van al primer display de 7 segmentos. Las decenas (0-9) se incrementan cada que las unidades llegan a 9, los bits 0, 1, 2 y 3 del puerto 2 son utilizadas para contar las decenas, por lo tanto p2.0, p2.1, p2.3 y p2.4 se configuran como salidas, estas salidas son utilizadas como las entradas binarias del segundo decodificador 74Ls47 y las salidas de este decodificador se conectan al segundo display de 7 segmentos.

Una vez explicada la práctica a desarrollar, a continuación se desglosan los pasos a seguir para llevarla a cabo,

1. Conectar el primer decodificador 74LS47 y el primer display de 7 segmentos de ánodo común como se muestra en el siguiente diagrama.

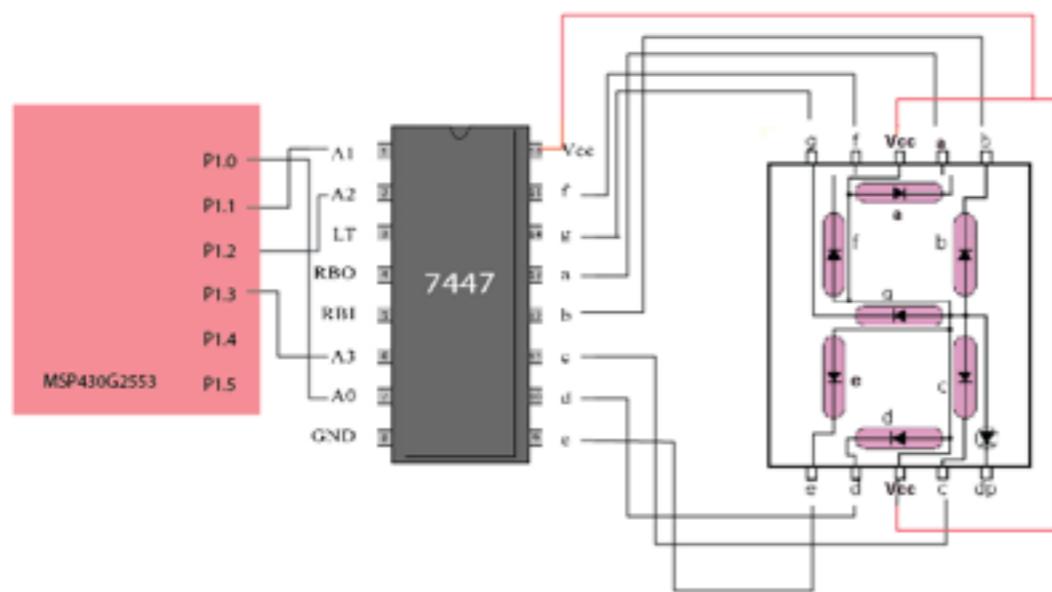


Figura 6.1. Diagrama de conexión para desplegar unidades.

2. Conectar el segundo decodificador 74LS47 y el segundo display de 7 segmentos de ánodo común como se muestra en el siguiente diagrama.

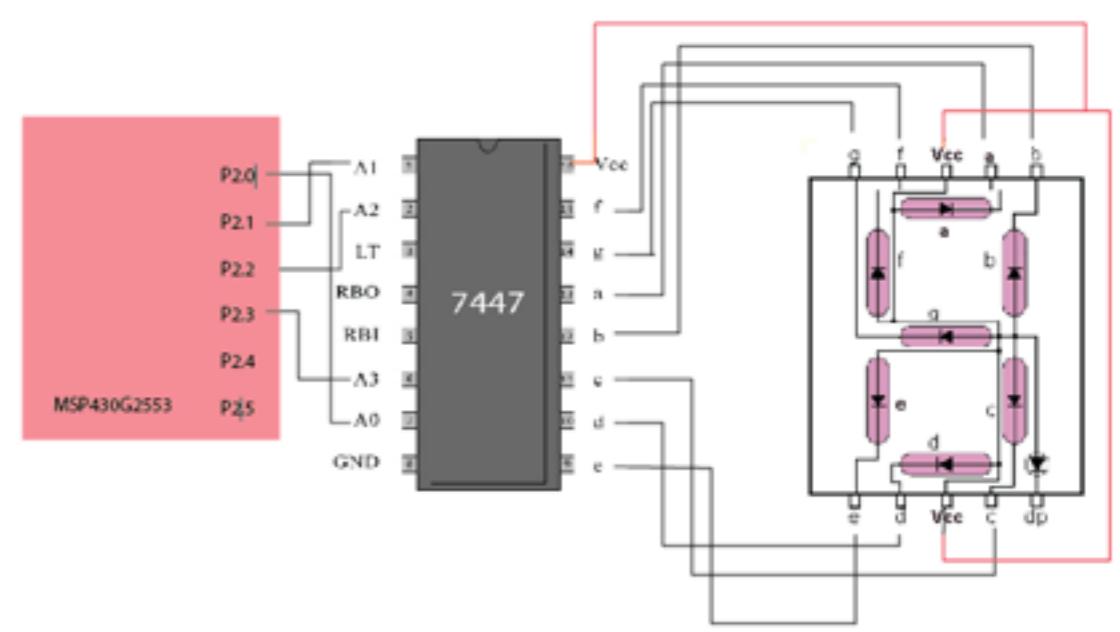


Figura 6.2. Diagrama de conexión para desplegar decenas.

3. Cargar el siguiente programa en código Assembly, y ejecutarlo.

```
.cdecls C,LIST, "msp430g2553.h"
.global RESET
;
.text ;Inicio de Programa
;
RESET mov.w #400h,SP ;Inicialización del StackPointer
StopWDT mov.w #WDTPW+WDTHOLD,&WDTCTL ;Detiene Perro Guardian.
        mov.b &CALBC1_1MHZ,&BCSCTL1 ;Calibracion del
        mov.b &CALDCO_1MHZ,&DCOCTL ;oscilador a 1MHz
;
;Configuración de puertos
;
SetupP1P2 bis.b #0xFF,&P1DIR ;Puerto 1 como Salida.
        bis.b #0xFF,&P2DIR ;Puerto 2 como salida.
        clr &P1OUT
        clr &P2OUT
```

```

;----- Configuración del TimerA_0
;El timer A canal 0 se configura con un reloj de 1MHz (SMCLK) y éste se divide
;en 8 (125kHz). El valor en TA0CCR0 es de 62500 lo que da un valor de
;0.5 segundos (62500*(1/125kHz))=0.5.
;Se configura con Modo Up/Down, cuenta hasta el valor en TA0CCR0 (62500)
;y despues del valor de TA0CCR0 hasta cero (62500) con una cuenta total de 125000
;lo cual es un segundo, cada que la cuenta se reinicie se genera una interrupción,
;es decir se genera una interrupción cada 1 segundo.
;*****
SetupC0    mov.w #CCIE,&TA0CCTL0          ; Se habilita interrupción de
                                                ; TA0CCR0.
                                                ;Número de cuentas para 5 segundos
SetupTA    mov.w #TASSEL_2+ID_3+MC_3,&TA0CTL ;SMCLK reloj(1MHZ),división del
                                                ;reloj en 8 (1MHz/8=125kHz),
                                                ;Modo Up/Down, Cuenta hasta
                                                ;62500(Valor de TA0CCR0) y luego
                                                ;de regreso desde 62500 hasta 0.
                                                ;para tener una base de tiempo
                                                ;de 1 segundo.
;*****
;                Programa principal
;*****
Mainloop   bis.w #CPUOFF+GIE,SR           ; CPU off, modo de bajo consumo de
                                                ;potencia. Habilita interrupciones
                                                ;globales
                                                ;No operación.
;----- TA0_ISR;    Rutina de servicio de interrupción. Contador 0-99
;-----
Unidades   inc &P1OUT                  ;Se incrementa el Puerto 1.
                                                ;Se transfiere contenido de P1OUT a R6
                                                ;Se enmascara sólo 8 bits de R6.
                                                ;cmp #0x0A,R6
                                                ;jz Decenas
                                                ;jmp Fin
Decenas    clr &P1OUT                  ;Se limpia unidades (limpia puerto 1).
                                                ;Incrementa decenas. (incrementa puerto2).
                                                ;Se transfiere contenido de P2OUT a R7.
                                                ;Se enmascara los 8 bits de R7.
                                                ;cmp #0x0A,R7
                                                ;jz Reset
                                                ;jmp Fin
Reset     clr &P2OUT                  ;Limpia decenas (limpia puerto 2).
                                                ;Retorno de interrupción.

```

```

;----- Vectores de interrupción.
;
        .sect ".reset"           ; MSP430 RESET Vector
        .short RESET             ;
;
        .sect ".int09"           ; Timer_A0 Vector
        .short TA0_ISR            ;
;
        .end                      ;Fin.

```

4. Cargar el mismo programa pero ahora en código C y ejecutarlo.

```

#include <msp430g2553.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                                // Detiene Perro Guardian.
    BCSCTL1= CALBC1_1MHZ;                                    // Calibración del reloj a 1MHz.
    DCOCTL= CALDCO_1MHZ;                                    // Calibración del reloj a 1MHz.
    P1DIR |= 0xFF;                                         // Puerto 1 como salida (Unidades).
    P2DIR |= 0xFF;                                         // Puerto 2 como salida (Decenas).
    P1OUT=0x00;                                           
    P2OUT=0x00;
    TA0CCTL0 = CCIE;                                         // Habilita interrupción en TA0CCR0
    TA0CCR0 = 62500;                                         // Número de cuentas para 0.5 segundos.
    TA0CTL = TASSEL_2 + ID_3+ MC_3;                         // SMCLK (1MHz), divide reloj /8=1MHz/8=125Khz
                                                               // Modo Up/Down
    _BIS_SR(LPM0_bits + GIE);                               // Habilita modo LPM0 e interrupciones globales

// Timer A0 Rutina de Servicio de Interrupción (ISR)
#pragma vector=TIMER0_A0_VECTOR

__interrupt void Timer_A (void)
{
    P1OUT += 0x01;                                         // Toggle P1.0
    if(P1OUT==10){
        P1OUT=0;
        P2OUT+=0x01;
        if(P2OUT==10){
            P2OUT=0;
        }
    }
}

```

5. Hacer reporte correspondiente a la práctica 6.

Práctica 7

Timer_A en modo de Comparación

OBJETIVOS

1. Aprender los registros del Timer_A para modo de comparación y captura.
2. Conocer los modos de salidas que tiene el Timer_A en modo de comparación.
3. Conocer el funcionamiento y configuración del Timer A en modo de comparación.
4. Aprender el funcionamiento de un servomotor.
5. Generar una señal de PWM con diferentes ciclos de trabajo, para mover la posición del servo motor.

MATERIALES

- 1 Microcontrolador MSP430G2553.
- 1 Servo-Motor chico o mediano.
- 1 Trimpot de 4.7K-ohms ó 10K-ohms.
- 1 Resistencias de 1K-ohm.
- 1 Fuente externa de 5 volts.
- 1 Cable BNC para osciloscopio,

Una vez ya estudiado el Timer_A como temporizador, es momento de conocer otra función muy importante con la que trabaja dicho Timer, que es el modo de comparación. Este modo nos permite generar intervalos de tiempo específicos, generar PWM a distintas frecuencias y con distintos ciclos de trabajo. En sí nos permite generar una señal con cierta frecuencia específica, y nos permite delimitar el tiempo en que la señal está activa ('1' lógico) y el tiempo en el que la señal está apagada ('0' lógico).



En esta práctica se hará uso de un periférico ya explicado anteriormente, el ADC, y del Timer0_A utilizando los canales 0 y 1. Se generará un PWM con una frecuencia de 50Hz, y diferentes ciclos de trabajo. Los ciclos de trabajo se generarán según el valor del ADC, para ésto hacemos uso del TRIMPOT, conforme cambie el valor del TRIMPOT cambiaré el ciclo de trabajo. La señal de PWM generada se conecta mediante una resistencia de 1k-ohm al servo-motor, y según

varíe el ciclo de trabajo de la señal, es decir, según varíe el valor del TRIMPOT, se moverá la posición del servo desde 0° hasta aproximadamente 180°.

Metodología

Registros de Captura y Comparación del Timer_A

En esta práctica se analizarán los bits de los registros de captura y comparación TACCRx y TACCTLx, los cuales no se revisaron en la práctica anterior y son de suma importancia para el modo de captura y comparación.

1. Registro TACCRx: Timer_A Capture/Compare Register (Registro de Captura /Comparación del Timer_A). Donde x puede ser 0, 1 ó 2 según el canal del timer que se esté configurando.

15	14	13	12	TACCRx	11	10	9	8
rw-(0)								
7	6	5	4	3	2	1	0	
TACCRx								
rw-(0)								

TACCRx Bits 15-0

Modo Comparación. TACCRx sostiene el dato para la comparación con el valor del Timer_A Register (TAR).

Modo Captura. El Timer_A Register (TAR), se copia en el registro TACCRx cuando la captura se haya efectuado.

2. Registro TACCTLx: Timer_A Capture/Compare Control Register (Registro de Control Captura/Comparación). En este registro se configura el Timer_A en modo captura o modo comparación. Se configura la entrada para captura y el modo de salida en el caso de comparación. Se habilita la interrupción de captura/comparación, y se tiene la bandera de interrupción. Donde x puede ser 0, 1 ó 2 según el canal del timer que se esté configurando.

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)
7	6	5	4	3	2	1	0
OUTMODx		CCIE		CCI		OUT	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)
TACCTLx		CCIFG		COV		CCIFG	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

CMx Bits 15-14 Modo Captura

00	No hay Captura.
01	Captura en flanco de subida.
10	Captura en flanco de bajada.
11	Captura en ambos: flanco de subida y flanco de bajada.

CCISx Bits 13-12 Selección de entrada de Captura.

00	CCIx A
01	CCIx B
10	GND
11	Vcc

SCS Bit 11 **Fuente de Captura Sincronizada.** Este bit es utilizado para sincronizar la señal de entrada de captura con el reloj del timer.

0	Captura no sincronizada
1	Captura sincronizada

SCCI Bit 10 **Entrada Captura/Comparación sincronizada.** La señal de entrada seleccionada CCI es conectada con la señal EQUx y puede ser leída por este bit.

Unused Bit 9

CAP Bit 8 **Modo Captura**

0	Modo Captura
1	Modo Comparación

OUTMODx Bits 7-5 Modos de salida.

000	Valor del bit de salida.
001	Set
010	Toggle/reset
011	Set/reset
100	Toggle
101	Reset
110	Toggle/set
111	Reset/set

CCIE Bit 4 **Habilitación de la interrupción de Captura/Comparación.**

0	Interrupción deshabilitada.
1	Interrupción habilitada.

CCI Bit3 **Entrada de captura.** La señal de entrada seleccionada se puede leer con este bit.

OUT Bit2 **Salida.** Para el modo de salida 000, este bit controla directamente el estado de la salida.

0	Salida baja
1	Salida alta

COV Bit1 **Sobreflujo de Captura.** Este bit indica que un sobreflujo de captura ha ocurrido. COV debe resetearse mediante software.

0	No ha ocurrido sobreflujo en la captura.
1	Ha ocurrido sobreflujo en la captura.

CCIF Bit0 **Bandera de interrupción de Captura/Comparación.**

0	No hay interrupción pendiente.
1	Hay interrupción pendiente.

Modos de salida en Comparación

Con la salida del modo de comparación se pueden generar señales PWM. Esta salida de comparación con ocho modos de operación que cuenta generan diferentes señales de salida, estos modos de salida se configuran con los bits de **OUTMOD_x (Bits 7-5)** del registro **TACCTL_x**. A continuación se explica cada modo de salida, individualmente. Para posteriormente mostrar el comportamiento de estos modos de salida, según el modo de cuenta (Ascendente, Continuo, Ascendente/Descendente) que se configuró en los bits de **MC_x** (Bits 5-4) del registro **TACTL** (los modos de cuenta fueron explicados a detalle en la práctica anterior).

OUTMOD_x(Bits 7-5) del Registro TACCTL_x.

000 -> Modo Salida. En este modo la salida es igual al contenido en el bit **OUT_x** del mismo registro (TACCTL_x).

001 -> Modo Set. La salida pasa a 1 cuando el registro TAR cuenta hasta el valor de TACCRx y permanece así hasta que TAR sea reiniciado u otro modo de salida sea seleccionado.

010 -> Modo Toggle/Reset. La salida es comutada cuando el registro TAR cuenta hasta el valor de TACCRx y es reiniciada cuando TAR llega al valor de TACCR0.

011 -> Modo Set/Reset. La salida se pone en '1' cuando la cuenta en el registro TAR llega al valor de TACCRx y se pone en '0' cuando la cuenta en TAR llega al valor de TACCR0.

100 -> Modo Toggle. La salida es comutada cuando la cuenta en el registro TAR llega al valor de TACCRx, el periodo de la salida es el doble del valor de TACCRx.

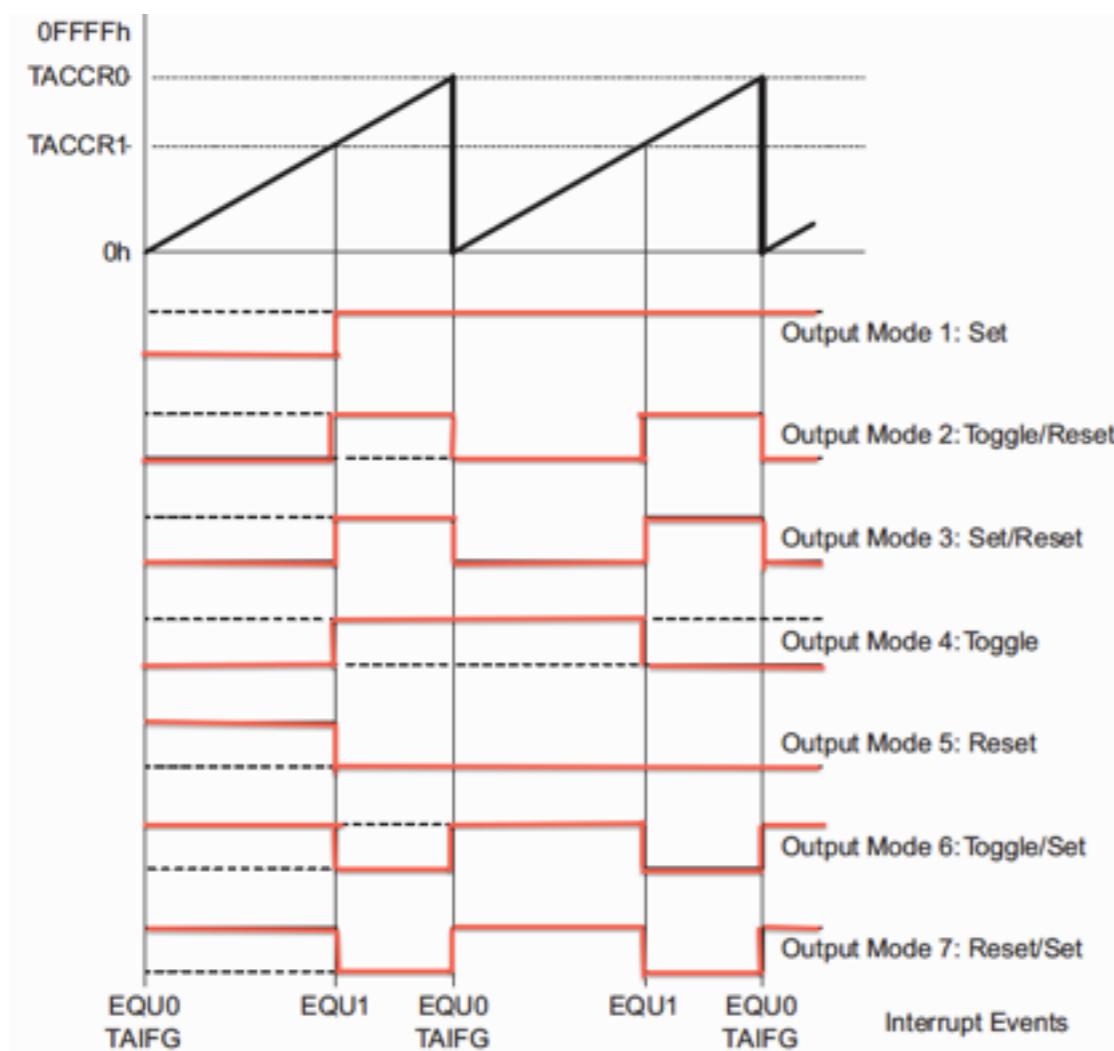
101 -> Modo Reset. La salida se pone en '0' cuando la cuenta en el registro TAR llega al valor de TACCRx y permanece así hasta que otro modo de salida sea seleccionado.

110 -> Modo Toggle/Set. La salida es comutada cuando la cuenta en el registro TAR alcance el valor de TACCRx, y es puesta a 1 cuando TAR llegue al valor de TACCR0.

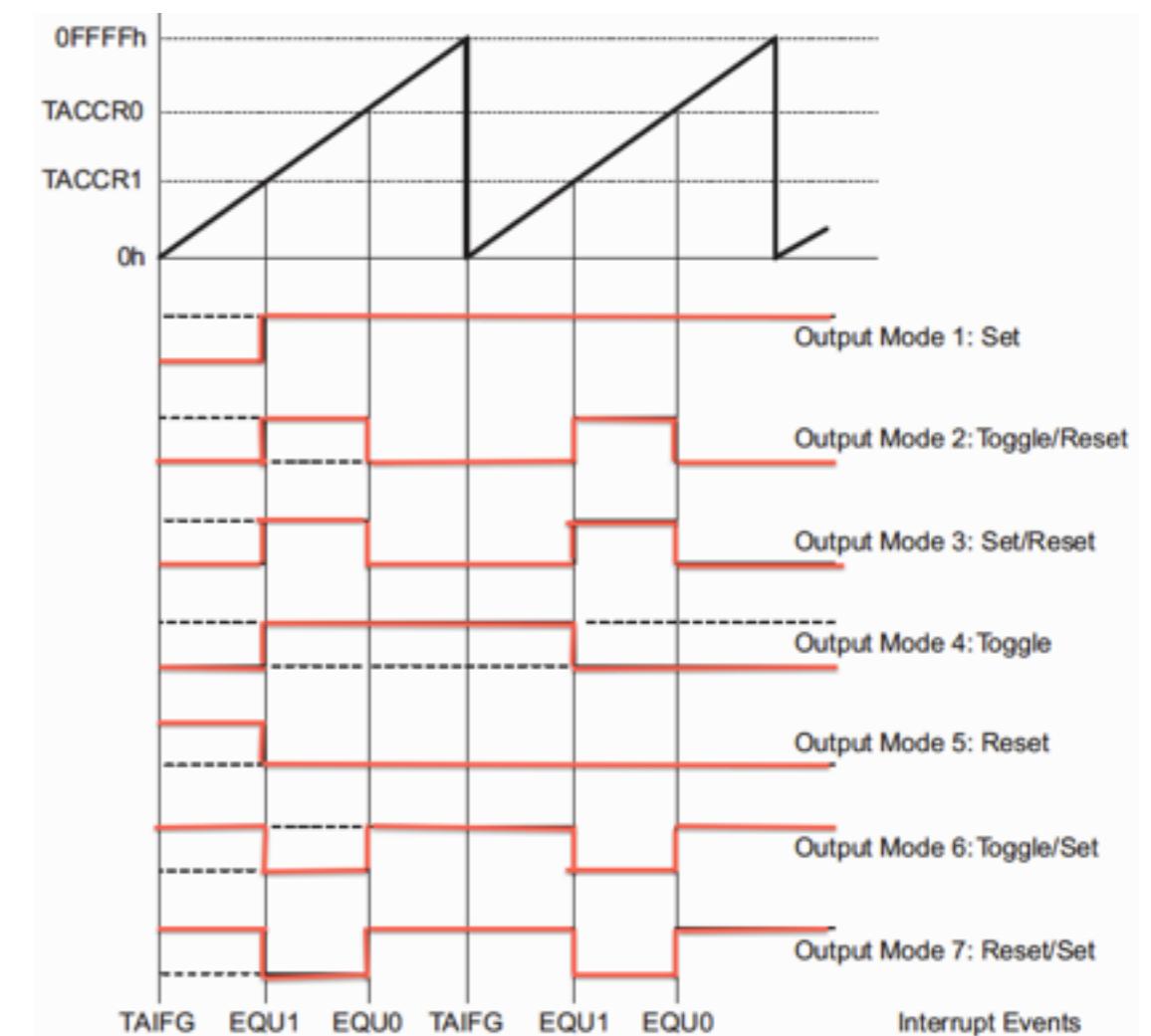
111 -> Modo Reset/Set. La salida es reiniciada cuando la cuenta en el registro TAR llega al valor en TACCRx, y se pone en '1' cuando TAR llega al valor de TACCR0.

Las siguientes imágenes muestran el comportamiento de los modos de salida en comparación según el modo de cuenta seleccionado.

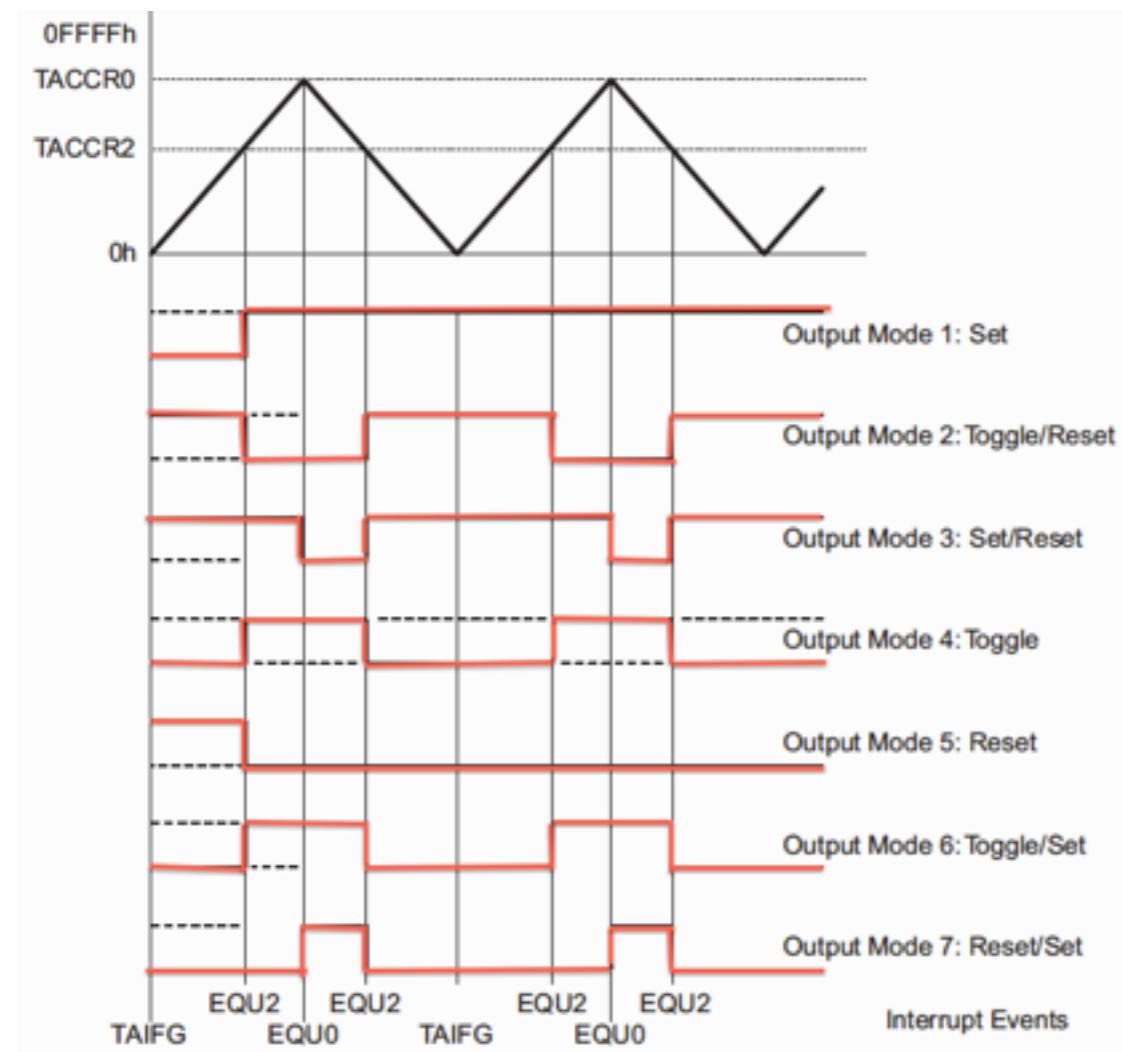
1. Modo Ascendente (Up)



2. Modo Continuo



3. Modo Ascendente/Descendente (Up/Down)



Configuración del Timer_A en comparación para generar una señal de PWM

El Timer_A tiene tres canales, aunque el canal 0 básicamente está perdido para muchas aplicaciones, porque su registro TACCR0 es necesario para limitar el periodo en los modos ascendente(Up) y ascendente/descendente (Up/Down). Cada canal es controlado por su registro TACCTLx.

Para la salida de la señal generada por el Timer_A en comparación, como puede ser el PWM, se debe de configurar el bit indicado como salida del Timer_A con el registro **PxSEL** como se ha manejado en prácticas anteriores. Por ejemplo el bit 2 del puerto 1, como segunda función maneja la salida del Timer0_A, si se utiliza este pin se configura esta segunda función poniendo un 1 en el bit2 del registro P1SEL: P1SEL=00000100 o bien P1SEL=BIT2.

El periodo de la señal del Timer_A se escribe en TACCR0 en el modo ascendente (Up), lo cual ofrece un control preciso del periodo.

El registro TAR cuenta desde cero ascendentemente hasta el valor en TACCR0 y regresa a cero para empezar un nuevo ciclo en la siguiente transición del reloj. Por lo tanto el periodo de la señal es $TACCR0+1=\text{Periodo de la señal}$.

En el registro TACCR1 se pone el ciclo de trabajo que se desea. Por ejemplo si TA0CCR0=100 (periodo de la señal), y se quiere un ciclo de trabajo del 50% en TA0CCR1 se pone 50 (TACCR1=50).

La bandera CCIFG0 se activa cuando TAR cuenta hasta TACCR0, mientras que la bandera TAIFG se activa cuando TAR regresa a cero, un ciclo de transición después. Por otra parte la bandera CCIFG1 se activa cuando TAR cuenta hasta TACCR1.

La idea detrás del PWM es que la ‘carga’ se prende y se apaga periódicamente, de modo que el voltaje promedio sea el voltaje deseado.

El ciclo de trabajo (Duty Cycle (D)) es la fracción de tiempo que la ‘carga’ está prendida o activa.

El voltaje promedio a través de la salida está dado por:

$$V_{prom} = D * V_{cc} = \frac{T_{ON}}{T_{ON} + T_{OFF}} V_{cc} = \frac{T_{ON}}{\text{Periodo}} V_{cc} = \frac{\text{Ancho de pulso}}{\text{Periodo}} V_{cc}$$

Donde D es el ciclo de trabajo ó Duty Cycle y está dado por:

$$D = \frac{T_{ON}}{T_{ON} + T_{OFF}} = \frac{\text{Ancho de pulso}}{\text{Periodo}}$$

PWM Simple

El arreglo usual para el PWM es aquel en el que la salida es puesta en activa ó en encendida cuando el registro de cuenta **TAR** regresa a ‘0’ (una vez que llegó a **TACCR0**) y se desactiva o se apaga cuando la cuenta en TAR llega al valor del Ciclo de Trabajo en el registro **TACCRx**, es importante volver a destacar que sólo se puede usar para el ciclo de trabajo TACCR1 ó TACCR2 ya que el canal 0 TACCR0 no se puede usar porque es el que da el periodo de la señal. Entonces si el ciclo de trabajo es el tiempo en el que la señal está encendida, cuando **TAR** llega a este valor que se guarda en el registro **TACCRx** la señal se apaga o se desactiva y se vuelve a encender hasta que llegue a **TACCR0** y regrese a cero, lo cual nos da el periodo.

Siendo que **TACCR0** es el periodo de la señal y **TACCRx** (donde $0 < x \leq 2$) es el tiempo que la señal está activa, la ecuación del ciclo de trabajo con estos registros es:

$$D = \frac{T_{ON}}{T_{ON} + T_{OFF}} = \frac{\text{Ancho de pulso}}{\text{Periodo}} = \frac{\text{TACCR}x}{\text{TACCR}0 + 1}$$

Con lo explicado anteriormente, se deduce que el modo de cuenta que se debe de elegir es el **modo ascendente (Up mode)**.

A continuación se muestra una imagen ilustrativa del ciclo de trabajo y el periodo de una señal de PWM.

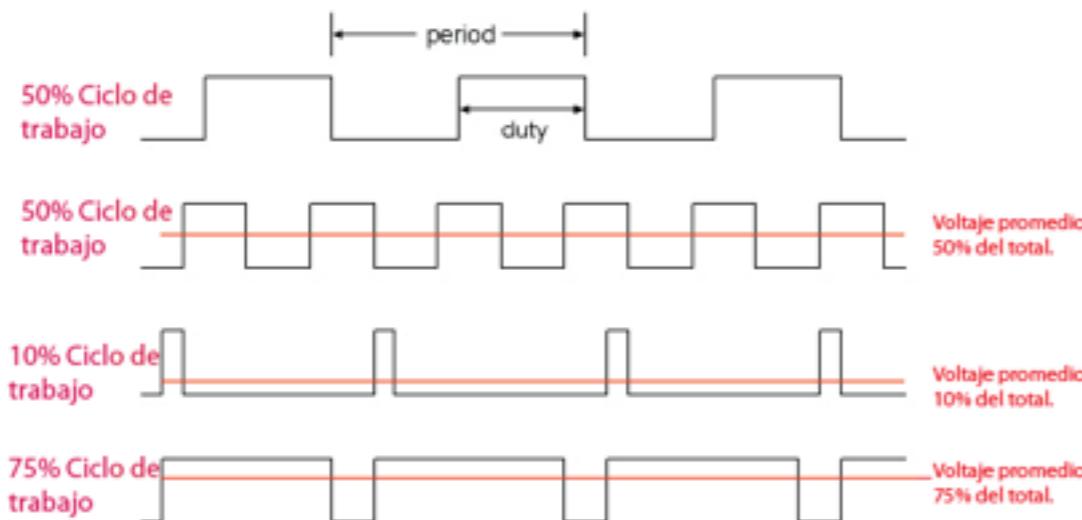


Figura 7.1. Señal de PWM, periodo de la señal y ciclo de trabajo (Duty).

Diseño de PWM

Existen dos parámetros principales que deben ser elegidos antes de seleccionar los valores que tendrá el PWM.

- 1.** El número de valores deseados del ciclo de trabajo (Resolución).
- 2.** La frecuencia de la señal de salida.

La frecuencia apropiada para la onda de salida de PWM depende del tipo de carga que se conectará.

El periodo del PWM es el mismo que el del Timer, y la frecuencia está dada por:

$$f_{PWM} = \frac{f_{reloj\ timer}}{\text{Periodo de TAR en cuentas}} = \frac{f_{reloj\ timer}}{\text{TACCR0} + 1}$$

Por ejemplo si la frecuencia del reloj del Timer es de 1MHz, y se requiere que la frecuencia del PWM sea de 100Hz el valor que se debe de poner en TACCR0 es:

$$\text{TACCR0} = \frac{f_{reloj\ timer}}{f_{PWM}} - 1 = \frac{1000000\text{Hz}}{100\text{Hz}} - 1 = 10,000 - 1 = 9,999$$

La frecuencia del reloj del Timer es de 1MHz, por lo que el periodo del reloj del Timer es de 1us. Si TACCR0 es el registro donde se pone el periodo de la señal de PWM en cuentas y tiene 10,000 cuentas, quiere decir que el periodo en segundos es igual al número de cuentas por el periodo del reloj del Timer, es decir: $(10,000 * 1\text{us}) = 0.01$ segundos, por lo que efectivamente da una frecuencia de 100Hz ($1/0.01$ segundos = 100Hz).

Hardware Externo

Servo-motor

Un Servo es un motor que permite el control del desplazamiento angular de su flecha . Este puede ser llevado a posiciones angulares específicas al enviar una señal modulada. Con tal de que una señal modulada exista en la línea de entrada, el servo mantendrá la posición angular del engranaje. Cuando la señal modulada cambia, la posición angular de la flecha cambia. Los Servo-motores son de gran utilidad en robótica, por ejemplo, para mover las articulaciones de un robot, entre otras aplicaciones. Además de su versatilidad es un motor que no consume mucha corriente.



modo de comparación del microcontrolador MSP430G2553.

¿Cómo trabaja un Servo-motor?

El servo cuenta con circuitos de control y un potenciómetro que es conectado al eje central del servo-motor. Este potenciómetro permite junto con los circuitos de control saber si el eje está en el ángulo correcto, de ser así el motor está apagado. Si el circuito de control detecta que el ángulo no es el correcto, el motor girará en la dirección adecuada hasta llegar al ángulo correcto. El eje del servo es capáz de llegar alrededor de los 180 grados. En algunos servos llega a los 210 grados. Un servo normalmente se usa para controlar un movimiento angular entre 0 y 180 grados.

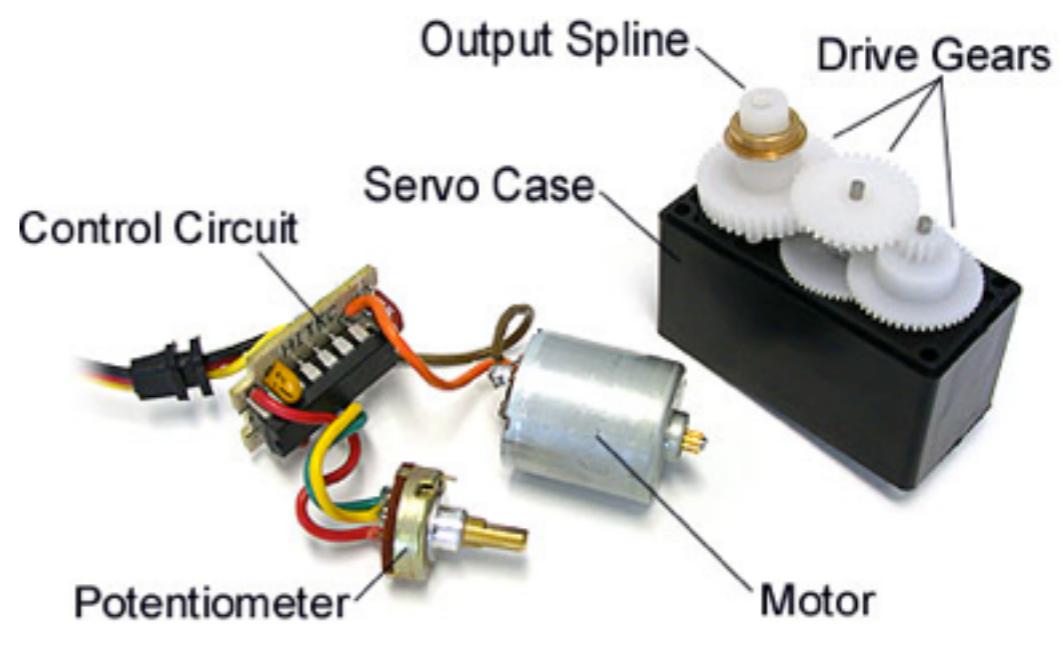


Figura 6.2. Servo-motor desarmado. Partes del servo-motor.

Los servos son controlados, enviándoles un ancho de pulso variable a el cable de control (naranja/amarillo/blanco) del servo. El ángulo en el que se posiciona el servo es determinado por la duración del pulso que es aplicado al cable de control del servo. Esto es llamado Modulación Por Ancho de Pulso (PWM). El servo espera tener un pulso cada 20ms. La longitud del pulso determinará que tanto el motor girará, o el ángulo en el que se posicionará. Por ejemplo, la posición neutral de un servo motor (90 grados) generalmente necesita un pulso de alrededor de 1.5 milisegundos (ms).

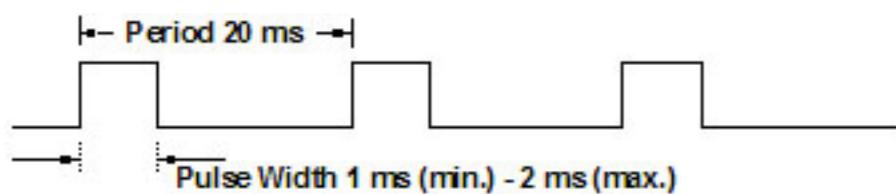


Figura 6.3. Señal de PWM con periodo de 20ms,un ancho de pulso de 1ms (Ciclo de Trabajo).

El mínimo ancho de pulso y el máximo ancho de pulso que necesita el motor para girar a posiciones válidas depende de cada servo-motor, pero generalmente, el mínimo ancho de pulso es de 1ms para llegar a la posición de 0 grados y el máximo ancho de pulso es de 2ms para llegar a la posición de 180°, por lo que para girar a la posición neutral, 90 grados, se requieren alrededor de 1.5ms.

Si el pulso es menor de 1.5 ms, el motor girará en dirección al ángulo de 0 grados, es decir, si el pulso es menor a 1.5ms, el motor se posicionará en ángulos menores a 90 grados hasta llegar a 0 grados. Si el pulso es mayor a 1.5ms el motor girará en dirección al ángulo de 180 grados, es decir, si el pulso es mayor a 1.5ms, el motor se posicionará en ángulos mayores a 90 grados.

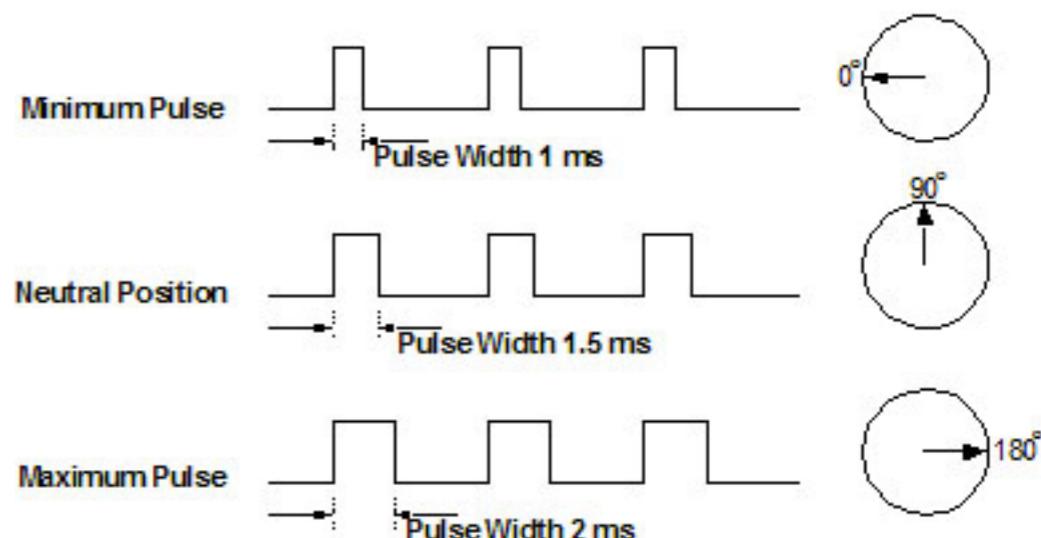


Figura 6.4. Mínimo y máximo ancho de pulso que necesitan la mayoría de los servos para funcionar.

Un periodo de 20ms implica una frecuencia de 50Hz($f=1/T=1/0.02=50Hz$).

Desarrollo de la práctica

1. Llevar acabo la conexión del hardware externo (TRIMPOT y Servo-motor) con el microcontrolador. La señal de PWM del bit 2 del puerto 1 (P1.2) se conecta mediante una resistencia de 1K-Ohm al cable (naranja/amarillo/blanco) de la señal de control del servo. El cable rojo del servo se conecta a una fuente externa de 5 Volts, y el cable marrón/negro a tierra. El TRIMPOT se conecta (como ya se vio en las prácticas anteriores) a la fuente de voltaje del microcontrolador (3.6 Volts) y la entrada analógica configurada para ADC es el bit 1 del puerto 1 (P1.1). La Tierra del servo, del TRIMPOT y del microcontrolador van juntas. El servo-motor que se usó en esta práctica es el DGSServo S06NF.

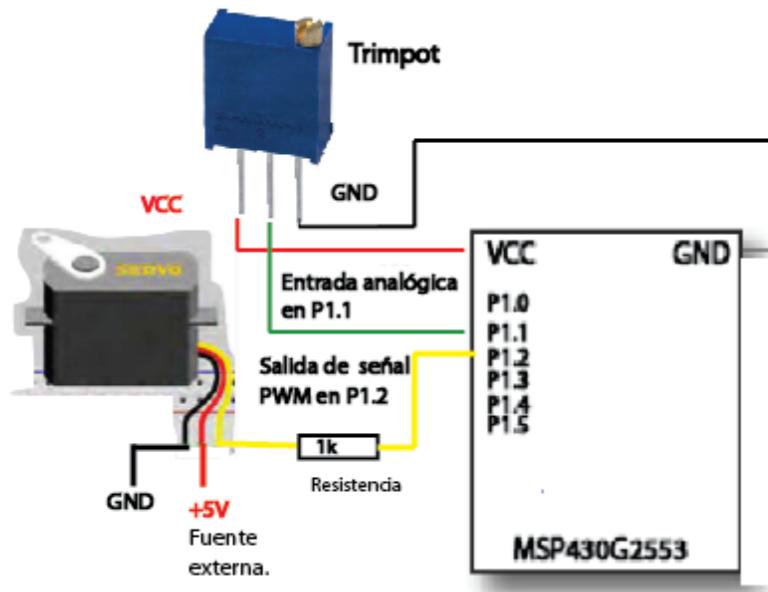


Figura 6.5.
Diagrama de
conexión.

2. Explicación del Programa a desarrollar:

En este programa, se utiliza la máxima frecuencia del reloj del Microcontrolador de 1MHz (1000000Hz) para alimentar el Timer_A. El periodo de la señal de salida PWM es de 20ms, es decir, tiene una frecuencia de 50Hz. Basándonos en las ecuaciones vistas en la sección de Metodología, quiere decir que el número de cuentas que se ponen en el Registro **TACCR0** para tener el periodo de 20ms es de $(1000000\text{Hz}/50\text{Hz}) = 20,000 - 1$ cuentas). El Ciclo de trabajo del PWM se obtiene del valor de la entrada analógica dada por **ADC10MEM**, a este valor se le hace un corrimiento dos veces a la izquierda, es decir, el valor de **ADC10MEM** se multiplica dos veces por dos, (éste es el efecto que produce el corrimiento a la izquierda). Por ejemplo si el valor de **ADC10MEM** es de 467 decimal y se le hace dos corrimientos a la izquierda este valor se convierte en $467 \times 2 = 934 \times 2 = 1868$. El valor obtenido al hacer dos veces el corrimiento en **ADC10MEM** se coloca en el registro **TA0CCR1** para que sea el ciclo de trabajo y así poder modificar éste a través del TRIMPOT, y cambiar la posición angular del servo. Se hace este corrimiento para que el giro del servo sea más rápido, es decir, para que se llegue más rápido al máximo y mínimo ancho de pulso que maneja el servo, aproximadamente (1ms para el mínimo ancho de pulso y 2ms para el máximo ancho de pulso).

El registro TA0CTL del Timer0_A se configura con el reloj SMCLK (1MHz) y con el modo de cuenta ascendente (Up). En el registro TA0CTL1 se configura el modo de salida de la onda de PWM como reset/set. En TA0CCR0 se pone el número de cuentas 20,000 -1, para tener un periodo de 20ms. El registro TA0CCR1, encargado del ciclo de trabajo del PWM, varía, según varíe la entrada analógica (según se varíe el TRIMPOT). La resistencia variable de 10K-Ohm (TRIMPOT) da el mínimo ancho de pulso para que el motor se posicione en 0 grados con el valor de aproximadamente 1K-ohm, y el máximo ancho de pulso para que el motor se posicione en 180 grados con el valor de aproximadamente 6K-Ohm.

El servo que se usó para realizar esta práctica es el DGservo S06NF.

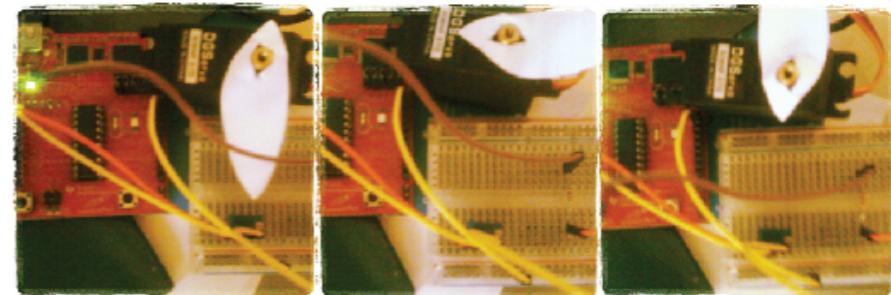
3. Copia y ejecuta el siguiente programa en C.

Varía el Trimpot y observa cómo se va modificando la posición angular del servo-motor, se recomienda poner un papel en el engrane para observar el movimiento giratorio con mayor facilidad. Puedes poner en pausa la ejecución del programa e ir observando el valor en los registros ADC10MEM y TA0CCR1, y ver con qué ancho de pulso da la mínima posición angular (0 grados) y con qué ancho de pulso da la máxima posición angular (180 grados ó 210 grados).

```
#include "msp430g2553.h"
//Conectar el cable de la señal del servo a P1.2 a través de una resistencia de 1k-ohm.
#define MCU_Reloj      1000000 //Reloj del Microcontrolador que alimenta el
                           // Timer_A-
#define PWM_FRECUENCIA 50      // Frecuencia de la señal de PWM deseada.
                           // Periodo de 20ms
unsigned int PWM_Periodo= (MCU_Reloj / PWM_FRECUENCIA); //Periodo en cuentas del PWM,
                           //Este valor se coloca en el
                           //registro TACCR0.
unsigned int PWM_Duty = 0; // Se limpia el valor del ciclo
                           // de trabajo.

//Programa Principal
void main (void){
    // Configuración del Timer0_A (PWM) y del ADC.
    WDTCTL = WDTTPN + WDTHOLD; // Detiene el Perro Guardián.
    BCSCCTL1= CALBC1_1MHZ; //Calibración del reloj del CPU a 1MHz
    DCOCCTL= CALDCO_1MHZ;
    TA0CTL1 = OUTMOD_7;
    TA0CTL = TASSEL_2 + MC_1;
    TA0CCR0 = PWM_Periodo-1;
    TA0CCR1 = PWM_Duty;
    P1DIR |= BIT2; // Se configura P1.2 como salida.
    P1SEL |= BIT2; // Se selecciona P1.2 como salida TA1
    ADC10CTL0 = ADC10SHT_2 + ADC10ON; // Tiempo de muestreo 16xADC10Clocks,habilita ADC10.
    ADC10CTL1 = INCH_1; // Se configura P1.1 como entrada analógica.

    for (;;)
    {
        ADC10CTL0 |= ENC + ADC10SC; // Inicio de muestreo y conversión.
        while (ADC10CTL1 & ADC10BUSY); // Pregunta si se está realizando una conversión.
        TA0CCR1 = ADC10MEM << 2; // Se toma el valor de la entrada analógica ADC10MEM y se
                           // hace dos corrimientos hacia la izquierda, es decir, se
                           // se multiplica dos veces por dos el valor del ADC10MEM
                           // Retardo
    }
}
```



4. Modificar el programa para generar una señal de PWM con una frecuencia de 1KHz y un ciclo de trabajo de 25%, después volverlo a modificar con un ciclo de trabajo de 50% y finalmente con un ciclo de trabajo de 75%, observar la señal de PWM en el osciloscopio con cada ciclo de trabajo.
5. Hacer reporte correspondiente a la práctica 7.

Práctica 8

Timer_A en modo de Captura

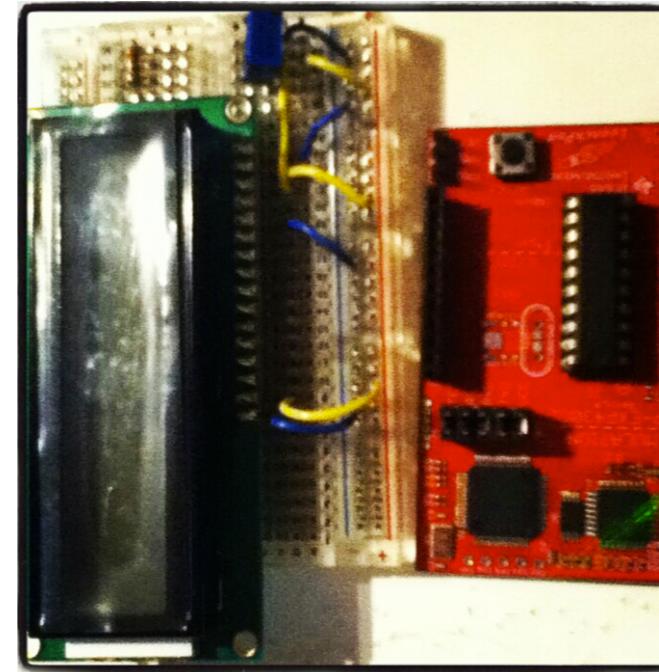
OBJETIVOS

1. Conocer el funcionamiento y configuración del Timer_A en modo de captura.
2. Aprender el funcionamiento, configuración y conexión del LCD.
3. Generar una señal de PWM con el TIMER1_A en modo comparación, medir su frecuencia con el TIMER0_A en modo captura, y mostrar esta frecuencia en el LCD.

MATERIALES

- 1 Microcontrolador MSP430G2553.
- 1 LCD de 16x2.
- 1 Trimpot 10K-ohms.
- 1 Header de una sola fila de 16 pines.
- 1 Fuente externa de 5 volts.
- Cables.

En las prácticas anteriores (6 y 7) ya se ha estado estudiando sobre el Timer_A, sobre sus registros, sobre su configuración y funcionamiento como temporizador y en modo de comparación, lo último que resta ver sobre este Timer_A es su funcionamiento en el modo de captura. En esta práctica se abordará este restante tema.



Esta práctica consiste en generar una señal PWM de 100Hz con el Timer1_A en modo de comparación, leer la frecuencia de esta señal generada con el Timer0_A en modo de captura y mostrar la frecuencia medida en el LCD.

Con el bit 1 del puerto 2 (P2.1) se genera la señal PWM de 100Hz inicialmente con un ciclo de trabajo del 50%, mientras que el bit 2 del puerto 1 (P1.2) se configura como entrada de captura. Por lo que la señal de

PWM generada en P2.1 se conecta a la entrada de captura P1.2, de este modo se puede comprobar que realmente el Timer0_A este trabajando adecuadamente. Posteriormente se puede jugar con los valores de frecuencia de la señal de PWM y observar que efectivamente en el LCD se muestra la frecuencia que se generó con el PWM, indicando que la captura de la señal se ha efectuado correctamente.

Metodología

Medición en el Modo de Captura

El modo de captura se usa para tomar el tiempo de un evento, es decir, para saber el tiempo en que el evento ocurrió. La medición generalmente requiere de dos o más captura.

En la mayoría de los casos el reloj que alimenta al Timer es ACLK ó SMCLK, cuya frecuencia ya es conocida, y una señal desconocida se aplica a la entrada de captura.

Para medir la longitud de un sólo pulso, se deben de capturar el flanco de subida (donde empieza el pulso) y el flanco de bajada (donde termina el pulso) y restar los tiempos de captura para conocer el tiempo que duró el pulso. La duración del pulso está en unidades del periodo del reloj que alimenta el Timer.

Para señales periódicas sólo es necesario capturar los flancos de subida (o de bajada si se prefiere) y la diferencia entre las capturas da directamente el periodo.

NOTA: El periodo del reloj que alimenta el Timer debe de ser mucho menor que la duración de la señal para tener una buena resolución.

Configuración del Timer_A en Modo de Captura

Para configurar el Timer_A en modo captura se utilizarán los registros vistos en las prácticas anteriores. Por lo que se recomienda revisar la sección de Metodología de las prácticas 6 y 7.

El registro **TACTL** se configura con el modo de cuenta preferentemente continuo, ya que con este modo es más fácil calcular las diferencias de los tiempos mientras TAR sigue con la cuenta. Se selecciona el reloj que alimenta el Timer_A en esta práctica se seleccionó el SMCLK (1MHz) y se limpia el timer.

En el registro **TACCTLx** se habilita el modo captura, se define en qué momento se va a llevar a cabo la captura (en el flanco de subida, en el flanco de bajada o en ambos flancos), y se habilitan las interrupciones, de modo que cada vez que se tenga una captura se llame a una Rutina de Servicio de Interrupción (ISR) para que se ejecuten ciertas acciones, además se recomienda que se habilite la fuente de captura sincronizada en el bit SCS de este registro.

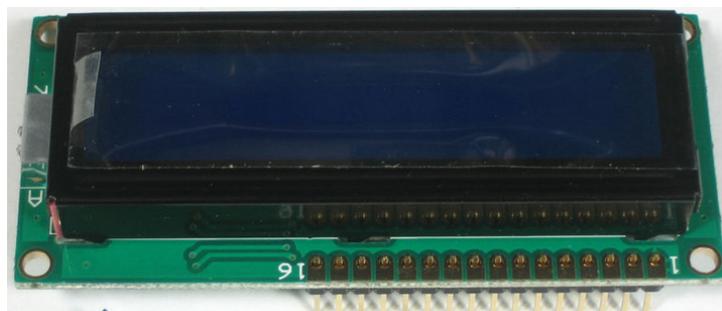
El registro de cuentas **TAR** va a estar contando, cuando ocurra una captura el valor de **TAR** se pasa al registro **TACCRx** y **TAR** continua con la cuenta hasta 0xFFFF si es que se selecciona el modo de cuenta continuo.

Se debe configurar el bit de entrada de captura como entrada en **PxDIR** y habilitar su función como entrada de captura del Timer_A con **PxSEL**. Suponiendo que se elige como entrada de captura (de acuerdo a la datasheet del MSP430G2553) al bit 2 del puerto 1 (P1.2) en el registro **P1DIR** se debe de poner un 0 en el bit 2, y en el registro **P1SEL** se pone un '1' en el bit 2.

Hardware Externo: LCD

Un LCD (Liquid Cristal Display) es una pantalla de cristal líquido que permite la visualización de caracteres. En un LCD de 2x16, se pueden visualizar dos líneas o filas de 16 caracteres cada una, es decir, $2 \times 16 = 32$ caracteres. Tiene un consumo de energía de menos de 5mA, se utilizan cuando se requiere una visualización pequeña o mediana.

Los MSP430 manejan un voltaje de 3.3-3.6 Volts, si se cuenta con un LCD que ocupe 5 Volts se debe de usar una fuente externa de 5 Volts.



El LCD cuenta 16 pines para su configuración, como se observa en la imagen.

Pin 1: Este pin va a tierra GND.

Pin 2: Este pin va a Vcc (+5Volts).

Pin 3: Voltaje de contraste, se conecta a un potenciómetro.

Pin 4: RS, A este pin se le envía un '0' lógico si se desea enviar un comando, y un '1' si se desea enviar un dato.

Pin 5: RW. Si se desea leer se conecta a Vcc (poco usual) y si se desea escribir en el LCD se conecta a tierra GND.

Pin 6: En, Este pin habilita el LCD si se le envía un '1' lógico.

Pin 7: D0. Pin de dato.

Pin 8: D1. Pin de dato.

Pin 9: D2. Pin de dato.

Pin 10: D3. Pin de dato.

Pin 11: D4. Pin de dato.

Pin 12: D5. Pin de dato.

Pin 13: D6. Pin de dato.

Pin 14: D7. Pin de dato.

Pin 15: LED-Ánodo: Luz de fondo . Se conecta a Vcc.

Pin 16: LED-Cátodo: Luz de fondo. Se conecta a GND.

El LCD cuenta con 11 líneas de bus : D0-D7, RS, RW y EN que a continuación se explican.

D0-D7(Pin 7-14): Son los pines en los que se manda el dato que queremos desplegar en el display, o bien el comando para posicionar el cursor, limpiar la pantalla, etc.

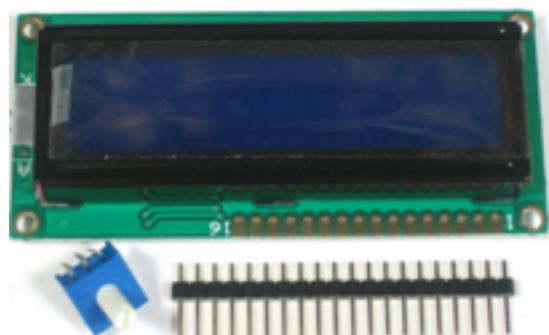
RS (Pin 4): Este pin le dice al LCD si el byte recibido es un dato o un comando.

RW (Pin 5): Este pin le dice al LCD si se quiere leer (no-común) o si se quiere escribir en éste (común).

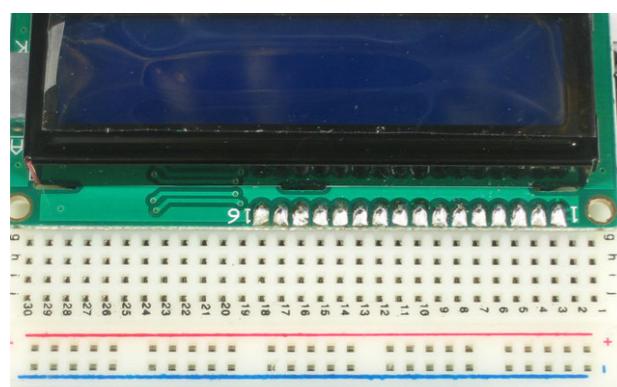
En (Pin6): Habilita el LCD. Este pin se usa para decirle al LCD que el dato está listo para leerse.

¿Cómo conectar el LCD?

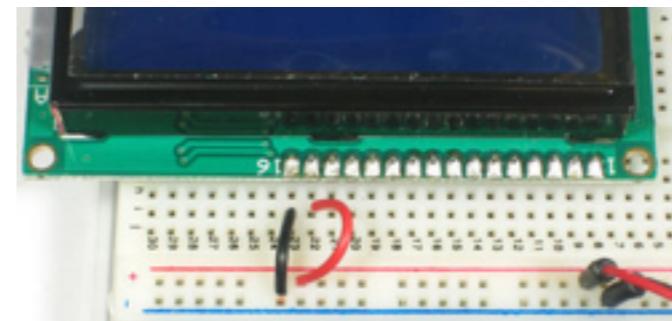
1. Solder el header de 16 pines al LCD.



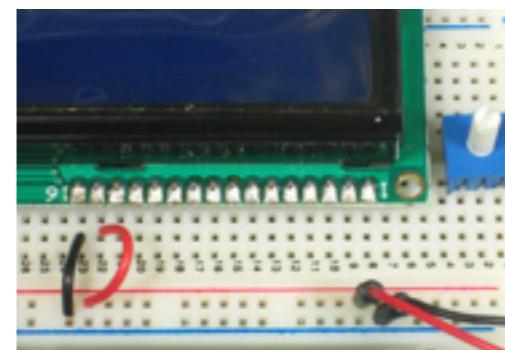
2. Colocar el LCD en una placa proto-board.



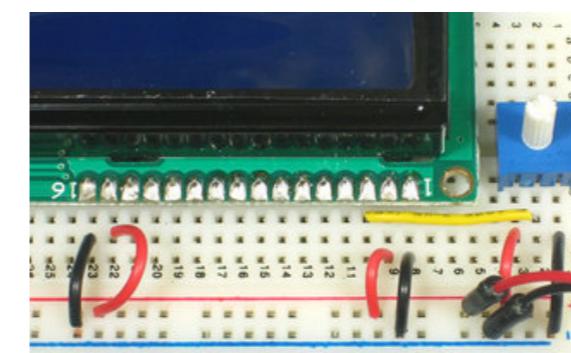
3. Conectar el Pin 16 a Tierra (GND) y el pin 15 a +5V (Vcc).



4. Posicionar el TRIMPOT para ajustar contraste cerca del Pin 1.



5. Conectar un lado del TRIMPOT a +5V y el otro a Tierra (GND). La pata de en medio del TRIMPOT conectarla al pin 3 del LCD, para ajustar contraste.



Además Conectar el pin 1 a Tierra y el pin 2 a +5V.

6. Conectar la fuente de voltaje externa +5V. Y ajustar el TRIMPOT hasta que aparezca la fila de rectángulos.

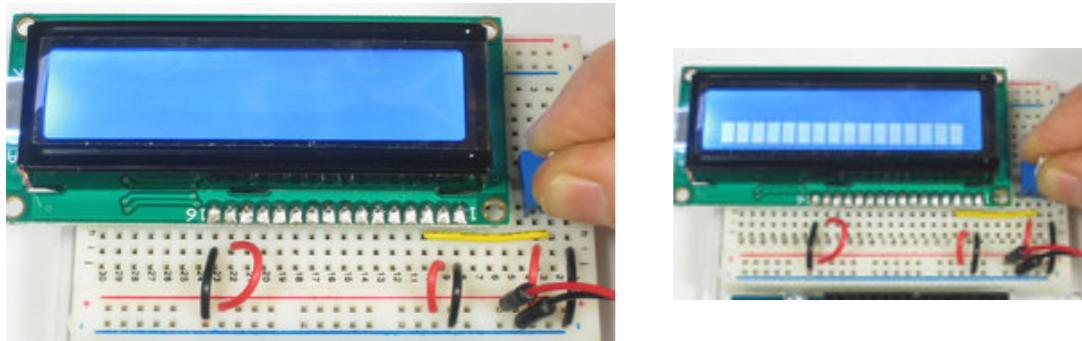
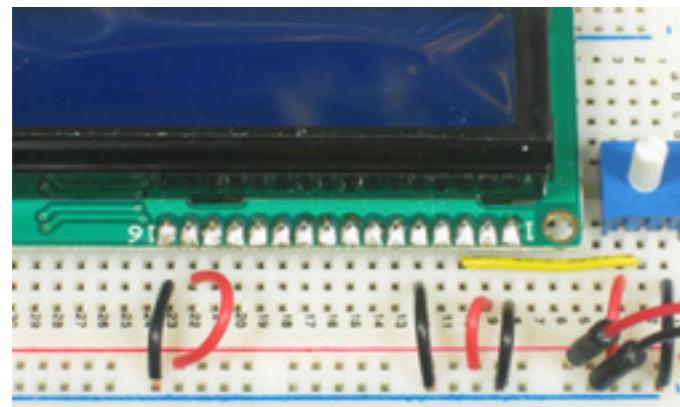


Tabla 8.1. Conexión del LCD con el MSP430G2553.

LCD Pin	LCD Función	MSP430G2553 Pin
Pin 4	RS	P1.0
Pin 6	EN	P1.1
Pin 14	D7	P1.7
Pin 13	D6	P1.6
Pin 12	D5	P1.5
Pin 11	D4	P1.4

7. Conectar el pin 5 RW a tierra ya que en esta práctica sólo se escribirá en el LCD.



En el programa proporcionado en la sección de Desarrollo de Práctica, se utilizaron sólo 4 pines de Datos (D7,D6,D5 y D4) que se conectan al Puerto 1 junto con el pin EN y el pin RS como se muestra en la tabla 8.1 y figura 8.1.

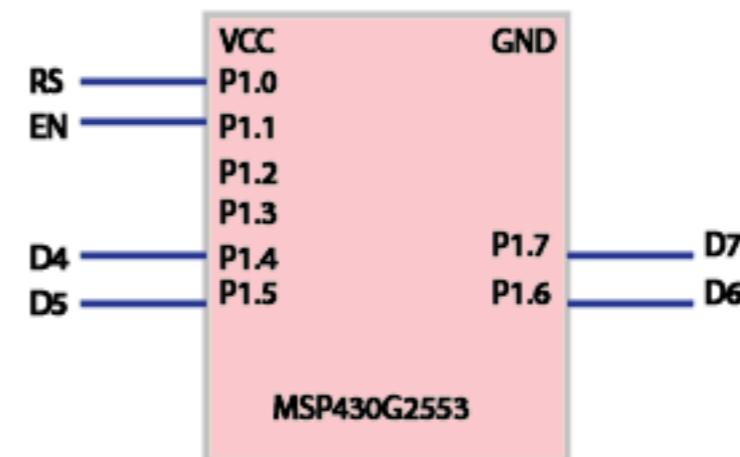


Figura 8.1. Conexión del LCD con el MSP430G2553.

*El tutorial de la conexión del LCD se tomó de:

<http://learn.adafruit.com/character-lcds/wiring-a-character-lcd>

Desarrollo de la práctica

1. Conectar el LCD con el MSP430G2553 como se describe en la sección de metodología. No olvides conectar la tierra del microcontrolador con la tierra de la fuente externa y el LCD.
2. Una vez que se ha conectado lo correspondiente al LCD. Conecta P2.1 que es la salida de la señal de PWM de 100Hz (con ciclo de trabajo de 50%) a P1.2 que es el bit que se configuró para entrada de captura, como se muestra en la figura 8.2.

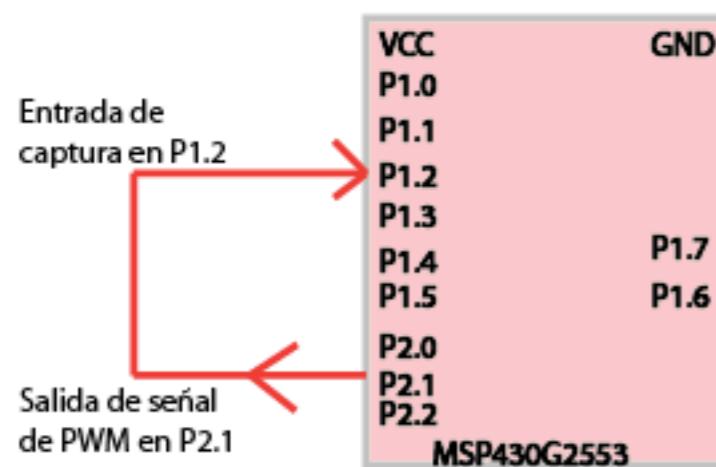


Figura 8.2. Conexión entre la señal de PWM generada, y la entrada de Captura.

3. Explicación del Programa.

En el programa que se proporciona a continuación, primeramente se encuentran las funciones para utilizar el LCD, cada función dentro del programa se explica a detalle.

En el programa principal, se configura el Timer1_A1 en modo de comparación con el reloj SMCLK de (1MHz) , el modo de cuenta ascendente y el modo de salida 7 reset/set,para generar una señal PWM con una frecuencia de 100 Hz y un ciclo de trabajo del 50%, como la frecuencia que alimenta el Timer es de 1MHz y se quiere una frecuencia de PWM 100HZ, en base a las ecuaciones vistas en la práctica anterior el número de cuentas que se ponen en TA1CCR0 es 10000-1. Y en TA1CCR1 5000 porque es el 50% del ciclo de trabajo. Esta señal de PWM sale por bit 1 del puerto 2.

El Timer0_A1 se configura en modo captura, con el reloj SMCLK (1MHz) y el modo de cuenta continuo, se selecciona captura por flanco de subida y se habilitan interrupciones para poder medir la frecuencia de la señal de PWM. Cada que existe una captura por flanco ascendente se llama a la Rutina de Servicio de Interrupción, la cual espera a que se tenga una segunda captura para hacer la resta de los tiempos de captura, el resultado de la resta se guarda en la variable aux. Se multiplica aux por (1/1MHz=0.000001) y se obtiene

el periodo de la señal a medir, pero como se busca a frecuencia se le saca el inverso al periodo y se obtiene la frecuencia, valor que se imprime en el LCD. Para la entrada de captura se configuró el bit 2 del puerto 1 (P1.2).

4. Copiar el siguiente programa y ejecutarlo.

```
#include "msp430g2553.h"

#define LCD_DIR P1DIR
#define LCD_OUT P1OUT
//Mapeo de los pines para el LCD
#define LCD_PIN_RS BIT0 // P1.0
#define LCD_PIN_EN BIT1 // P1.1
#define LCD_PIN_D7 BIT7 // P1.7
#define LCD_PIN_D6 BIT6 // P1.6
#define LCD_PIN_D5 BIT5 // P1.5
#define LCD_PIN_D4 BIT4 // P1.4

#define LCD_PIN_MASK ((LCD_PIN_RS | LCD_PIN_EN | LCD_PIN_D7| LCD_PIN_D6 | LCD_PIN_D5 | LCD_PIN_D4))

#define COMANDO 0
#define DATO 1
unsigned char d1= 0x00 | 0x30,d2= 0x00 | 0x30,d3= 0x00 | 0x30,d4= 0x00 | 0x30,d5= 0x00 | 0x30;
unsigned char samplestate=0;
unsigned char sampleok=0;
unsigned int sample1, sample2, bynbyte;
unsigned int aux;

unsigned int bynbyte, v, xx, xxx;

unsigned int bynbyte, v, xx, xxx;
// Función de parpadeo de cursor.
// Esta función se llamará cuando se requiera
// enviar un dato o un comando.
void PulseLCD()
{
    // Desactiva el bit EN
    LCD_OUT &= ~LCD_PIN_EN;
    __delay_cycles(200);
    //Activa el bit En
    LCD_OUT |= LCD_PIN_EN;
    __delay_cycles(200);
    //Vuelve a desactivar el bit EN
    LCD_OUT &= (~LCD_PIN_EN);
    __delay_cycles(200);
}
```

```
// Esta función envía un byte al LCD ya sea
// un comando o un dato, se utilizan sólo cuatro
// bits para enviarlo, por lo que es necesario
// mandar el byte en dos partes. Primero el nibble
// alto y luego el nibble bajo.
//
// Parámetros:
//   ByteParaEnviar - El byte que se quiere enviar el
//                     LCD
//
//   DatCom - Se pone DATO si el byte es un carácter o
//             dato y se pone COMANDO si es un comando.
void EnviarByte(char ByteParaEnviar, int DatCom)
{
    //Limpiar todos los pines de salida.
    LCD_OUT &= (~LCD_PIN_MASK);
    //

    //Activa el Nibble Alto enmascarando
    //sólo los bits P1.7-P1.4 (DB7-DB4)
    LCD_OUT |= (ByteParaEnviar & 0xF0);

    if (DatCom == DATO)
    {
        //Si es dato manda un '1' al pin 4 del LCD
        //para decirle que es un dato.
        LCD_OUT |= LCD_PIN_RS;
    }
    else
    {

        //
        //Si es comando manda un '0' al pin 4 del LCD
        //para decirle que es un comando.
        LCD_OUT &= ~LCD_PIN_RS;
    }

    //Una vez que ya se configuró el pin 4 del LCD (RS)
    //como dato ó comando, es momento de leer el byte.
    PulseLCD();
}
```

```

//Limpia todos los bits de salida
LCD_OUT &= (~LCD_PIN_MASK);
//Activa el nibble bajo, enmascarando los 4 bits menos
//significativos y haciendo 4 corrimientos a la izquierda.
LCD_OUT |= ((ByteParaEnviar & 0x0F) << 4);

if (DatCom == DATO)
{
    LCD_OUT |= LCD_PIN_RS;
}
else
{
    LCD_OUT &= ~LCD_PIN_RS;
}

PulseLCD();
}

// Función para poner el cursor en una posición específica.
// Parámetros:
//
// Fila- Si Fila es cero la posición es la primer fila.
// Si Fila es uno la posición del cursor es la
// segunda fila.
//
// Col - Poner el número de la columna en que se quiera
// posicionar el cursor, desde la columna cero
// hasta 15. (0-15).

void LCDPosicionarCursor(char Fila, char Col)
{
    char address;
    //la variable address se forma con el valor de Fila y Col.
    if (Fila == 0)
    {
        address = 0;
    }
    else
    {
        address = 0x40;
    }

    address |= Col;

    EnviarByte(0x80 | address, COMANDO);
}

// Función Para limpiar Pantalla.
//Limpia la pantalla y pone al cursor en su posición base.

void LimpiaPantallaLCD()
{
    // Limpia Pantalla
    EnviarByte(0x01, COMANDO);
    // Pone Cursor en posición base.
    EnviarByte(0x02, COMANDO);
}

// Función para inicializar el LCD.
// Initialize the LCM after power-up.
// Esta función no debe ser llamada dos veces por
// el programa principal.

void InicializarLCD(void)
{
    //Limpia todos los bits del LCD.
    LCD_DIR |= LCD_PIN_MASK;
    LCD_OUT &= ~(LCD_PIN_MASK);

    //Retardo para que el LCD active sus pines.
    delay_cycles(100000);
    //Inicialización del LCD
    // Activar la entrada de 4-bits
    LCD_OUT &= ~LCD_PIN_RS;
    LCD_OUT &= ~LCD_PIN_EN;

    LCD_OUT = 0x20;
    PulseLCD();

    EnviarByte(0x28, COMANDO);

    //Activar LCD y Cursor
    EnviarByte(0x0E, COMANDO);

    //Posición de cursor se auto-incremente.
    EnviarByte(0x06, COMANDO);
}

```

```

// Función para Imprimir un String
// Imprime una cadena de caracteres en la
// pantalla
void ImprimirStr(char *Texto)
{
    char *c;
    c = Texto;

    while ((c != 0) && (*c != 0))
    {
        EnviarByte(*c, DATO);
        c++;
    }
}

// Función para Imprimir un entero
// En esta función un número entero se convierte
// a ASCII para poder imprimirla en el LCD.
// Primero se separa el número en unidades y se
// le suman 0x30 para convertirlo en ASCII y así hasta
// llegar a diezmiles.
void ImprimirEntero(unsigned int entero)
{
    bynbyte=entero;
    v = bynbyte/10;
    d1 = bynbyte % 10; //unidades
    d1 = d1 | 0x30;
    d2 = v % 10; //decenas
    d2 = d2 | 0x30;
    xx = v / 10;
    d3 = xx % 10; // centenas
    d3 = d3 | 0x30;
    xxx= xx/10;
    d4= xxx%10; //miles
    d4= d4 | 0x30;
    d5=xxx/10;
    d5= d5 | 0x30; //diezmiles

    EnviarByte(d5, DATO);
    EnviarByte(d4, DATO);
    EnviarByte(d3, DATO);
    EnviarByte(d2,DATO);
    EnviarByte(d1,DATO);
}

// Programa principal
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Detiene perro guardián.
    InicializarLCD();
    LimpiaPantallaLCD();
    ImprimirStr("PRACTICA 8:");
    //Configuración de los puertos
    P2DIR |= BIT1;
    P2SEL |= BIT1;
    P1DIR =~(BIT2);
    P1SEL|= BIT2;
    //Configuración de la señal de PWM
    TA1CCR0=10000-1;
    TA1CCTL1=OUTMOD_7;
    //Periodo de la señal de PWM.
    //Modo de salida en comparación
    // reset/set
    TA1CCR1=5000;
    TA1CTL=TASSEL_2+ID_0+MC_1+TACLR;
    // Configuración de la Captura
    TA0CCTL1=CM_1+CCIS_0+SCS+CAP+CCIE; //Captura en Flanco de subida, entrada
    //captura CCI1A, fuente de captura
    //Sincronizada e interrupciones.
    TA0CTL=TASSEL_2+ID_0+MC_2+TACLR; //Reloj=SMCLK, Modo de cuenta continuo.
    _BIS_SR(LPM0_bits + GIE); //Habilitar interrupciones globales.

    while (1)
    {
        __delay_cycles(100000);
    }
}

//Rutina de servicio de interrupción (ISR)
#pragma vector=TIMER0_A1_VECTOR
__interrupt void InterruptCapture (void)
{
    if(samplestate==0){
        sample1=TA0CCR1; //Muestra 1
        samplestate=1;
    } else if(samplestate==1){
        sample2=TA0CCR1; // Muestra 2
        samplestate=0;
        aux=sample2-sample1; //Muestra2-Muestra 1
        aux=1/(aux*0.000001); //Obtener frecuencia
        LimpiaPantallaLCD();
        ImprimirStr("Frecuencia:");
        LCDPosicionCursor(2,0);
        ImprimirEntero(aux); //Imprimir frecuencia medida.
        ImprimirStr(" Hertz");
        __delay_cycles(1000000);
    }
    TA0CCTL1 &=~CCIFG;
}

```



5. Cambiar la frecuencia del PWM a 800Hz con un ciclo de trabajo del 50 % poniendo TA1CCR0=1250 -1 y TA1CCR1=625.

6. Modifica el programa para que en lugar de que mida la frecuencia de la señal mida el ancho del pulso, es decir, el ciclo de trabajo.

Práctica 9

Comunicación Serial Síncrona: USCI en modo SPI

OBJETIVOS

1. Conocer el protocolo de comunicación SPI (Serial Peripheral Interface).
2. Aprender el funcionamiento y configuración del módulo USCI (Universal Serial Communication Interface) en modo SPI.
3. Establecer comunicación serial síncrona mediante el protocolo SPI entre dos microcontroladores MSP430G2553 uno configurado como Maestro (Master) y otro como Esclavo (Slave).

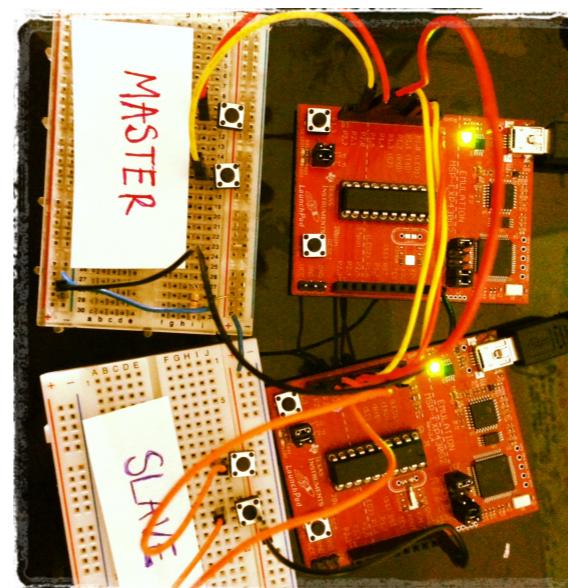
MATERIALES

2 Microcontroladores MSP430G2553.

4 Push-button.

Cables

En esta práctica se introduce a la comunicación serial mediante el protocolo de Interfaz de periféricos serie o bus SPI (Serial Peripheral Interface). Así como al módulo del microcontrolador MSP430G2553 USCI (Universal Serial Communication Interface) y su configuración como modo SPI.



Esta práctica consiste en programar un microcontrolador MSP430G2553 Launch-Pad como maestro (master) y otro MSP430G2553 Launch-Pad como esclavo (slave) a cada microcontrolador se le conectan dos push-buttons externos y se le configuran las resistencias internas de pull-up para cada push-button externo, cuando el push-button 1 conectado al microcontrolador configurado como Maestro se presiona se debe de

encender el LED2(Verde) conectado internamente en P1.6 del microcontrolador configurado como Esclavo, y cuando se presiona el push-button 2 conectado al microcontrolador Maestro se debe de encender el LED1(Rojo) conectado internamente en P1.0 del microcontrolador Esclavo. Del mismo modo cuando se presiona el push-button 1 del microcontrolador Esclavo se debe de encender el LED2(Verde) conectado internamente en P1.6 del microcontrolador Maestro y cuando se presiona el push-button 2 del microcontrolador Esclavo se debe de encender el LED1(Rojo) del microcontrolador Maestro.

Metodología

Serial Peripheral Interface (SPI)

El bus SPI ó Interfaz de periféricos serie, es un protocolo de comunicación serial local, cuya tarea principal es la transferencia de información entre circuitos integrados, equipos electrónicos, o bien permite que un microcontrolador se comunique con otro microcontrolador o que un mircoccontrolador se comunique con otros dispositivos electrónicos como sensores, convertidores, entre otros.

Operación del SPI

SPI es un protocolo de comunicación síncrono local, y la transmisión de datos se realiza por medio de cuatro señales (en algunos casos sólo con tres señales es suficiente). Cada una de estas señales porta la información entre los diferentes dispositivos conectados al bus. Cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo, por lo que este tipo de comunicación serial es full-duplex.

En este tipo de comunicación un dispositivo se configura como Maestro (Master) y otro como Esclavo (Slave) un dispositivo Maestro puede tener uno o varios esclavos conectados. El Maestro proporciona la señal de reloj a uno o varios esclavos.

Las señales de operación del SPI son SCLK, MOSI, MISO y SS>Select.

SCLK (Clock): Es el pulso que marca la sincronización. Con cada pulso se lee o se envía un bit . Esta señal la proporciona el dispositivo configurado como Master (Maestro). También se conoce en alemán como TAKT.

MOSI (Master Output Slave Input): Salida de datos del Maestro y entrada de datos al Esclavo. También llamado **SIMO**.

MISO (Master Input Slave Output): Entrada de datos al Maestro Salida de datos del Esclavo (Slave). También llamado **SOMI**.

SS>Select: Para seleccionar un esclavo, o para que el Maestro le diga al Esclavo que se active. También llamado SSTE, Chip Select. Si se cuenta con un sólo esclavo, esta línea puede omitirse, o bien, aún cuando se cuente con un sólo esclavo se puede habilitar. Cuando se tienen muchos esclavos esta señal indica con cual esclavo desea transferir datos.

La cadena de bits es enviada de manera síncrona con los pulsos del reloj, es decir con cada pulso, el Maestro envía un bit. Para que empiece la transmisión el Maestro baja la señal SSTE ó SS>Select a cero, con esto el esclavo se activa y

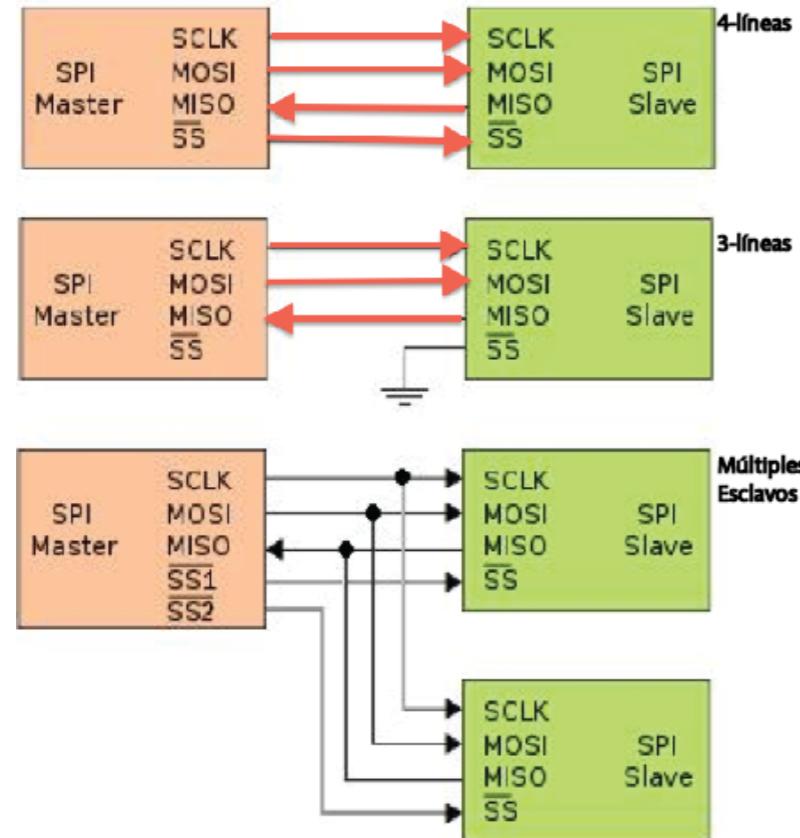


Figura 9.1.
Conexión
entre el
Maestro y
Esclavo en el
protocolo SPI.

CPHA=0 El dato es leído (o cambiado) en el primer flanco del reloj y escrito (capturado) en el siguiente flanco.

CPHA=1 El dato es escrito (capturado) en el primer flanco del reloj y leído (o cambiado) en el siguiente flanco.

CPOL= 0 El estado Idle de la línea de reloj está en bajo entre las transferencias. (El estado inactivo es bajo).

CPOL=1 El estado Idle de la línea de reloj está en alto entre las transferencias. (El estado inactivo es alto).

Tabla 9.1. Modos del Reloj SPI

Modo SPI	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Modos del Reloj

La mayoría de las interfaces SPI tienen 2 bits de configuración llamados CPOL (Clock Polarity, Polaridad de Reloj) y CPHA (Clock Phase, Fase del Reloj). CPOL determina si el estado **Idle** de la línea del reloj está en bajo (CPOL=0) o si se encuentra en un estado alto (CPOL=1). CPHA determina en qué flanco del reloj los datos son capturados o cambiados. En el Maestro se configura la polaridad y fase del reloj. La tabla 9.1 muestra los cuatro modos de reloj del SPI.

Configuración del módulo USCI en modo SPI

El módulo USCI (Universal Serial Communication Interface) soporta distintos modos de comunicación serial. Cada módulo diferente USCI se llama con una letra diferente USCI_A y USCI_B, y cada uno de estos módulos pueden tener otro módulo idéntico. Por ejemplo USCI_A0 y USCI_A1, o bien USCI_B0 y USCI_B1. El microcontrolador MSP430G2553 cuenta con los módulos USCI_A0 y USCI_B0 únicamente.

El módulo USCI_A0 contiene:

- Modo UART.
- Forma de pulso para comunicaciones IrDA.
- Detección Automática de velocidad de transmisión para comunicaciones LIN.
- Modo SPI.

El módulo USCI_B0 contiene:

- Modo I2C.
- Modo SPI.

En modo síncrono, el módulo USCI conecta al MSP430 con un sistema externo mediante tres o cuatro líneas: UCxSIMO, UCxSOMI, UCxCLK, y UCxSTE. El modo SPI se selecciona cuando el bit UCSYNC se activa, el modo SPI (3-pin ó 4-pin) se selecciona con los bits de UCMODEx.

Registros del módulo USCI en modo SPI

Se recomienda leer el capítulo de USCI: Modo SPI de la Guía de Usuario que proporciona el fabricante Texas Instruments.

Los registros del módulo USCI tienen la misma configuración para el módulo USCI_A0 y USCI_B0.

1. UCA0CTL0: USCI_A0 Control Register 0 (Registro de Control 0).
2. UCA0CTL1: USCI_A0 Control Register 1 (Registro de Control 1).
3. UCA0BR0: USCI_A0 Baud Rate Control Register 0 (Registro de Control de velocidad de transmisión 0).
4. UCA0BR1: USCI_A0 Baud Rate Control Register 1 (Registro de Control de velocidad de transmisión 1).
5. UCA0STAT: USCI_A0 Status Register (Registro de Estado).

6. UCA0MCTL: USCI_A0 Modulation Control Register (Registro de Control de modulación).

7. UCA0RXBUF: USCI_A0 Receive Buffer Register. (Registro del Buffer de Receptor).

8. UCA0TXBUF: USCI_A0 Transmit Buffer Register (Registro del Buffer de Transmisión).

9. IE2. Interrupt Enable Register 2. (Registro de Habilitación de Interrupciones 2).

10.IFG2. Interrupt Flag Register 2. (Registro de Bandera de Interrupción 2).

A continuación se explican los bits de cada registro, para el modo SPI.

El registro de modulación UCA0MCTL no se utiliza en el modo SPI, sin embargo, durante la programación se debe de limpiar este registro (UCA0MCTL=0).

1. **UCA0CTL0:** USCI_A0 Control Register 0 (Registro de Control 0).

7	6	5	4	3	2	1	0
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx	UCSYNC=1	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	

UCCKPH BIT 7

Selección de la Fase de Reloj.

0	El dato cambia en el primer flanco del reloj y se captura en el siguiente flanco.
1	El dato es capturado en el primer flanco del reloj y se cambia en el siguiente flanco.

UCCKPL BIT 6

Selección de Polaridad del Reloj

0	El estado inactivo es bajo.
1	El estado inactivo es alto.

UCMSB BIT 5

Seleccionar que primero se envíe el bit más significativo (MSB) o el bit menos significativo (LSB).

0	Primero LSB
1	Primero MSB

UC7BIT BIT 4 Longitud de Carácter. Selecciona la longitud del carácter de 7 u 8 bits.

0	Dato de 8 bits.
1	Dato de 7 bits.

UCMST BIT 3 Selección de modo Master o Slave.

0	Modo Slave(Esclavo).
1	Modo Master(Maestro).

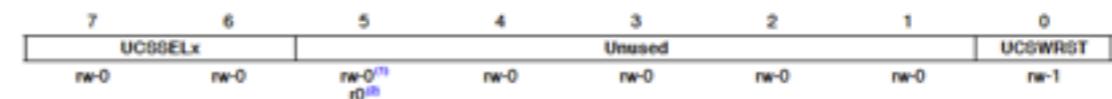
UCMODE BITS2-1 Modo de USCI.

00	3-pin SPI.
01	4-pin SPI. El pin de UCxSTE se activa en alto: Se habilita el esclavo cuando UCxSTE=1.
10	4-pin SPI. El pin de UCxSTE se activa en bajo: Se habilita el esclavo cuando UCxSTE=0.
11	Modo I2C.

UCSYNC BIT0 Habilitar Modo Síncrono.

0	Modo Síncrono.
1	Modo Asíncrono.

2. UCA0CTL1: USCI_A0 Control Register 1 (Registro de Control 1).



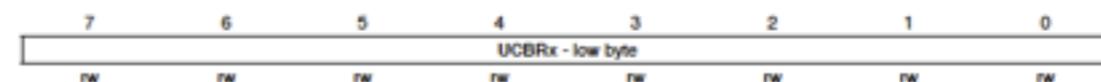
USSELx BITS 7-6 Selección de la Fuente de Reloj para el módulo USCI.

00	NA.
01	ACLK.
10	SMCLK.
11	SMCLK.

UNUSED BITS 5-1

UCSWRST BIT0 Habilita el Reset del software.

3. UCA0BR0: USCI_A0 Baud Rate Control Register 0 (Registro de Control de velocidad de transmisión 0).



4. UCA0BR1: USCI_A0 Baud Rate Control Register 1 (Registro de Control de velocidad de transmisión 1).



Configuración del preescalado del reloj. El valor de 16 bits de UCA0BR0+UCA0BR1*256 forma el valor del preescalado.

5. UCA0STAT: USCI_A0 Status Register (Registro de Estado).

UCLISTEN BIT 7 Habilita el modo LISTEN.

0	Deshabilitado.
1	Habilitado. La salida del transmisor es internamente alimentada al receptor.

UCFE BIT 6 Bandera de Error de Trama (Framing Error). Este bit indica un conflicto en el bus en el modo de 4-pin SPI. UCFE no se usa en el modo 3pin.

0	No hay Error.
1	Un conflicto con el bus ha ocurrido.

UCOE BIT5 Bandera de error por overrun. Este bit se activa cuando un carácter es transferido al registro UCxRXBUF antes de que el carácter anterior se haya leído.

0	No hay Error.
1	Error por overrun ha ocurrido.

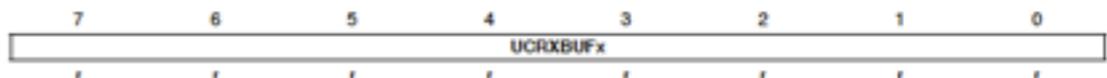
UNUSED BITS 4-1

UCBUSY BIT 0

USCI Ocupado. Este bit indica si la operación de transmisión o recepción está en proceso.

0	USCI inactivo.
1	USCI está Transmitiendo o recibiendo.

6. UCA0RXBUF: USCI_A0 Receive Buffer Register. (Registro del Buffer de Receptor).



UCRXBUFx BITS 7-0:

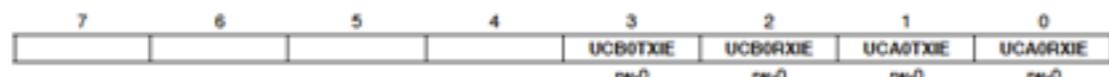
El buffer del receptor contiene el último carácter recibido del registro Receive Shift Register. Leyendo UCA0RXBUF limpia los bits de Error de Recepción y la bandera de interrupción UCA0RXIFG.

7. UCA0RXBUF: USCI_A0 Transmit Buffer Register. (Registro del Buffer de Transmisión).



UCTXBUFx BITS 7-0:

El dato en el buffer de transmisión sostiene al dato esperando a ser movido en el registro Transmit Shift Register y ser transmitido. Escribiendo en el buffer de transmisión de datos UCA0TXIFG se limpia la bandera de interrupción UCA0TXIFG.

8. IE2: Interrupt Enable Register 2. (Registro de Habilitación de Interrupciones 2).


BITS 7-4 Estos bits son usados en otros módulos.

UCB0TXIE BIT3 USCI_B0 Habitar Interrupción de transmisión.

0	Interrupción deshabilitada.
1	Interrupción habilitada.

UCB0RXIE BIT2 USCI_B0 Habitar Interrupción de recepción.

0	Interrupción deshabilitada.
1	Interrupción habilitada.

UCA0TXIE BIT1

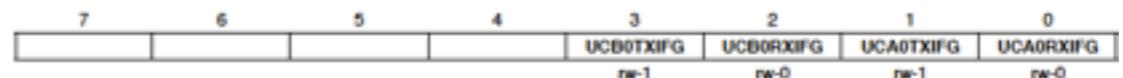
USCI_A0 Habitar Interrupción de transmisión.

0	Interrupción deshabilitada.
1	Interrupción habilitada.

UCA0RXIE BIT0

USCI_A0 Habitar Interrupción de recepción.

0	Interrupción deshabilitada.
1	Interrupción habilitada.

9. IFG2: Interrupt Flag Register 2. (Registro de Bandera de Interrupción 2).


BITS 7-4 Estos bits son usados en otros módulos.

UCB0TXIFG BIT3 USCI_B0 Bandera de Interrupción de transmisión.

0	No hay interrupción pendiente.
1	Interrupción pendiente.

UCB0RXIFG BIT2 **USCI_B0 Bandera de Interrupción de recepción.**

0	No hay interrupción pendiente.
1	Interrupción pendiente.

UCA0TXIFG BIT1 **USCI_A0 Bandera de Interrupción de transmisión.**

0	No hay interrupción pendiente.
1	Interrupción pendiente.

UCA0RXIFG BIT0 **USCI_A0 Bandera de Interrupción de recepción.**

0	No hay interrupción pendiente.
1	Interrupción pendiente.

Un pin adicional, UCxSTE es proporcionado para habilitar un dispositivo en modo Esclavo (Slave) para transmitir o recibir datos y es controlado por el Maestro (Master).

Tres o cuatro señales son usadas para el intercambio de datos SPI.

1. UCxSIMO: Slave In Master Out.

Modo Maestro: UCxSIMO es la línea del dato de salida.

Modo Esclavo: UCxSIMO es la línea del dato de entrada.

2. UCxSOMI: Slave Out Master In.

Modo Maestro: UCxSOMI es la línea del dato de entrada.

Modo Esclavo: UCxSOMI es la línea del dato de salida.

3. UCxCLK: Reloj USCI SPI

Modo Master: UCxCLK es una salida.

Modo Slave: UCxCLK es una entrada.

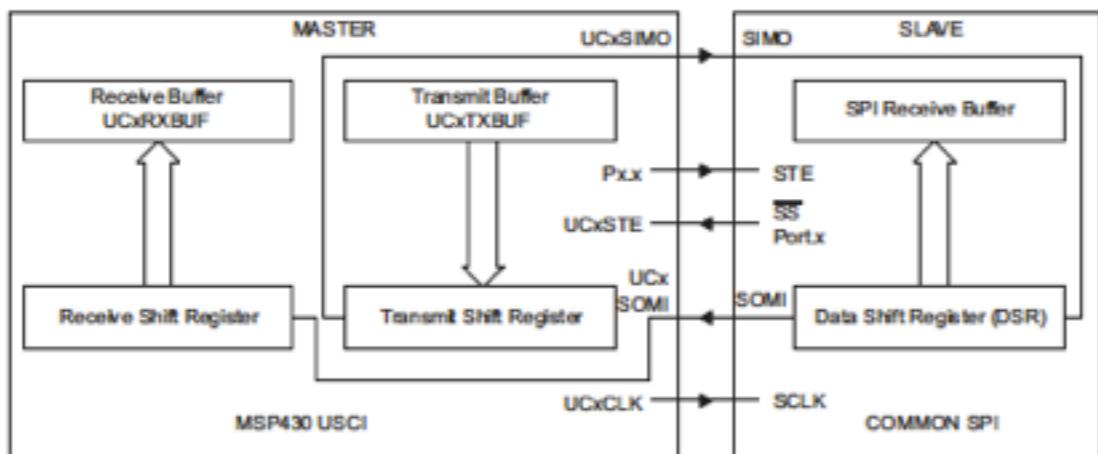
4. UCxSTE: Slave Transmit Enable. Habilitar Transmisión de Esclavo.

Utilizado en el modo 4-pin permite múltiples esclavos (slaves) en un sólo bus. No se usa en el modo 3-pin.

Operación del módulo USCI en el modo SPI

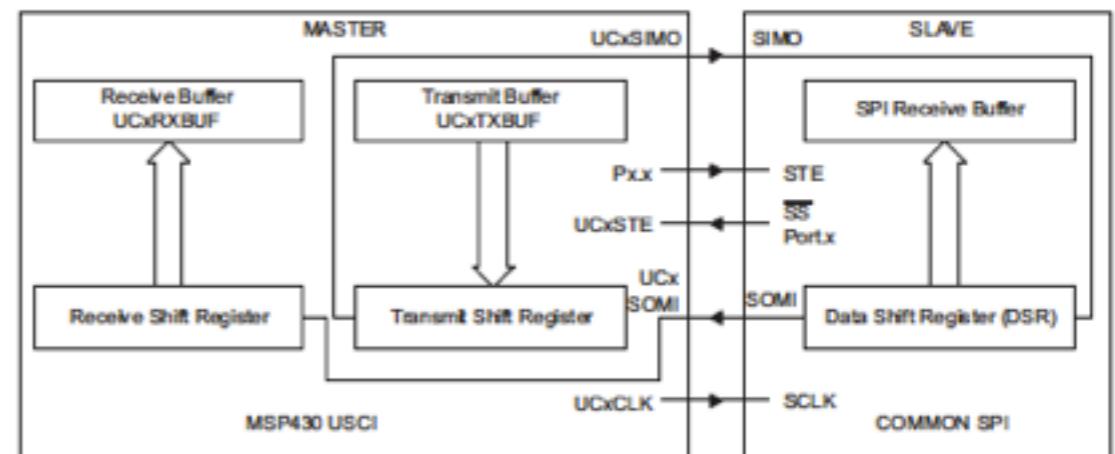
En el modo SPI el dato en serie es transmitido y recibido por múltiples dispositivos usando un reloj compartido proporcionado por el Maestro.

Operación del modo Maestro (Master) con un esclavo (Slave) externo.



El módulo USCI inicia la transmisión del dato cuando el dato se escribe en el registro **UCxTXBUF**, posteriormente el dato escrito se transfiere al registro **Transmit Shift Register**, cuando este último registro está vacío, se inicia la transferencia del dato en la pin **UCxSIMO** empezando ya sea con el más significativo o menos significativo bit dependiendo de la configuración. El Data en el pin **UCxSOMI** es recorrido en el registro **Receive Shift Register** en el siguiente flanco del reloj. Cuando el carácter es recibido el dato se mueve desde el registro **Receive Shift Register** a el registro del buffer de recepción **UCxRXBUF**.

Operación del modo Esclavo (Slave) con un Maestro (Master) externo.



El pin **UCxCLK** es una entrada para el SPI en modo de esclavo y tiene que ser proporcionado por el Maestro externo. La velocidad de transferencia es determinada por este reloj y no por el bit interno de generador de reloj. El dato se escribe en **UCxTXBUF** y se mueve a el registro **Transmit Shift Register** antes de que el **UCxCLK** sea transmitido a UCxSOMI. El dato en **UCxSIMO** es recorrido a el registro **Receive Shift Register** en el siguiente flanco de reloj y es movido al registro **UCxRXBUF** cuando el número de bits a transmitir se ha recibido. Cuando el dato es movido del registro **Receive Shift Register** a el registro del buffer de recepción **UCxRXBUF**, la bandera de interrupción **UCxRXIFG** se activa.

Operación del modo Maestro(Master) y Esclavo (Slave) con dos MSP430.

En el diagrama que se muestra a continuación se explica gráficamente la operación de comunicación serial SPI con dos MSP430 uno configurado como Maestro y el otro como Esclavo.

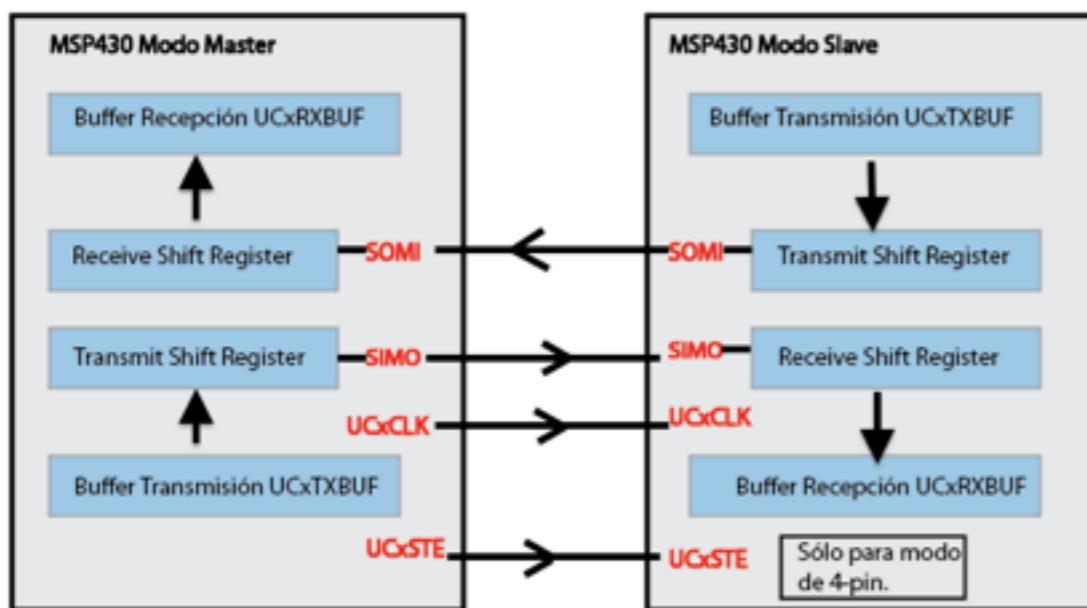


Figura 9.2 Conexión del MSP430 configurado como Maestro con el MSP430 configurado como Esclavo.

Bits UCA0SOMI, UCA0SIMO, UCA0CLK y UCA0STE del MSP430G2553 Launch Pad.

El microcontrolador MSP430G2553 en los bits 1, 2, 4 y 5 del puerto 1 (P1.1, P1.2, P1.4 y P1.5) tiene las cuatro señales para el intercambio de datos en el modo SPI del módulo USCI_A0 como se muestra a continuación.

P1.1 UCA0SOMI: USCI_A0 Modo SPI: Slave data out/ Master in (Salida del dato del Esclavo/Entrada al Maestro).

P1.2 UCA0SIMO: USCI_A0 Modo SPI: Slave data in/ Master out (Entrada de dato al Esclavo/Salida del Maestro).

P1.4 UCA0CLK: USCI_A0 Modo SPI: Clock input/output (Reloj entrada/salida).

P1.5 UCA0STE: USCI_A0 Modo SPI: Slave Transmit Enable (Habilita Transmisión al Esclavo). Sólo para modo 4-pin.

Para seleccionar la función USCI_A0 modo SPI en P1.1, P1.2, P1.4 y P1.5 se deben de configurar los registros P1DIR, P1SEL y P1SEL2 como se muestra en la tabla 9.2

Tabla 9.2. Configuración de los pines del MSP430G2553 para el módulo USCI en modo SPI.

Pin del MSP430G2553	Función	P1DIR	P1SEL	P1SEL2
P1.1	UCA0SOMI	Desde USCI	1	1
P1.2	UCA0SIMO	Desde USCI	1	1
P1.4	UCA0CLK	Desde USCI	1	1
P1.5	UCA0STE	Desde USCI	1	1

Inicialización y restauración del módulo USCI

La restauración (reset) del módulo USCI se lleva a cabo con el bit UCSWRST del registro UCA0CTL1.

Cuando UCSWRST es puesto en ‘1’ resetea los bits UCA0RXIE, UCA0TXIE, UCA0RXIFG, UCOE y UCFE y activa la bandera de la interrupción de transmisión UCA0RXIFG.

Cuando UCSWRST se limpia (se pone en ‘0’) desbloquea el módulo USCI para operar.

El proceso recomendado de re-configuración e inicialización del módulo USCI se enumera a continuación:

1. Poner en ‘1’ el bit UCSWRST del registro UCA0CTL1:

BIS.B #UCSWRST,&UCA0CTL1 (En código Assembly).

UCA0CTL1=UCSWRST; (En Código C).

2. Inicializar todos los registros.

3. Configurar todos los puertos a utilizar.

4. Limpiar (poner en ‘0’) el bit UCSWRST.

BIC.B #UCSWRST,&UCA0CTL1 (En código Assembly).

UCA0CTL1&=~UCSWRST (En Código C).

5. Habilitar las interrupciones (opcional) mediante los bits

Desarrollo de la práctica

1. Llevar a cabo la conexión de los microcontroladores MSP430G2553 como se muestra en el diagrama de la figura 9.3.

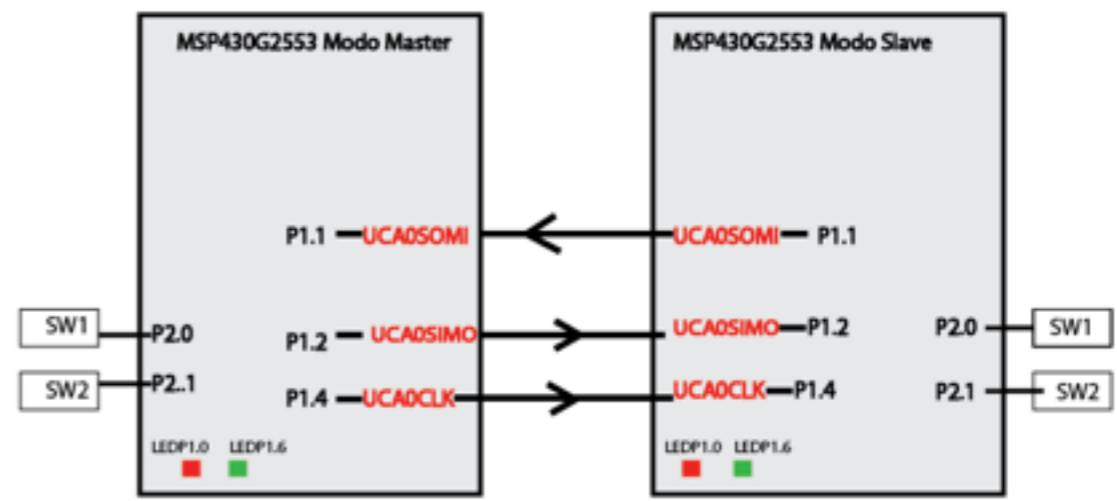


Figura 9.3. Conexión de los MSP430G2553 en modo Maestro y en modo Esclavo.

2. Explicación del programa.

El programa consiste en que ambos microcontroladores tengan dos Push-buttons (o switches) externos, y se configuren sus resistencias internas de pull-up. De modo que a través de la comunicación serial SPI al presionar el switch 1 (o push-button 1) del MSP430 Maestro se encienda el LED2 (Verde) conectado internamente a P1.6 del MSP430 Esclavo.

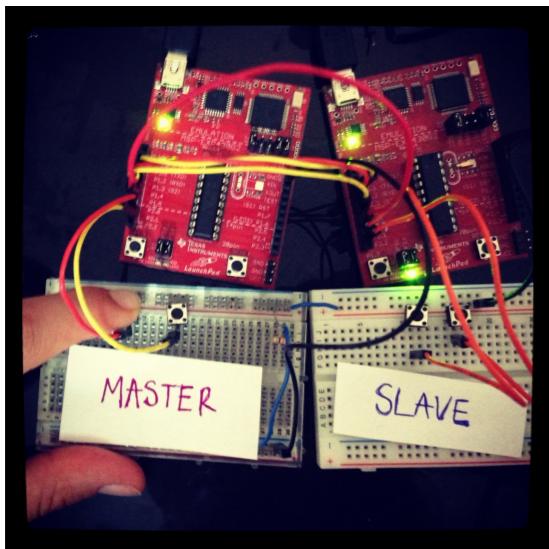


Figura 9.4. Switch 1 del MSP430 Maestro presionado, enciende LED2(Verde) conectado internamente a P1.6 del MSP430 Esclavo.

Y cuando se presione el switch 2 (o push-button 2) del MSP430 Maestro se encienda el LED1(rojo) conectado internamente a P1.0 del MSP430 Esclavo.

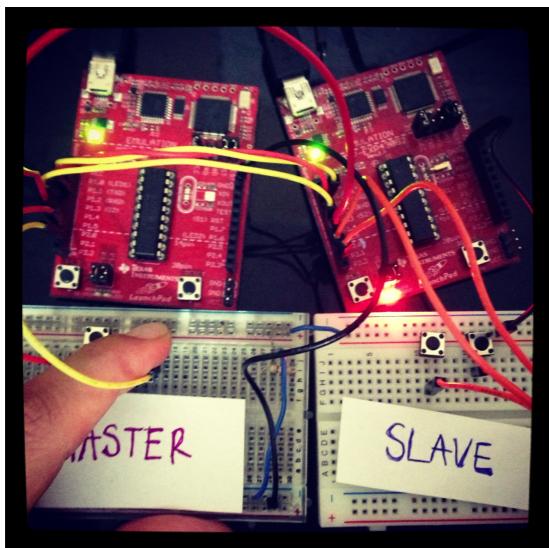


Figura 9.5. Switch 1 del MSP430 Maestro presionado, enciende LED1(rojo) conectado internamente a P1.6 del MSP430 Esclavo.

Del mismo modo cuando se presione el switch 1 (o push-button 1) del MSP430 Esclavo se encienda el LED2(verde) conectado internamente a P1.6 del MSP430 Maestro.

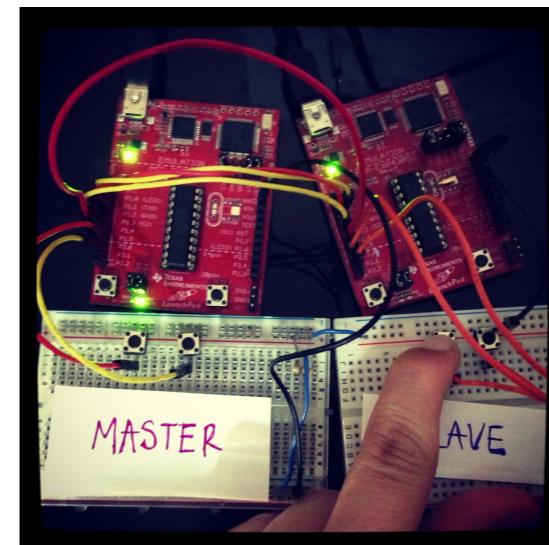


Figura 9.6. Switch 1 del MSP430 Esclavo presionado, enciende LED2(Verde) conectado internamente a P1.0 del MSP430 Maestro.

Finalmente cuando se presione el switch 2 (o push-button 2) del MSP430 Esclavo se encienda el LED1(rojo) conectado internamente a P1.0 del MSP430 Maestro.

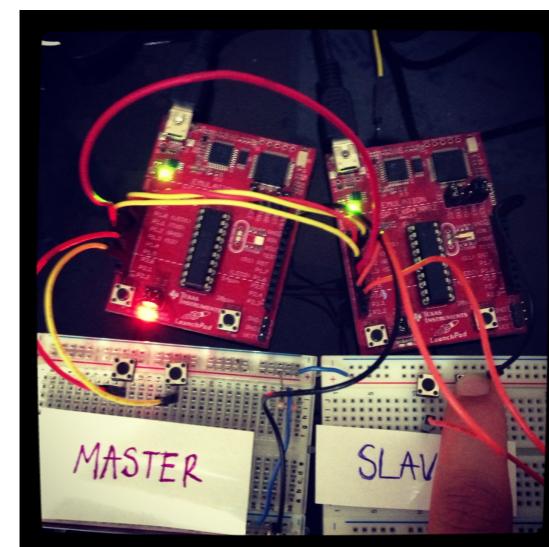


Figura 9.7. Switch 2 del MSP430 Esclavo presionado, enciende LED1(rojo) conectado internamente a P1.6 del MSP430 Maestro.

3. Hacer equipos de dos personas

Copiar y ejecutar el código de Modo Maestro (Master) SPI en un MSP430G2553 y copiar y ejecutar el código Modo Esclavo (Esclavo) SPI en el otro MSP430G2553.

Código Modo Maestro (Master) SPI

```
#include <msp430g2553.h>
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           //Detiene Perro Guardián.
    BCSCTL1= CALBC1_1MHZ;              //Calibración del SMCLK a 1MHz
    DCOCTL= CALDCO_1MHZ;
    UCA0CTL1=UCSWRST;                 // UCSWRST=1
    P1OUT = 0x00;                      // Limpia salidas
    P1DIR |= BIT0 + BIT5 + BIT6;        //Configura el BIT0,BIT5 y BIT6
                                        //de puerto 1 como salidas.
    P1SEL = BIT1 + BIT2 + BIT4;         //Selección de función de UCA0SOMI,
                                        //UCA0SIMO y UCA0CLK.
    P1SEL2 = BIT1 + BIT2 + BIT4;        //Selección de función de UCA0SOMI,
                                        //UCA0SIMO y UCA0CLK.
    P2DIR=0x00;                        //Puerto 2 como entrada.
    P2REN |= BIT0 + BIT1;               //Configuración de resistencias
    P2OUT |= BIT0 + BIT1;               //internas de pull-up en P2.0 y P2.1
    UCA0CTL0 |= UCCKPL + UCMST + UCMODE_0+ UCSYNC+UCMSB; // Modo 3-pin, SPI master
                                                        //Enviar primero MSB,
                                                        //Modo Síncrono.
    UCA0CTL1 |= UCSSSEL_2;             // Selección de reloj SMCLK
    UCA0BR0 |= 0x02;                   // Preescalado de reloj entre 2.
    UCA0BR1 = 0;                      // No hay modulación
    UCA0MCTL = 0;                     // Desbloqueo del Módulo USCI
    IE2 |= UCA0RXIE;                  // Habilitación de interrupción de Recepción.

    P1OUT &= ~BITS5;                  // Reseta Esclavo
    P1OUT |= BITS5;                   //Señales de SPI inicializadas.

    __delay_cycles(7500);             //Retardo de espera para que el esclavo se
                                    //Incialice.

    UCA0TXBUF = P2IN;                //Transmisión del dato generado por el switch 1
                                    // y el switch 2.

    __bis_SR_register(LPM0_bits + GIE); //CPU off, y habilitación general de
                                    //interrupciones.
}
```

```
//Rutina de Servicio de Interrupción (ISR)
#pragma vector=USCIAB0RX_VECTOR
interrupt void USCIA0RX_ISR(void)
{
    while (UCB0RXIFG ==0);          //Espera que todo el dato sea recibido.

    if (UCAB0RXBUF == 0x01)          //Compara el dato recibido con 0x01
        P1OUT |= BIT0;              //Si es correcto el switch 2 está presionado y
                                    //enciende el LED1 rojo conectado internamente a P1.0
    else if (UCA0RXBUF==0x02){      //Compara el dato recibido con 0x02
        P1OUT |= BIT6;              //Si es correcto el switch 1 esta presionado y
                                    //Enciende el LED2 verde conectado internamente a P1.6.
    }
    else{                           //Si el dato recibido no es ninguna de las opciones
        P1OUT=0x00;                //anteriores limpia la salida del puerto 1.
    }

    UCA0TXBUF = P2IN;              // Envía siguiente dato.

    __delay_cycles(7500);           //Añade un retardo de tiempo entre las transmisiones.
}
```

Código Modo Esclavo (Slave) SPI

```
#include <msp430g2553.h>
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           //Detiene Perro guardián.
    UCA0CTL1=UCSWRST;                 //UCSWRST=1
    P1OUT = 0x00;                      //Limpia salidas de puerto 1.
    P1DIR |= BIT0 + BIT6;               //BIT0 y BIT 1 de puerto 1 como salidas.
    P1SEL = BIT1 + BIT2 + BIT4;         //Selección de función de UCA0SOMI,
                                        //UCA0SIMO y UCA0CLK.
    P1SEL2 = BIT1 + BIT2 + BIT4;        //Selección de función de UCA0SOMI,
                                        //UCA0SIMO y UCA0CLK.
    P2DIR=0x00;                        //Puerto 2 como entrada.
    P2REN |= BIT0 + BIT1;               //Configuración de resistencias
    P2OUT |= BIT0 + BIT1;               //internas de pull-up en P2.0 y P2.1
    UCA0CTL0 |= UCCKPL + UCMODE_0+ UCSYNC+UCMSB; // Modo 3-pin, SPI Slave
                                                        //Enviar primero MSB,
                                                        //Modo Síncrono.
    UCA0MCTL = 0;                     // No hay modulación.
    UCA0CTL1 &= ~UCSWRST;             // Desbloqueo del Módulo USCI
    IE2 |= UCA0RXIE;                  // Habilitación de interrupción de Recepción.

    __delay_cycles(7500);             // Retardo de tiempo para que el Esclavo se
                                    //Incialice.

    UCA0TXBUF = P2IN;                //Transmisión del dato generado por el switch 1
                                    // y el switch 2.

    __bis_SR_register(LPM0_bits + GIE); // CPU off, y habilitación Global de Interrupciones.
}
```

```

//Rutina de Servicio de Interrupción (ISR)
#pragma vector=USCIAB0RX_VECTOR
_interrupt void USCIAB0RX_ISR(void)
{
    while (UCB0RXIFG ==0);      //Espera que todo el dato sea recibido.

    if (UCA0RXBUF == 0x01)      //Compara el dato recibido con 0x01
        P1OUT |= BIT0;          //Si es correcto el switch 2 está presionado y
                                //enciende el LED1 rojo conectado internamente a P1.0

    else if (UCA0RXBUF==0x02){  //Compara el dato recibido con 0x02
        P1OUT |= BIT6;          //Si es correcto el switch 1 esta presionado y
                                //Enciende el LED2 verde conectado internamente a P1.6.

    }else{
        P1OUT=0x00;             //Si el dato recibido no es ninguna de las opciones
                                //anteriores limpia la salida del puerto 1.

    }

    UCA0TXBUF =P2IN;           // Envía siguiente dato.

    __delay_cycles(7500);       //Añade un retardo de tiempo entre las transmisiones.
}

```

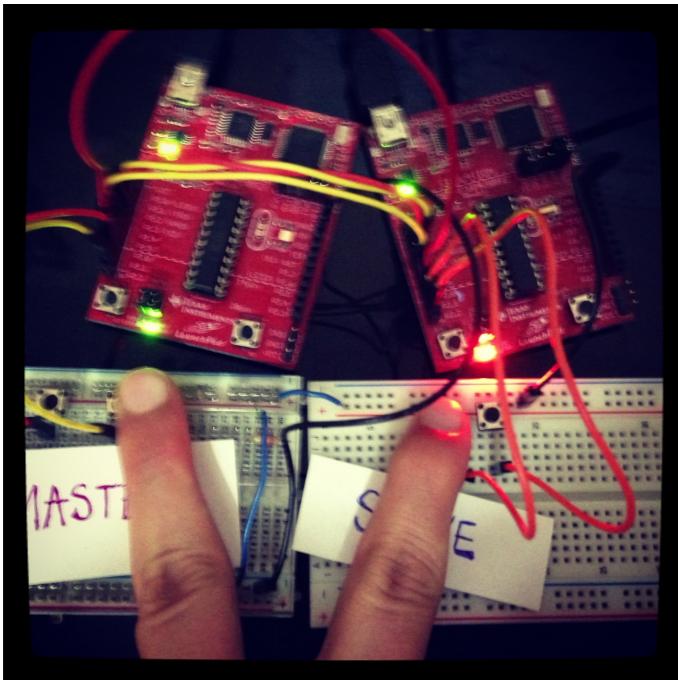


Figura 9.8. Switch 2 del MSP430 Maestro presionado y enciende LED1(Rojo) del MSP430 Esclavo. Switch 1 del MSP430 Esclavo presionado y enciende LED2(Verde) del MPS430 Maestro.

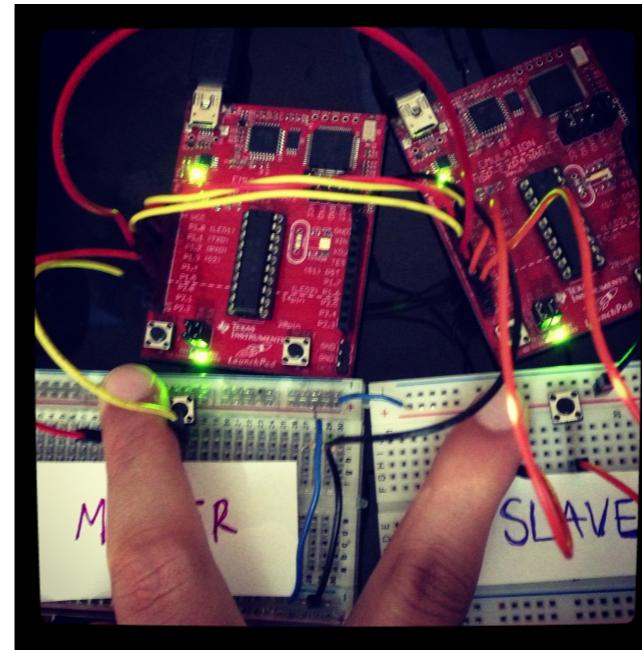


Figura 9.9. Switch 1 del MSP430 Maestro presionado y enciende LED2(Verde) del MSP430 Esclavo. Switch 1 del MSP430 Esclavo presionado y enciende LED2(Verde) del MSP430 Maestro.

4. Hacer reporte correspondiente a la práctica 9.

Práctica 10

Comunicación Serial Asíncrona: USCI en modo UART

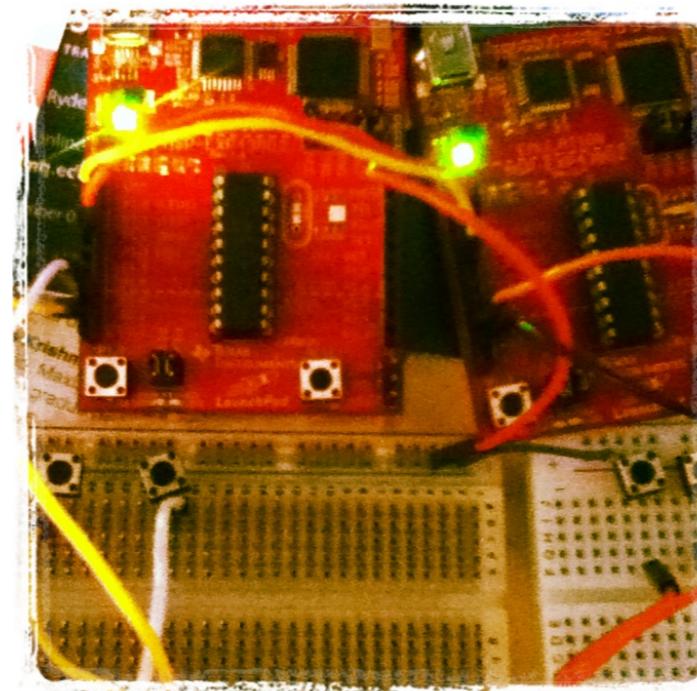
OBJETIVOS

1. Conocer el protocolo de comunicación **UART** (Universal Asynchronous Receiver-Transmitter).
2. Aprender el funcionamiento y configuración del módulo **USCI** (Universal Serial Communication Interface) en modo **UART**.
3. Establecer comunicación serial síncrona mediante el protocolo **UART** entre dos micro-controladores **MSP430G2553**.

MATERIALES

- 2 Microcontrolador **MSP430G2553**.
- 4 Push-button.
- Cables

En la práctica anterior se estudió la comunicación serial síncrona configurando el módulo USCI en modo SPI. Continuando con la comunicación serial, en esta práctica se aborda la comunicación serial asíncrona configurando el módulo USCI en modo **UART** (Universal Asynchronous Receiver-Transmitter). Como se mencionó en la práctica anterior el módulo USCI (Universal Serial Communication Interface) permite múltiples modos de comunicación serial.



El programa a desarrollar se basa en el efectuado en la práctica anterior, con la diferencia de que la comunicación es generada de manera asíncrona con el módulo USCI en modo **UART**. De igual manera se utilizan dos MSP430G2553 para llevar a cabo la transferencia de datos y dos push-buttons (o switches) en cada tarjeta para enviar el dato a transmitir y una vez que se recibe el dato desplegar en los LEDs internos conectados a P1.0 y P1.6 el switch o push-button que fue presionado.

Metodología

Comunicación serial Asíncrona UART

UART por las siglas en inglés de Universal Asynchronous Receiver-Transmitter (Transmisor-Receptor Asíncrono Universal) es uno de los protocolos seriales remotor más utilizados, la mayoría de los microcontroladores disponen de un hardware UART.

La transmisión asíncrona permite que el dato sea transmitido sin que el transmisor tenga que enviar una señal de reloj al receptor. En lugar de ésto, el transmisor y receptor se configuran con parámetros de tiempo en bits especiales añadidos a cada palabra (8bits) de dato.

La comunicación UART usa una línea de datos simple para transmitir y otra para recibir datos. El dato o palabra se transmite de la siguiente forma:

Un bit llamado bit de inicio (start bit) en nivel bajo es añadido al inicio del dato a transmitir. El bit de inicio (start bit) se utiliza para informarle al receptor que un dato o una palabra (8bits) de dato se transmitirá, forzando a el reloj del receptor entrar en sincronización con el reloj del transmisor. Estos dos relojes deben de tener su velocidad (o frecuencia) lo más similar posible para evitar problemas en la comunicación.

Después de haber enviado el bit de inicio, cada bit del dato se envía individualmente. Cada bit es transmitido en la misma cantidad de tiempo. El receptor analiza si el bit transmitido es '1' ó '0' en la mitad de tiempo en el que se transmitió dicho bit. Por ejemplo si un bit se envía en 2 segundos el receptor tarda 1 segundo en determinar si el bit recibido es '1' ó '0'.

Cuando el dato (o palabra) se ha transmitido completamente, el transmisor añade un bit de paridad (bit parity) si es que éste se configura. El bit de paridad se utiliza para verificar si algún error ocurrió, y éste puede ser par o impar.

Posteriormente el receptor busca el bit de parada (stop bit) en nivel alto. Si el bit de parada no aparece cuando lo debe de hacer, UART desprecia todo el dato o palabra recibida y reporta un Error de Trama (Framing Error) a el procesador. La causa más común que genera el error de trama es que los relojes del transmisor y del receptor no tienen la misma velocidad. Otro error que puede causar el error de trama es que la señal haya sido interrumpida.

Si otro dato esta listo para transmitirse, el bit de inicio del nuevo dato debe de enviarse tan pronto como el bit de parada del dato anterior haya sido enviado. El bit de inicio se envía en nivel bajo y el bit de parada en nivel alto indicando que siempre hay un transición de alto a bajo para iniciar la transmisión.

Configuración del módulo USCI en modo UART

Se recomienda ampliamente leer el apartado de USCI en modo UART en la Guía de Usuario proporcionada por el fabricante.

En el modo asíncrono, el módulo USCI_A0 conecta al MSP430G2553 con un sistema externa mediante dos pines externos, UCA0RXD y UCA0TXD. El modo UART se selecciona limpiando el bit de sincronización UCSYNC del registro UAC0CTL0.

En el modo UART, el módulo USCI transmite y recibe caracteres en una tasa de bit asíncrona con otro dispositivo. El tiempo de transmisión de cada carácter está basado en el baudaje (baud rate) seleccionado del USCI. Las funciones de transmisión y recepción usan la misma frecuencia de baudaje (baud rate).

Inicialización y restauración del módulo USCI

La restauración (reset) del módulo USCI se lleva a cabo con el bit UCSWRST del registro UCA0CTL1.

Cuando UCSWRST es puesto en ‘1’ resetea los bits UCA0RXIE, UCA0TXIE, UCA0RXIFG, UCOE y UCFE y activa la bandera de la interrupción de transmisión UCA0RXIFG.

Cuando UCSWRST se limpia (se pone en ‘0’) desbloquea el módulo USCI para operar.

El proceso recomendado de re-configuración e inicialización del módulo USCI se enumera a continuación:

1. Poner en ‘1’ el bit UCSWRST del registro UCA0CTL1:
BIS.B #UCSWRST,&UCA0CTL1 (En código Assembly).
UCA0CTL1=UCSWRST; (En Código C).
2. Inicializar todos los registros.
3. Configurar todos los puertos a utilizar.
4. Limpiar (poner en ‘0’) el bit UCSWRST.
BIC.B #UCSWRST,&UCA0CTL1 (En código Assembly).
UCA0CTL1&=~UCSWRST (En Código C).
5. Habilitar las interrupciones (opcional) mediante los bits UCA0TXIE y UCA0RXIE del registro IE2.

Formato del carácter

El formato del carácter en UART consiste en un bit de inicio (start bit), siete u ocho bits de dato, un bit de paridad par/impar/no-bit de paridad, un bit de dirección (address-bit mode) y uno o dos bits de parada (stop bit). El bit UCMSB controla la dirección de transferencia seleccionando que se transfiera primero el MSB o el LSB. Transferir primero el LSB es la configuración que típicamente requiere la comunicación. UART. El esquemático del formato se muestra a continuación.

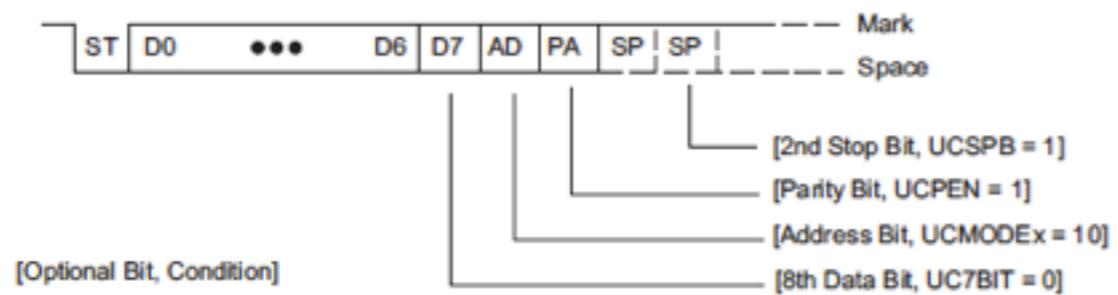


Figura 10.1. Formato del carácter.

Generación USCI de la tasa de Baudaje (Baud Rate).

El generador USCI de la tasa de baudaje es capaz de producir tasas de baudaje estándar a partir de fuentes de frecuencias no-estándar. Este provee dos modos de operación seleccionados por el bit UCOS16 del registro UAC0MCTL.

Generación de la tasa de Baudaje (Baud Rate) con reloj de baja frecuencia.

El modo de baja frecuencia se selecciona cuando UCOS16=0. Este modo permite generación de tasas de baudaje desde fuentes de reloj de baja frecuencia (por ejemplo, 9,600 baud desde un cristal de 32768Hz). Utilizando una entrada de baja frecuencia el consumo de potencia del módulo se reduce. En este modo no se recomienda utilizar osciladores de alta frecuencia (arriba de 8MHz) ya que para

altas frecuencias existe el otro modo (cuando UCOS16=1). Están involucrados los registros UCBRx (compuesto por dos registros UCA0BR0 y UCA0BR1), y los bits de UCBRSx del registro UCA0MCTL. La tabla para configurar la tasa de baudaje (baud rate) es la **tabla 15-4** (páginas 435-436) de la Guía de Usuario proporcionada por el fabricante.

Generación de la tasa de Baudaje (Baud Rate) con reloj de alta frecuencia (Oversampling Baud Rate).

El modo de alta frecuencia se selecciona cuando UCOS16=0. Este modo permite generación de tasas de baudaje con relojes de alta frecuencia con bajo porcentaje de error en los baudios (bauds). Están involucrados los registros UCBRx (compuesto por dos registros UCA0BR0 y UCA0BR1), y los bits de UCBRSx y UCBRFx del registro UCA0MCTL. La tabla para configurar la tasa de baudaje (baud rate) es la **tabla 15-5** (página 436) de la Guía de Usuario proporcionada por el fabricante.

Registros del módulo USCI en modo UART

1. UCA0CTL0: USCI_A0 Control Register 0 (Registro de Control 0).
2. UCA0CTL1: USCI_A0 Control Register 1 (Registro de Control 1).

3. UCA0BR0: USCI_A0 Baud Rate Control Register 0 (Registro de Control de velocidad de transmisión 0).
4. UCA0BR1: USCI_A0 Baud Rate Control Register 1 (Registro de Control de velocidad de transmisión 1).
5. UCA0STAT: USCI_A0 Status Register (Registro de Estado).
6. UCA0MCTL: USCI_A0 Modulation Control Register (Registro de Control de modulación).
7. UCA0RXBUF: USCI_A0 Receive Buffer Register. (Registro del Buffer de Receptor).
8. UCA0TXBUF: USCI_A0 Transmit Buffer Register (Registro del Buffer de Transmisión).
9. UCA0ABCTL: USCI_A0 Auto Baud Control Register (Registro de Control de Auto Baud).
- 10.UCA0IRTCTL: USCI_A0 IrDA Transmit Control Register (Registro de Control de Transmisión IrDA).
- 11.UCA0IRRCTL: USCI_A0 IrDA Receive Control Register (Registro de Control de Recepción IrDA).IE2.
- 12.IE2: Interrupt Enable Register 2. (Registro de Habilitación de Interrupciones 2).

13. IFG2. Interrupt Flag Register 2. (Registro de Bandera de Interrupción 2).

A continuación se explican los bits de los registros para el modo UART que se utilizan en esta práctica: UCA0CTL0, UCA0CTL1, UCA0BR0, UCA0BR1, UCA0MCTL, UCA0RXBUF, UCA0TXBUF e IE2. El resto de los registros se dejan al lector para que los estudie por su propia cuenta (se encuentran explicados en la Guía de Usuario proporcionada por el fabricante).

1. **UCA0CTL0: USCI_A0 Control Register 0 (Registro de Control 0).**

7	6	5	4	3	2	1	0
UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx	UCSYNC	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCPEN BIT 7 Habilitación de paridad.

0	Bit de paridad deshabilitado.
1	Bit de paridad habilitado. El bit de paridad es generado por UCA0TXD y esperado por UCA0RXD

UCPAR BIT6 Selección de Paridad. Cuando UCPE=1

0	Paridad Impar.
1	Paridad Par

UCMSB BIT5 Selecciona que primero se envíe el bit más significativo (MSB) o el bit menos significativo (LSB).

0	Primero LSB
1	Primero MSB

UC7BIT BIT4 Longitud de carácter.

0	Dato de 8-bits
1	Dato de 7-bits

UCSPB BIT3 Número de bits de parada (stop bits).

0	Un bit de parada.
1	Dos bits de parada(Stop bits).

UCMODEX BITS2-1 Modo USCI.

00	Modo UART.
01	Modo multiprocessor Idle-line.
10	Modo multiprocessor Adress-bit.
11	Modo UART con detección automática de tasa de baudaje (baud rate).

UCSYNC BIT0 Habilitación de Modo síncro o asíncrono.

0	Modo Asíncrono.
1	Modo Síncrono.

2. UCA0CTL1: USCI_A0 Control Register 0 (Registro de Control 0).

7	6	5	4	3	2	1	0
UCSSELx	UCRXIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST	rw-1

UCSSEL_x BITS7-6 Selección de Fuente de reloj (BRCLK)

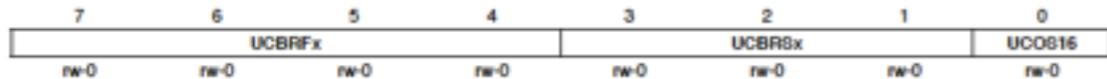
00	UCLK
01	ACLK
10	SMCLK
11	SMCLK

UCRXIE BIT5 Habilita Interrupción de recepción de caracteres erróneos.

0	Caracteres erróneos son rechazados y no se activa la bandera UCA0RXIFG.
1	Caracteres erróneos son recibidos y se activa la bandera UCA0RXIFG.

UCBRKIE BIT4	Habilita Interrupción de recepción de carácter truncado.	UCSWRST BIT0	Habilita el Reset del software.																				
	<table border="1"> <tr> <td>0</td><td>Caracteres recibidos truncados no activan la bandera UCA0RXIFG.</td></tr> <tr> <td>1</td><td>Caracteres recibidos truncados no activan la bandera UCA0RXIFG.</td></tr> </table>	0	Caracteres recibidos truncados no activan la bandera UCA0RXIFG.	1	Caracteres recibidos truncados no activan la bandera UCA0RXIFG.		<table border="1"> <tr> <td>0</td><td>Deshabilitado. Desbloquea el módulo USCI para que pueda operar.</td></tr> <tr> <td>1</td><td>Habilitado. Lógica USCI se mantiene en el estado de reset.</td></tr> </table>	0	Deshabilitado. Desbloquea el módulo USCI para que pueda operar.	1	Habilitado. Lógica USCI se mantiene en el estado de reset.												
0	Caracteres recibidos truncados no activan la bandera UCA0RXIFG.																						
1	Caracteres recibidos truncados no activan la bandera UCA0RXIFG.																						
0	Deshabilitado. Desbloquea el módulo USCI para que pueda operar.																						
1	Habilitado. Lógica USCI se mantiene en el estado de reset.																						
UCDORM BIT3	Dormant (Inactivo) Pone al módulo USCI en modo dormido(sleep).	3. UCA0BR0: USCI_A0 Baud Rate Control Register 0 (Registro de Control de velocidad de transmisión 0).	BYTE BAJO.																				
	<table border="1"> <tr> <td>0</td><td>No Inactivo(No-Dormant). Todos los caracteres recibidos activan bandera UCA0RXIFG.</td></tr> <tr> <td>1</td><td>Inactivo(Dormant). Sólo caracteres que provienen de idle-line o address-bit activan bandera UCA0RXIFG.</td></tr> </table>	0	No Inactivo(No-Dormant). Todos los caracteres recibidos activan bandera UCA0RXIFG.	1	Inactivo(Dormant). Sólo caracteres que provienen de idle-line o address-bit activan bandera UCA0RXIFG.		<p>UCBRx - low byte</p> <table border="1"> <tr> <td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>RW</td><td>RW</td><td>RW</td><td>RW</td><td>RW</td><td>RW</td><td>RW</td><td>RW</td> </tr> </table>	7	6	5	4	3	2	1	0	RW							
0	No Inactivo(No-Dormant). Todos los caracteres recibidos activan bandera UCA0RXIFG.																						
1	Inactivo(Dormant). Sólo caracteres que provienen de idle-line o address-bit activan bandera UCA0RXIFG.																						
7	6	5	4	3	2	1	0																
RW	RW	RW	RW	RW	RW	RW	RW																
UCTXADDR BIT2	Dirección de Transmisión.	4. UCA0BR1: USCI_A0 Baud Rate Control Register 1 (Registro de Control de velocidad de transmisión 1).	BYTE ALTO.																				
	<table border="1"> <tr> <td>0</td><td>El siguiente frame transmitido es un dato.</td></tr> <tr> <td>1</td><td>El siguiente frame transmitido es una dirección.</td></tr> </table>	0	El siguiente frame transmitido es un dato.	1	El siguiente frame transmitido es una dirección.		<p>UCBRx - high byte</p> <table border="1"> <tr> <td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>RW</td><td>RW</td><td>RW</td><td>RW</td><td>RW</td><td>RW</td><td>RW</td><td>RW</td> </tr> </table>	7	6	5	4	3	2	1	0	RW							
0	El siguiente frame transmitido es un dato.																						
1	El siguiente frame transmitido es una dirección.																						
7	6	5	4	3	2	1	0																
RW	RW	RW	RW	RW	RW	RW	RW																
UCTXBRK BIT1	Descanso de Transmisión (Transmit break).		Configuración del preescalado del reloj. El valor de 16 bits de UCA0BR0+UCA0BR1*256 forma el valor del preescalado.																				
	<table border="1"> <tr> <td>0</td><td>El siguiente frame transmitido no es un descanso (break).</td></tr> <tr> <td>1</td><td>El siguiente frame transmitido es un descanso (break).</td></tr> </table>	0	El siguiente frame transmitido no es un descanso (break).	1	El siguiente frame transmitido es un descanso (break).																		
0	El siguiente frame transmitido no es un descanso (break).																						
1	El siguiente frame transmitido es un descanso (break).																						

5. UCA0MCTL: USCI_A0 Modulation Control Register
(Registro de Control de modulación).



UCBRFX BITS 7-4 Selección del primer tipo de modulación.

Estos bits determinan el patrón de modulación para BTCLK 16 cuando UCOS16=1. En la Tabla 15-3 de la Guía de Usuario se proporciona los patrones de modulación cuando UCOS16=1.

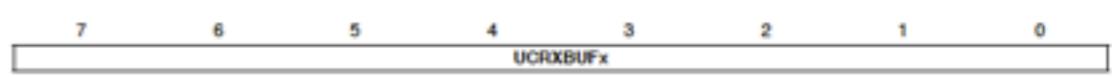
UCBRSx BITS 3-1 Selección del segundo tipo de modulación.

Estos bits determinan el patrón de modulación para BTCLK cuando UCOS16=0. En la Tabla 15-2 de la Guía de Usuario se proporciona los patrones de modulación cuando UCOS16=0.

UCOS16 BIT0 Habilitación de Modo de Oversampling (Generación de baud rate con reloj de alta frecuencia)

0	Deshabilitado.
1	Habilitado.

6. UCA0RXBUF: USCI_A0 Receive Buffer Register.
(Registro del Buffer de Receptor).



UCRXBUFX BITS 7-0:

El buffer del receptor contiene el último carácter recibido del registro Receive Shift Register. Leyendo UCA0RXBUF limpia los bits de Error de Recepción y la bandera de interrupción UCA0RXIFG.

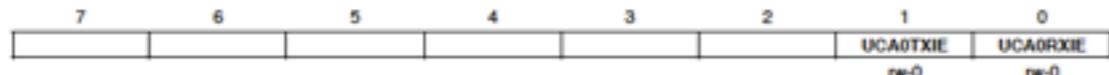
7. UCA0RXBUF: USCI_A0 Transmit Buffer Register.
(Registro del Buffer de Transmisión).



UCTXBUFX BITS 7-0:

El dato en el buffer de transmisión sostiene al dato esperando a ser movido en el registro transmit shift y ser transmitido. Escribiendo en el buffer de transmisión de datos UCA0TXIFG se limpia la bandera de interrupción UCA0TXIFG.

8. IE2: Interrupt Enable Register 2. (Registro de Habilitación de Interrupciones 2).



BITS 7-2 Estos bits son usados en otros módulos.

UCA0TXIE BIT1 Habilitación de Interrupción de Transmisión USCI_A0.

0	Interrupción deshabilitada.
1	Interrupción habilitada.

UCA0RXIE BIT0 Habilitación de Interrupción de Recepción USCI_A0.

0	Interrupción deshabilitada.
1	Interrupción habilitada.

Bits UCA0TXD, UCA0RXD del MSP430G2553 Launch Pad.

El microcontrolador MSP430G2553 en los bits 1 y 2 del puerto 1 (P1.1 y P1.2) tiene las dos líneas de transmisión y recepción necesarias para el intercambio de datos en el modo UART del módulo USCI_A0 como se muestra a continuación.

P1.1 UCA0RXD: USCI_A0 Modo UART: Receive Data Input (Entrada de Dato al Receptor).

P1.2 UCA0TXD: USCI_A0 Modo UART: Transmit Data Output (Salida de Dato del Transmisor).

Para seleccionar la función USCI_A0 modo UART en P1.1 y P1.2 se deben de configurar los registros PxDIR, PxSEL y PxSEL2 como se muestra en la tabla 10.1.

Tabla 10.1 Configuración de los pines del MSP430G2553 para el módulo USCI en modo UART.

Pin del MSP430G2553	Función	P1 DIR	P1 SEL	P1 SEL2
P1.1	UCA0SOMI	Desde USCI	1	1
P1.2	UCA0SIMO	Desde USCI	1	1

Desarrollo de la práctica

1. Hacer equipo de dos personas y llevar a cabo la conexión de los microcontroladores MSP430G2553 como se muestra en el diagrama de la figura 10.2.



Figura 10.2. Conexión entre los microcontroladores MSP430G2553 en el modo de comunicación UART.

2. Explicación del programa.

El programa consiste en que ambos microcontroladores tengan dos Push-buttons (o switches) externos, y se configuren sus resistencias internas de pull-up. De modo que a través de la comunicación serial asíncrona UART al presionar el switch 1 del MSP430 UNO se encienda el LED1 (Rojo) conectado internamente a P1.0 del MSP430 DOS.

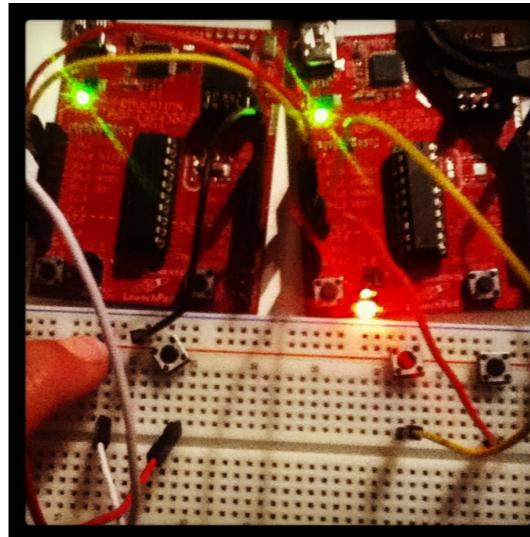
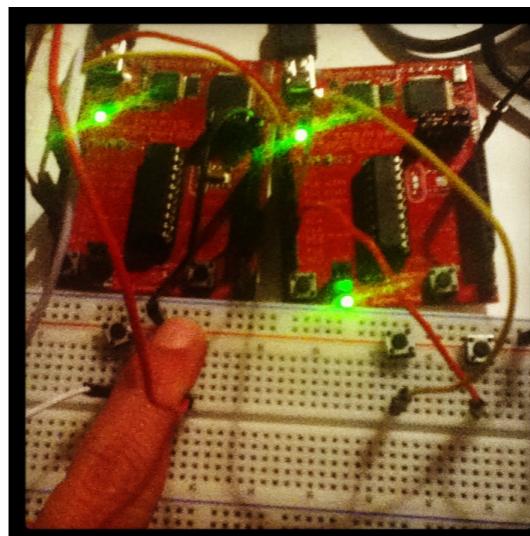


Figura 10.3. Switch 1 del MSP430 UNO presionado, enciende LED1(Rojo) conectado internamente a P1.0 del MSP430 DOS.



Y cuando se presione el switch 2 (o push-button 2) del MSP430 UNO se encienda el LED2(Verde) conectado internamente a P1.6 del MSP430 DOS.

Figura 10.4. Switch 2 del MSP430 UNO presionado, enciende LED2(Verde) conectado internamente a P1.6 del MSP430 DOS.

Del mismo modo cuando se presione el switch 1 del MSP430 DOS se encienda el LED1(Rojo) conectado internamente a P1.60 del MSP430 UNO.

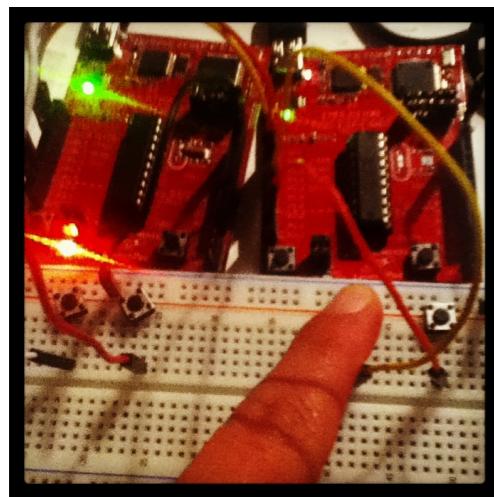


Figura 10.5. Switch 1 del MSP430 DOS presionado, enciende LED1(Rojo) conectado internamente a P1.0 del MSP430 UNO.

Y cuando se presione el switch 2 (o push-button 2) del MSP430 DOS se encienda el LED2(Verde) conectado internamente a P1.6 del MSP430 UNO.

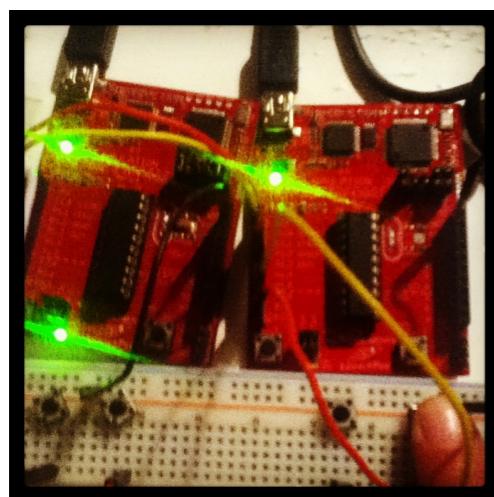


Figura 10.6. Switch 2 del MSP430 DOS presionado, enciende LED2(Verde) conectado internamente a P1.6 del MSP430 UNO.

Si se presionan el push-button 1 del MSP430G2553 **UNO** y el push-button 1 del MSP430G2553 **DOS** al mismo tiempo, se debe de encender el LED1(Rojo) conectado internamente a P1.0 de ambos microcontroladores.

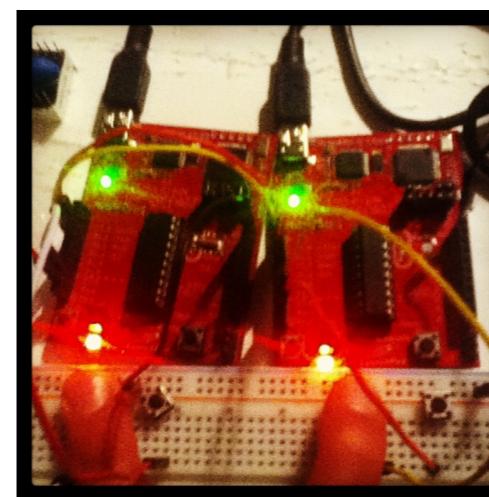


Figura 10.7. Switch 1 del MSP430 UNO y DOS presionados, encienden LEDs1(Rojos) conectados internamente a P1.0 del MSP430 UNO y DOS.

Si se presionan el push-button 2 del MSP430G2553 **UNO** y el push-button 2 del MSP430G2553 **DOS** al mismo tiempo, se debe de encender el LED2(Verde) conectado internamente a P1.0 de ambos microcontroladores.

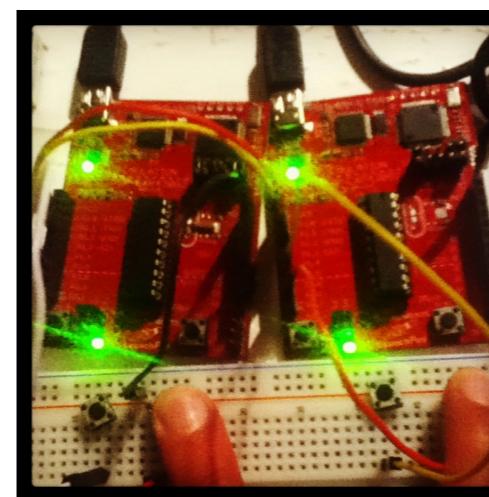


Figura 10.8. Switch 2 del MSP430 UNO y DOS presionados, encienden LEDs2(Verdes) conectados internamente a P1.6 del MSP430 UNO y DOS.

3. Copiar y ejecutar el código en C que se presenta a continuación en ambos microcontroladores y corroborar su funcionamiento.

```
#include <msp430G2553.h>
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           // Detiene Perro Guardián.
    BCSCTL1= CALBC1_1MHZ;             // Calibración del SMCLK a 1MHz
    DCOCTL= CALDCO_1MHZ;
    UCA0CTL1 |=UCSWRST;               // UCSWRST=1 para Inicialización
                                       // del módulo USCI.
    P1OUT = 0x00;                     // Limpia salidas del puerto 1
                                       // (P1.0 y P1.6).
    P1DIR |= BIT0 + BIT6;              // BIT0 y BIT 1 de puerto 1 como salidas.
    P1SEL = BIT1 + BIT2;               // Selección de función USCI_A0 en modo
                                       // UART: UCA0TXD y UCA0RXD.
    P1SEL2 = BIT1 + BIT2;              // Selección de función USCI_A0 en modo
                                       // UART: UCA0TXD y UCA0RXD.
    P2DIR=0x00;                      // Puerto 2 como entrada.
    P2REN |= BIT0 + BIT1;              // Configuración de resistencias
                                       // internas de pull-up en P2.0 y P2.1.
    P2OUT |= BIT0 + BIT1;              // BCLK = SMCLK (1MHz).
    UCA0CTL1 |= UCSEL_2;              // 1MHz/128000 = 7.81 (redondeado 7).
    UCA0BR0 = 0x07;                  // Baud Rate=128000bauds.

    UCA0BR1 = 0x00;
    UCA0MCTL = UCBRS2+ UCBRS1 + UCBRS0; // Modulación tipo UCBRSx = 7 (111b).
                                         // Baud Rate=128000 bauds.
    UCA0CTL1 &= ~UCSWRST;            // Habilita operación del módulo USCI.
    IE2 |= UCA0RXIE + UCA0TXIE;      // Habilita Interrupción de transmisión y
                                       // recepción.
    __bis_SR_register(LPM3_bits + GIE); // Modo de consumo LPM3 y habilitación
                                       // global de interrupciones.
}

// Rutina de Servicio de Interrupción (ISR) del Transmisor.
#pragma vector=USCIAB0TX_VECTOR
_interrupt void USCI0TX_ISR(void)
{
    unsigned char TxByte=0;
    if (P2IN == 0x02)                // Si el switch 1 está presionado el dato a
        TxByte |= BIT0;              // transmitir es 0x01 para que encienda el LED1
                                       // conectado internamente a P1.0 del receptor.
    if (P2IN== 0x01)                 // Si el switch 2 está presionado el dato a
        TxByte |= BIT6;              // transmitir es 0x40 para que encienda el LED2
                                       // conectado internamente a P1.6 del receptor.

    UCA0TXBUF = TxByte;              // Lee UCA0TXBUF para limpiar la bandera de
                                       // interrupción y escribe el siguiente dato a
                                       // enviar.
}
```

```
// Rutina de Servicio de Interrupción (ISR) del Receptor.
#pragma vector=USCIAB0RX_VECTOR
_interrupt void USCI0RX_ISR(void)
{
    P1OUT = UCA0RXBUF;              // P1OUT toma el valor recibido a través del
                                       // registro UCA0RXBUF, de este modo se conoce
                                       // cual switch fue oprimido según el LED (LED1 o
                                       // LED2) que encienda.
}
```

4. Hacer reporte correspondiente a la práctica 10.

Bibliografía y Mesografía

BIBLIOGRAFÍA

- [1]. Texas Instruments. User's Guide. MSP430x2xx Family. (Última revisión Diciembre 2012).
- [2]. Texas Instruments. MSP430G2x53/MSP430G2x13 Data-sheet (Última Revisión Febrero 2013).
- [3]. John H. Davies. MSP430 Microcontroller Basics. Editorial Newnes. (2008).
- [4]. Robert L. Boylestad y Louis Nashelsky. "Electrónica: Teoría de Circuitos y Dispositivos Electrónicos". Editorial PEARSON. Edición 10ma.

MESOGRAFÍA

- [1]. MSP430. <http://todomcu.scienceontheweb.net> [Consulta:Diciembre 2012-Marzo 2013].
- [2]. Henry Laredo. Blog MicroEmbebidos (PIC,MSP430,LPC,RTOS).
<http://microembebidos.wordpress.com> [Consulta: Diciembre 2012-Marzo 2013]
- [3]. Adafruit Learning Systems. Character LCDs.
<http://learn.adafruit.com/character-lcds/wiring-a-character-lcd> [Consulta: Marzo 2013]