Ejemplo de programación de PIC16F84: Cruce de semáforos

Fernando Peral Pérez Noviembre 2005

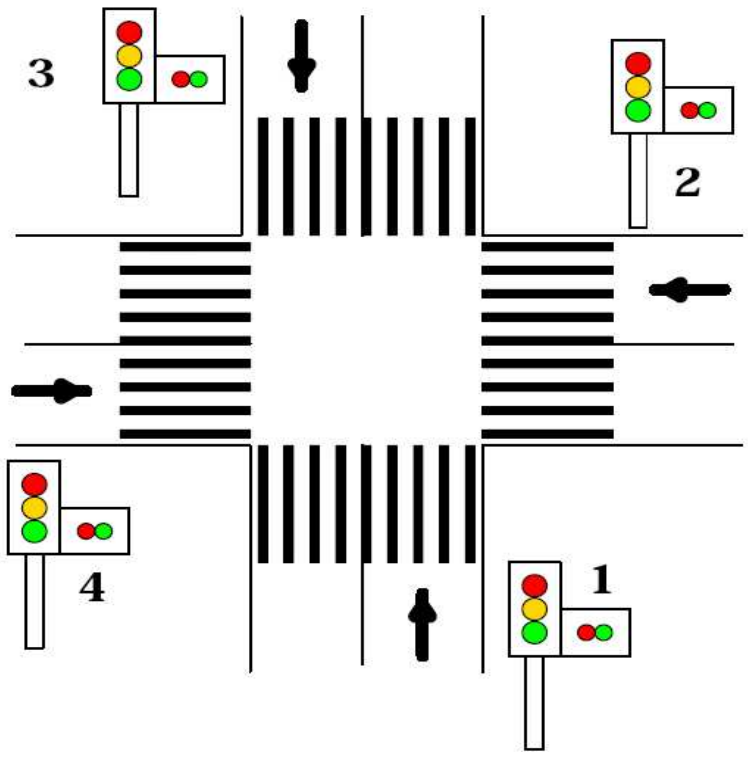
0. Planteamiento del problema.

Los semáforos constan de luces de aviso a peatones (rojo y verde) y de aviso a vehículos (rojo, ámbar y verde). Los semáforos 1 y 3 se comportan de idéntica forma. Lo mismo ocurre con el 2 y el 4. Cuando los semáforos 1 y 3 permitan el tráfico de vehículos por sus correspondientes vías, los semáforos 2 y 4 deben prohibirlo y a la inversa.

La temporización para cada semáforo será:

- La luz verde está activa durante 25 segundos.
- La luz ámbar parpadea durante 5 segundos (cada ½ segundo cambia su estado).
- La luz roja se mantiene encendida durante 30 segundos.
- Cuando la luz verde, o la luz ámbar se encuentren encendidas, la luz roja de los peatones debe estar activada.
- Cuando la luz roja se encuentre encendida, la luz verde de paso a los peatones debe estar activada, salvo durante los 10 últimos segundos en que debe parpadear con un periodo de un segundo.

Se pide el esquema hardware utilizando el microcontrolador PIC16F84A, así como el programa de control escrito en ensamblador. Suponed que los semáforos se representan con LED de colores (hacer un cálculo de las intensidades necesarias).



1. Asignación de pines

Los semáforos S1 y S3 son iguales y lo mismo ocurre con S2 y S4, por otra parte, el comportamiento de los semáforos de peatones es el siguiente:

- El verde está encendido siempre a la vez que el rojo de los coches, excepto en los últimos 10 segundos, que parpadea, por lo tanto es otra señal
- El rojo está encendido cuando está apagado el verde, es una señal distinta de las otras.

En resumen, vamos a tener que generar las siguientes señales:

- Semáforo 1: Rojo (R1), Verde (V1) y Ámbar (A1) para los coches, Rojo para los peatones (RP1), verde para los peatones (VP1)
- Semáforo 2: Rojo (R2), Verde (V2) y Ámbar (A2) para los coches, Rojo para los peatones (RP2), verde para los peatones (VP2)

En total, 10 señales, que se pueden generar con un puerto de 8 bits, el puerto B del microcontrolador, y un par de bits adicionales del puerto A. Utilizaremos las señales activas a nivel alto, ya que al atacar a leds, necesitaremos generar corrientes elevadas, y los pines del micro son capaces de absorber más corriente de la que pueden entregar.

La asignación de pines en el puerto B va a quedar distribuida de la siguiente forma:

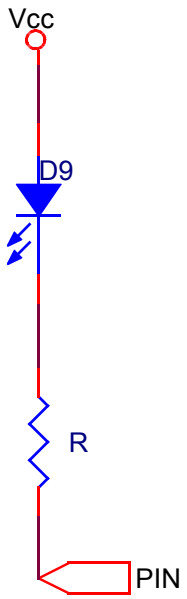
Pin	7	6	5	4	3	2	1	0
Semáforo	S2	S1	S2				S1	
Señal	RP2	RP1	R2	A2	V2	R1	A1	V1

En el puerto A se utilizará RA0 para VP1 y RA1 para VP2.

2. Intensidad por los leds

La conexión de los leds al circuito, al ser las salidas activas a nivel bajo, se realiza como en la siguiente figura. Para el cálculo de las intensidades (y por lo tanto de las resistencias) hay que tener en cuenta los siguientes factores:

- Hay distintos tipos de leds, con distintas tensiones en directo (V_F) y distintas intensidades luminosas en función de la corriente que los atraviesa, pero podemos tomar de forma general, que el led lucirá con corrientes por encima de 5mA. Como tensión en directo del led vamos a tomar 1'4V
- Los pines de del PIC16F84 funcionando como salida nos dan un nivel bajo $V_{OL(max)}$=0'6V
- Los pines del PIC16F84 pueden absorber, como máximo 25mA por pin, y en conjunto, el puerto B un máximo de 150mA.



- Vamos a tener varios leds conectados a cada salida del PIC, y varios encendidos a la vez. El peor caso para una salida (por ejemplo para RB2) tenemos conectados tres leds, los rojos de los semáforos S1 y S3. De forma que está limitada a 25mA, la corriente total que se puede entregar a los 4 leds. Se elegirá un valor de corriente de 8mA por led. En caso de querer más intensidad, se podría optar por un circuito del tipo que se explicará más adelante para los arrays de leds. El peor caso para el circuito en conjunto será prácticamente el caso general: cada semáforo tendrá encendida una luz para los coches y otra para los peatones, en total, 8 leds luciendo, al circular por ellos 8mA, como hemos dicho, el total, 64mA no supone ningún problema para el puerto B.

Con todo lo anterior, y teniendo en cuenta que el circuito va alimentado a V_{CC}=5V, la V_{OL}=0'6V y la V_{LED}=1'4V, en la resistencia, cuando el led conduzca, habrá una tensión de 3V, si queremos que circulen 8mA el valor de la resistencia será:

$$R = \frac{V_R}{I_R} = 3\frac{V}{8}mA = 375\,ohmios$$

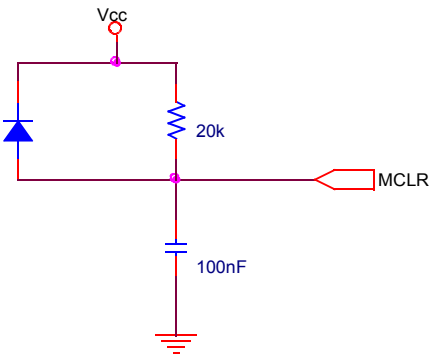
3. Circuito de reset

El PIC va equipado con un circuito interno de "Power On Reset" (Inicialización en el encendido), que inicializa el circuito al conectarse la alimentación. Para activarlo es suficiente con conectar la patilla MCLR (reset) directamente a V_{cc}. Así es como se ha realizado en el montaje práctico, funcionando correctamente.

Este sistema, sin embargo, tiene dos inconvenientes:

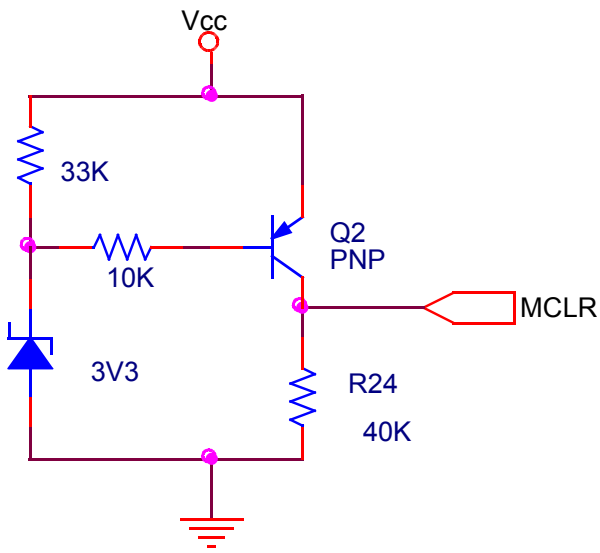
- Necesita una velocidad mínima de subida de la tensión de alimentación, de 0'05V/ms, que la proporciona sin problemas una fuente de laboratorio, pero quizá no otro tipo de fuente.
- La tensión debe empezar a subir desde V_{SS} para que se produzca el reset. Por lo tanto caso de una caída de la tensión por debajo del valor mínimo de la alimentación (4V) no se provoca un reset.

Para solucionar estos problemas, Microchip propone dos circuitos de reset distintos: para generar un POR con señales de alimentación que sean más lentas de 0'05 V/ms, se puede usar un circuito RC, que, al cargarse C lentamente, mantiene la señal de reset a nivel bajo mientras V_{DD} sube. El diodo permite que la tensión del condensador se descarge al apagar la alimentación, para evitar que la tensión en MCLR sea mayor que en V_{DD} y se pueda polarizar en directo alguna unión PN. Este circuito no resuelve el segundo problema.



Circuito de reset

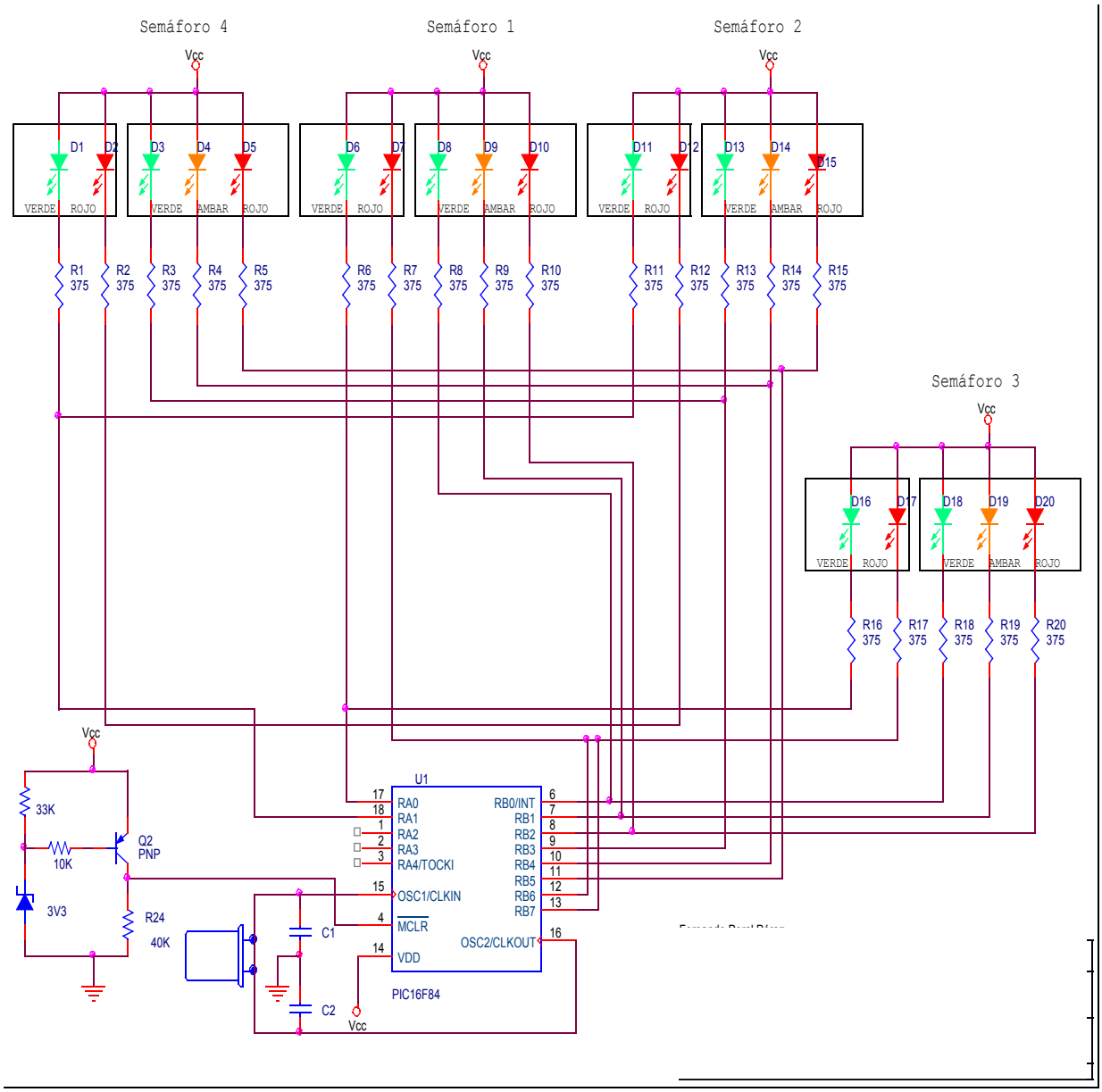
Para solucionar el segundo problema, Microchip propone varios circuito de "Brown-Out", uno de ellos es el representado en la figura. Cuando la tensión cae por debajo de V_Z + 0'7V (es decir, 4V), el transistor queda cortado y MCLR=0



Circuito de Brown Out

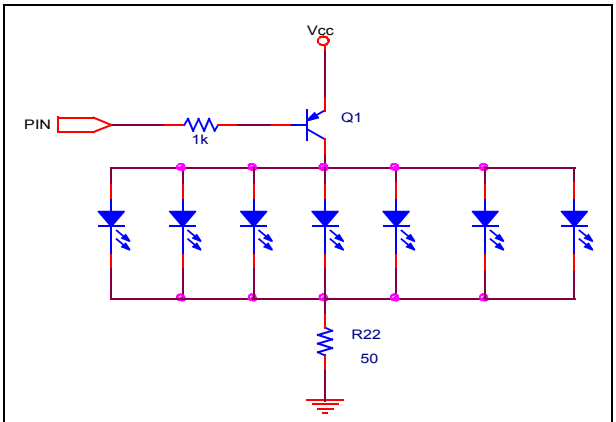
4. Diseño del circuito

Con todo lo expuesto anteriormente, el circuito completo, queda como se ve en el esquema de la página siguiente. En el se puede ver un bloque de leds para cada semáforo, con dos leds a la izquierda para el semáforo de peatones, y tres a la derecha para el de coches. Además se ha incluido el circuito de reloj típico propuesto por Microchip. En este circuito, el cristal de cuarzo elegido tiene una frecuencia característica de 2'4576MHz. Los condensadores, según catálogo deben tener un valor entre 15 y 33pF, por lo que se ha elegido un valor de 22pF para ambos.



5. Modificación del circuito para excitar un array de leds

En el caso de que se quiera sustituir uno de los leds por un array de leds, sería necesario entregar mucha más corriente, lo que no va a ser posible directamente con un pin del microcontrolador. Habrá que montar un circuito que se encargue de entregar la corriente como el que se muestra en la siguiente figura:



Array de leds

En este circuito el transistor trabajará en conmutación y será el encargado de entregar la

corriente a los leds. Para V_{PIN} =1 el transistor se encuentra cortado. Para V_{PIN}=0 habrá que asegurar que el transistor se encuentre en saturación. Para eso, teniendo en cuenta que queremos que circulen 8mA por led, en total tenemos (si hay 7 leds) 56mA. Tomando, como antes V_{LED} = 1'4V, y teniendo en cuenta V_{OL}=0'6V, V_{RC}=2'8V, y la resistencia de colector debe ser R_C=50 ohmios.

La de base deberá ser suficientemente pequeña para que la corriente de base meta al transistor en saturación y suficientemente grande para no "pedir" demasiada corriente al PIC. Suponiendo que el transistor tenga una β mínima de 50, se debe cumplir $I_{C(SAT)}$=56mA < βI_B de donde I_B>1'12mA, y para esa I_B mínima sale un valor de resistencia máximo para la resistencia de base:

$$R_{B(max)} = \frac{V_R}{I_{B(min)}} = \frac{V_{cc} - V_{OL} - V_{BE(sat)}}{I_{B(min)}} = \frac{5 - 0'6 - 0'7}{1'12\,mA} = 3\,K3$$

Pero con una resistencia de 1K obtenemos I_B=3'7mA, que no es demasiada corriente para el PIC, y el transistor está de sobra metido en saturación (incluso valdría un transistor con una β mínima de 15).

6. Desarrollo del programa

6.1 Diseño a alto nivel (pseudocódigo)

6.1.1 Introducción

La temporización se va a hacer utilizando el temporizador del microcontrolador, que activa una interrupción en desbordamiento. El programa constará de dos partes: el programa principal, que va a realizar la inicializador del microcontrolador, y la rutina de atención a la interrupción, que hará el resto del trabajo.

Como variables tenemos "S1" y "S2", variables semáforo, que pueden tomar valores, Rojo, Ambar, Verde y Apagado, P1 y P2, variables semáforo de los peatones, que pueden tomar los valores Rojo, Verde y Apagado, y una variable contador, para llevar cuenta del tiempo.

Para minimizar los errores de temporización, se va a reprogramar el temporizador al inicio de la rutina de interrupción, con un periodo fijo, de 0'5s. El contador, por lo tanto, contará periodos de 0'5s.

6.1.2 Programa principal

```
Inicializa micro.
Programa PA salida
Programa PB salida
contador:=0
S1:=Verde
P1:=Rojo
S2:=Rojo
P2:=Verde
Programar interrupciones          (*habilita IRQ del temporizador *)
Programar temporizador para disparo 0'5s
REPETIR                           (* bucle infinito *)
    borra perro guardián
SIEMPRE
```

6.1.3 Rutina de atención a la interrupción del temporizador

```
(* No se comprueba la fuente de interrupción porque sólo se habilita la del temporizador *)
Reprogramar Timer a 0'5s
contador:=contador+1
SEGUN SEA contador
    CASO contador<40    (*hasta 20s *)
    INICIO
        S1=Verde
        P1=Rojo
        S2=Rojo
        P2=Verde
    FIN
    CASO 40<=contador<50      (*entre 20s y 25s *)
    INICIO
        S1=Verde
        P1=Rojo
        S2=Rojo
        SI contador es par      (*parpadeo una vez de cada dos, es decir, cada segundo *)
            conmuta P2.Verde     (*si está encendido pasa a apagado y viceversa*)
        FINSI
    FIN
```

```
CASO 50<=contador<60      (*entre 25s y 30s *)
INICIO
        apaga S1.Verde
        conmuta S1.Ambar    (* parpadeo cada 0'5s *)
        P1=Rojo
        S2=Rojo
        SI contador es par      (*una vez de cada dos, es decir, cada segundo *)
                conmuta P2.Verde     (*si está encendido pasa a apagado y viceversa*)
        FINSI
FIN
CASO 60<=contador<100     (*entre 30s y 50s *)
INICIO
        S1=Rojo
        P1=Verde
        S2=Verde
        P2=Rojo
FIN
CASO 100<=contador<110   (*entre 50s y 55s *)
INICIO
        S1=Rojo
        SI contador es par      (*parpadeo una vez de cada dos, es decir, cada segundo *)
                conmuta P1.Verde     (*si está encendido pasa a apagado y viceversa*)
        FINSI
        S2=Verde
        P2=Rojo
FIN
CASO 110<=contador<120   (*entre 55s y 60s *)
INICIO
        S1=Rojo
        SI contador es par      (*una vez de cada dos, es decir, cada segundo *)
                conmuta P1.Verde     (*si está encendido pasa a apagado y viceversa*)
        FINSI
        apaga S2.Verde
        conmuta S2.Ambar    (* parpadeo cada 0'5s *)
        P2=Rojo
FIN
```

```
    ENOTROCASO              (* se pasó de 60 *)
    INICIO
        S1:=Verde
        P1:=Rojo
        S2:=Rojo
        P2:=Verde
        contador:=0
    FIN
FINCASO
Reactiva interrupción
retorno de interrupción.
```

6.2 Codificación del programa.

```
      list p=p16f84, r=dec
      #include <p16f84.inc>


      __CONFIG _CP_OFF & _PWRTE_OFF & _WDT_ON & _XT_OSC

      org 0
      goto principal        ;salto al prrograma principal al resetear

      org 4
      goto irq              ;salto a la rutina de interrupcion


;definicion de constantes
cuenta      equ 16  ;valor de cuenta del TMR0  (256-240)
RP2   equ 7         ;bits de los semaforos de peatones
RP1   equ 6
VP2   equ 1
VP1   equ 0

;declaracion de variables (posiciones de memoria que usan)
contador equ 0x10
retardo  equ 0x11


;*****************************************************************************
; Programa principal
;
; Inicializacion del micro: registros de control, puertos e interrupciones.
; Programa y lanza el temporizador para que se active la interrupcion, y luego
; se queda en espera en un bucle infinito
;

principal:
      ;inicializacion del micro: programacion del registro option
      ;bit7. RPBU=1  resistencias de pull-up internas del puerto B deshabilitadas
      ;(no es necesario, ya que se deshabilitan al usarlo como salida)
      ;bit6. INTEDG=0.  Da igual, ya que no usamos INT
      ;bit5. TOCS=0     Temporizador con reloj interno
      ;bit4. TOSE=0  Da igual
      ;bit3.  PSA=0  Prescaler para el TMR0
      ;bits2-0 = 111 Ajuste del prescaler del TMR0   a 256

      movlw b'10000111'     ;palabra para OPTION hay que modificar bits 2-0
      bsf STATUS, RP0       ;banco 1
      movwf OPTION_REG      ;escribe la palabra en el registro OPTION
      bcf STATUS, RP0       ;banco 0
```

```
;programacion de los puertos
;PuertoB=salidas => TRISB= todo a cero
;Estructura de PB:    7   6   5  4  3  2  1  0
;                    RP2 RP1  R2 A2 V2 R1 A1 V1
;                              --^--    --^--
;                                S2      S1
bsf STATUS, RP0       ;banco 1
clrf TRISB            ;borra todos los bits, programando PB como salida
clrf TRISA            ;PA tammbién salida
bcf STATUS, RP0       ;banco 0

;inicializacion de contador y puerto
clrf contador         ;contador=0
movlw b'10011110'     ;S2 rojo, S1 verde. Peatones, Rojo para S1
movwf PORTB           ;lo saca por el puerto
bcf PORTA, VP2        ;peatones, Verde para S2
bsf PORTA, VP1



;programacion del temporizador
movlw cuenta
movwf TMR0            ;cuenta inicial del temporizador, lanza la cuenta.

;programacion de interrupciones:
;Bit7. GIE=1   para habilitar las interrupciones
;bit5. TOIE=1  para habilitar la interrupcion del temporizador
;resto a 0 para deshabilitar las demas interrupciones y borrar flags
movlw b'10100000'
movwf INTCON          ;programa las interrupciones




espera:               ;bucle infinito que borra continuamente el perro guardian
       clrwdt         ;el trabajo lo hace la irq. Solo si hay algun problema, se
goto espera           ;reiniciara el micro
```

```
;*************************************************************************************
; Rutina de atencion a la interrupcion del temporizador
;
; La interrupcion se producira cada 0'05s.
; Contaremos 5 llamadas
; Se incrementara un contador de segundos para controlar la temporizacion del semaforo, y
; se modifica el estado de los semaforos, escribiendo en el puerto segun el tiempo pasado
;
; No se comprueba cual es la fuente de interrupcion, porque solo esta habilitada
; la interrupcion del temporizador.

irq:
     ;empezamos reprogramando el temporizador, para que vaya contando ya, y no introducir
     ;un retardo en la temporizacion debido al proceso de varias linea de codigo
     movlw cuenta
     movwf TMR0     ;reprogramacion del timer

     incf retardo
     movlw 5
     subwf retardo, w      ;compara retardo con 5
     btfss STATUS, Z       ;si no es 5
     goto finswitch        ;salta al final de la rutina de interrupción

     clrf retardo

;este código se ejecutará cada 0'5s. El contador cuenta medios segundos
     incf contador, f      ;contador=contador+1


     ;implementacion de la sentencia switch-case
caso20:     ;contador<20?
     movlw 40
     subwf contador, w     ;compara contador con 40
     btfsc STATUS, C               ;¿acarreo?
     goto fincaso20        ;no hay acarreo => contador>=40

     ;contador<40 (no hay acarreo)
     movlw b'10011110'     ;Coches: S2 rojo, S1 verde, peatones RP2 OFF, RP1 ON
     movwf PORTB           ;lo escribe en el puerto
     bcf PORTA, VP2                ;peatones, Verde para S2
     bsf PORTA, VP1
     goto finswitch
fincaso20:
```

```
caso25:     ;contador<25?
     movlw 50
     subwf contador, w     ;compara contador con 50
     btfsc STATUS, C               ;¿acarreo?
     goto fincaso25        ;no hay acarreo => contador>=50

     ;contador<25 (no hay acarreo)
     movlw b'10011110'     ;Coches: S2 rojo, S1 verde. Peatones: RP2 OFF, RP1 ON
     movwf PORTB           ;lo escribe en el puerto

     btfsc contador, 0     ;si contador par (1 segundo)
     goto finswitch
     movlw b'10'           ;mascara XOR para conmutar VP2 sin modificar el resto
     xorwf PORTA, f        ;conmuta VP2  (cada 1s)

     goto finswitch
fincaso25:
caso30:     ;25<=contador<30?
     movlw 60
     subwf contador, w     ;compara contador con 60
     btfsc STATUS, C               ;¿acarreo?
     goto fincaso30        ;no hay acarreo => contador>=60

     ;contador<30 (no hay acarreo)
     movlw b'00000010'     ;mascara XOR para conmutar S1.Ambar sin modificar el resto
     xorwf PORTB, f        ;conmuta S1.Ambar  (cada 0'5s durante el tiempo entre 25 y 30)
     bsf PORTB, 0          ;apaga S1.Verde

     btfsc contador, 0     ;si contador par (cada segundo)
     goto finswitch
     movlw b'10'           ;mascara para VP2
     xorwf PORTA, f        ;conmuta VP2

     goto finswitch ;
fincaso30:
caso50:     ;30<=contador<50?
     movlw 100
     subwf contador, w     ;compara contador con 100
     btfsc STATUS, C               ;¿acarreo?
     goto fincaso50        ;no hay acarreo => contador>=100

     ;contador<50 (no hay acarreo)
     movlw b'01110011'     ;Coches: S2 verde, S1 rojo. Peatones:RP2 ON, RP1 OFF
     movwf PORTB           ;lo escribe en el puerto
     bsf PORTA, VP2                ;peatones, Verde para S1
     bcf PORTA, VP1
     goto finswitch ;
fincaso50:
```

```
caso55:     ;30<=contador<55?
     movlw 110
     subwf contador, w     ;compara contador con 110
     btfsc STATUS, C               ;¿acarreo?
     goto fincaso55        ;no hay acarreo => contador>=110

     ;contador<55 (no hay acarreo)
     movlw b'01110011'     ;S2 verde, S1 rojo. Peatones:RP2 ON, RP1 OFF
     movwf PORTB           ;lo escribe en el puerto

     btfss contador, 0     ;contador ¿par? => 1 segundo
     goto finswitch

     movlw b'01'           ;mascara XOR para conmutar VP1 sin modificar el resto
     xorwf PORTA, f        ;conmuta VP1  (cada 1s)
     goto finswitch ;
fincaso55:

caso60:     ;55<=contador<60?
     movlw 120
     subwf contador, w     ;compara contador con 120
     btfsc STATUS, C               ;¿acarreo?
     goto fincaso60        ;no hay acarreo => contador>=120

     ;contador<60 (no hay acarreo)
     movlw b'00010000'     ;mascara XOR para conmutar S2.Ambar sin modificar el resto
     btfss contador, 0     ;si contador par (cada segundo)
     bsf W, VP1            ;activa la mascara también para      VP1
     xorwf PORTB, f        ;conmuta S2.Ambar  (cada 0'5s durante el tiempo entre 55 y 60)
     bsf PORTB, 3          ;apaga S2.Verde

     btfsc contador, 0     ;si contador par (cada segundo)
     goto finswitch
     movlw b'01'           ;mascara para VP1
     xorwf PORTA, f        ;conmuta VP1


     goto finswitch ;
fincaso60:

otherwise:  ;llego a 60
     movlw b'10011110'     ;Coches: S2 rojo, S1 verde. Peatones: RP2 OFF, RP1 ON
     movwf PORTB           ;lo escribe en el puerto
     bcf PORTA, VP2                ;peatones, Verde para S2
     bsf PORTA, VP1
     clrf contador         ;empieza el ciclo de nuevo

finswitch:  ;fin de la sentencia switch-case
```

```
        movlw b'10100000'
        movwf INTCON
        retfie                ;retorno de interrupcion

    end
```

7. Cuestiones sobre la temporización

- Para generar el reloj del microcontrolador se utiliza un cristal de 2'4576 Mhz, por lo tanto el periodo del reloj es de 406'9ns.
- El reloj de instrucción es 4 veces mayor que el periodo de reloj, esto es, 1'6276us.
- Se ha ajustado el prescaler del temporizador al máximo, esto es 256, con lo que cada flanco de reloj del temporizador se produce cada 0'41667 ms.
- Ajustando la cuenta del temporizador a 16 para que el temporizador cuente 240 pulsos antes de desbordarse (ya que cuenta ascendentemente), el desbordamiento del temporizador (y por lo tanto la interrupción) se produce una vez cada 0'1s (exactos).
- Para el funcionamiento exacto de la temporización, necesitaríamos que la interrupción se produjera cada 0'5s, como el contador es de 8bits, no da para más, de forma que se introduce un contador auxiliar que cuenta cinco llamadas a la interrupción “antes de hacer algo”.
- Hay un factor de error: el tiempo que pasa desde que se produce la interrupción, hasta que volvemos a programar (y por lo tanto disparar) el temporizador. Este tiempo está formado por los siguientes componentes:
  - Tiempo de latencia de la interrupción, esto es, tiempo desde que se produce el suceso hasta que se ejecuta el código. Para interrupciones síncronas (internas) es de 3 ciclos de instrucción
  - Tiempo de ejecución de la sentencia GOTO irq = 2 ciclos de instrucción
  - Tiempo de reprogramación del temporizador (2 instrucciones) = 2 ciclos de instrucción.
  - Inhibición del temporizador desde su programación = 2 ciclos de instrucción.

En total, un error (retardo) de 9 ciclos de instrucción, esto es, aproximadamente 14'65us, error que se produce en cada llamada a la interrupción, esto es, cada 0'1s. Lo que significa que la temporización va a tener un error de casi el 0'015% (se retrasará), lo que equivale a unos 12'65s a lo largo del día.

Teniendo en cuenta que el error es un error sistemático, si que podría compensarse, usando un reloj un poco más rápido, de forma que la llamada a la interrupción, contando todos los retardos que se han descrito antes, fuera de exactamente cada 0'1s.

8. Consumo de potencia

Según el catálogo del PIC16F84, el consumo de potencia en el PIC se calcula de la siguiente forma: $P = V_{DD} \cdot (I_{DD} - \sum I_{OH}) + \sum [(V_{DD} - V_{OH}) \cdot I_{OH}] + \sum (V_{OL} \cdot I_{OL})$

Hay que tener en cuenta que I_{DD} es la corriente de alimentación medida entre V_{DD} y el pin de alimentación (V_{DD}) del PIC. El sentido de cada uno de los términos es el siguiente:

- El primero (I), $V_{DD} \cdot (I_{DD} - \sum I_{OH})$ es el consumo de potencia debido al propio funcionamiento del PIC, ya que la corriente que se tiene en cuenta, es la corriente de alimentación (la que entra por V_{DD}) menos la que entregan las salidas (las que están a nivel alto, que son las que entregan corriente).
- El segundo (II), $\sum [(V_{DD} - V_{OH}) \cdot I_{OH}]$ es el consumo de potencia en las resistencias internas de los transistores de pull-up de las salidas que están a nivel alto. En cada uno de esos transistores cae una tensión V_{DD}-V_{OH}, y la potencia disipada en el será proporcional a la corriente que entregue (la que sale por la salida correspondiente)
- El tercero (III), $\sum (V_{OL} \cdot I_{OL})$ es el consumo de potencia en las resitencias internas de los transistores de pull-down de las salidas que están a nivel bajo. En cada uno de esos transistores cae una tensión V_{OL} y la corriente que circula es la que absorbe esa salida.

Además de esta potencia, que es la que consume el PIC, el circuito consumirá potencia (la mayor parte) en los leds y las resistencias de los leds. En cada conjunto led-resistencia (cuando estén conduciendo), y teniendo en cuenta que están todos excitados por señales activas a nivel bajo, caerá una tensión (V_{DD}-V_{OL})•I_{OL}. (IV)

En conjunto, como no se utilizan en ningún momento salidas activas a nivel alto, (I) queda V_{DD}•I_{DD}, (II) es cero, y la suma de (III) y (IV) es $\sum (V_{DD} \cdot I_{OL})$.

Esta potencia será variable en función del número de leds que estén encendidos en cada momento. En el peor caso, tenemos encendidos 2 leds por cada semáforo, es decir, 8 ,leds, conduciendo cada uno I_{OL}= 8mA, y el PIC, según he medido, absorbe I_{DD}=0'47mA, por lo que en total:

$$P = V_{DD} \cdot I_{DD} + \sum (V_{DD} \cdot I_{OL}) = 0'47\,mA \cdot 5\,V + 8 \cdot 5\,V \cdot 8\,mA = 2'35\,mW + 320\,mW = 322'35\,mW$$

donde se ve que la mayor parte del consumo es debido a las salidas.

De este consumo, la parte que se produce dentro del propio micro es, como máximo:

$$P = V_{DD} \cdot I_{DD} + \sum (V_{OL} \cdot I_{OL}) = 2'35\,mW + 8 \cdot 0'6\,V \cdot 8\,mA = 2'35\,mW + 38'4\,mW = 40'75\,mW$$

Para ver el consumo medio, podemos tomar 30 segundos, ya que la situación se repite durante los 30s siguientes:

- Durante 20s están encendidos 8 leds.
- Durante 5s están encendidos 6 leds y los dos restantes el 50%, como media, 7'5leds
- Durante 5s están encendidos 4 leds, y los otros 4 al 50%, como media, 6 leds.

La media durante los 30s, es de 7'5 leds, por lo que repitiendo los cálculos anteriores, sale un consumo medio de: 302'35mW, de los que 38'35mW se disipan dentro del PIC.