# Diseño de un servomotor controlado por bus $I^2C$ mediante un microcontrolador de gama media

---

**Alejandro Alonso Puig – mundobot.com**
**Septiembre 2003**

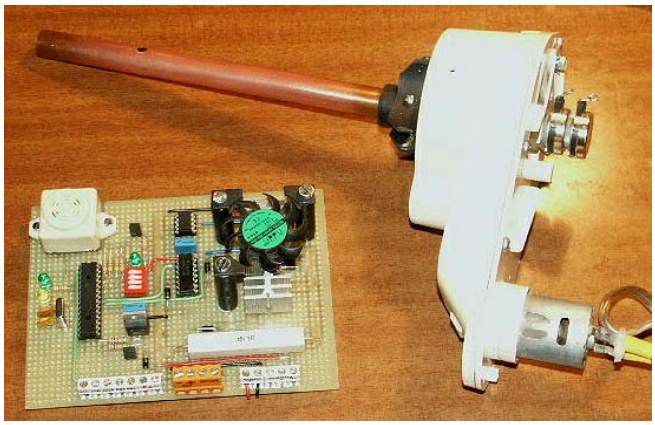
# Introducción

Este siguiente informe técnico describe el diseño, tanto desde el punto de vista electrónico, como informático de un modelo de servomotor cuya particularidad consiste en ser controlado por bus $I^2C$.

Las características principales del módulo presentado son las siguientes:

- Actúa como Slave permitiendo seleccionar mediante switches dip la dirección que utilizará en la red $I^2C$.
- Se puede establecer mediante bus $I^2C$ tanto la posición deseada, como el DeadBand
- Se puede obtener en todo instante mediante bus $I^2C$ el valor del consumo de corriente del módulo, la temperatura, la posición actual del eje así como otros ciertos valores de estado
- El módulo está protegido para evitar sobrecalentamiento del mismo mediante sensor de temperatura que activa un mecanismo de ventilación e incluso la parada del servomotor para evitar daños internos.

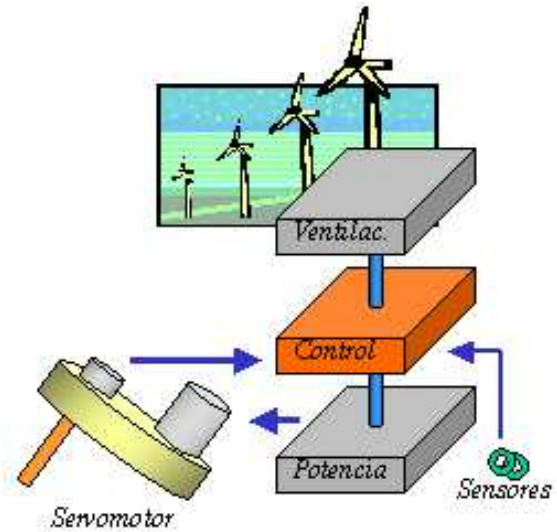
La ventaja que se obtiene con este tipo de módulos es precisamente un control completo por bus $I^2C$ que hace innecesario tener módulos específicos para control de servos como ocurre con los de control tipo PWM. De esta manera pueden controlarse gran cantidad de servos desde un controlador principal sin apenas sobrecarga en el mismo. Adicionalmente se tienen medida no habituales, como la de posición real, que permite a nivel de microcontrolador principal, saber si el servo llegó realmente a su destino o encontró algún obstáculo que ejercía mayor fuerza que su par. Igualmente la medida de consumo eléctrico permite conocer cuando el servomotor tiene una sobrecarga de fuerza contraria a su dirección de movimiento, lo que provoca una subida del consumo eléctrico medible en este caso.



# Diseño Electrónico

El módulo está básicamente compuesto por:

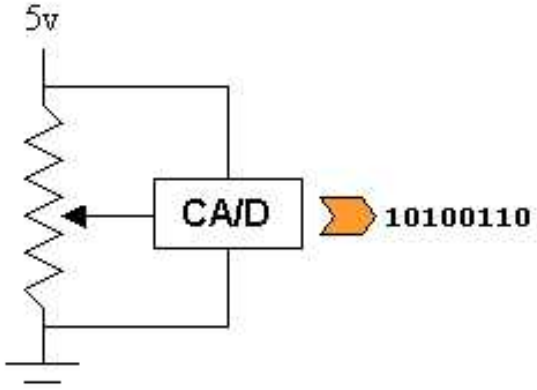
- Servomotor
- Sección de control
- Sección de potencia para el servomotor
- Sección de ventilación
- Sensores
- Alimentación



## *Servomotor*

La reductora como es bien sabido permite una reducción de la velocidad de giro del eje a favor de un aumento del par del mismo.

En la parte trasera del eje se ha adaptado un potenciómetro lineal de 10KΩ utilizado como divisor de tensión entre 0 y 5v. La salida del mismo irá a una entrada del microcontrolador, configurada para hacer una conversión analógico-digital de 8 bits de resolución, lo que implica que se podrán establecer 256 puntos de posicionamiento en el servo.




El eje de la reductora se ha provisto de topes físicos para evitar que un giro incontrolado pueda dañar los topes del potenciómetro.

## *Sección de control*

La sección de control está basada en un microcontrolador PIC 16F876 de Microchip funcionando a 4Mhz.

Este microcontrolador posee un módulo MSSP que permite el control de las comunicaciones $I^2C$ por Hardware. Se utilizan los pines SCL (RC3) y SDA (RC4) conectados a resistencias Pull-Up de 1KΩ para las comunicaciones $I^2C$.

Adicionalmente posee un grupo de conversores analógico digitales (A/D) de los que se utilizarán tres. Uno (RA0/AN0) para la identificación del feedback del potenciómetro, que servirá para obtener la posición del eje del servomotor; otro (RA3/AN3) para convertir la señal analógica lineal (10mV/ºC) proveniente de un sensor de temperatura (LM35), que permitirá prevenir sobrecalentamientos del módulo; y otro (RA1/AN1) para medir la caída de tensión en una resistencia cementada de 1Ω colocada en serie con la alimentación del módulo de potencia, lo que permitirá identificar el consumo de corriente del mismo y del servomotor.

Se utilizan adicionalmente 2 salidas para la señalización de alarmas por sobrecalentamiento. Una (RA2) activará la sección de ventilación e indicará su actividad mediante un led. La otra (RA5) se activará solo cuando la actividad del servomotor haya sido detenida por programa debido a un sobrecalentamiento excesivo. Esta salida tiene asociado un led y está conectada a un driver que permite soportar una carga de 2 amperios. Por una parte se le conecta un buzzer de alarma, pero por otra parte se dejan salidas (X3 en esquema) para poder conectarle sistemas adicionales de acción que no superen los 2 amperios mencionados.

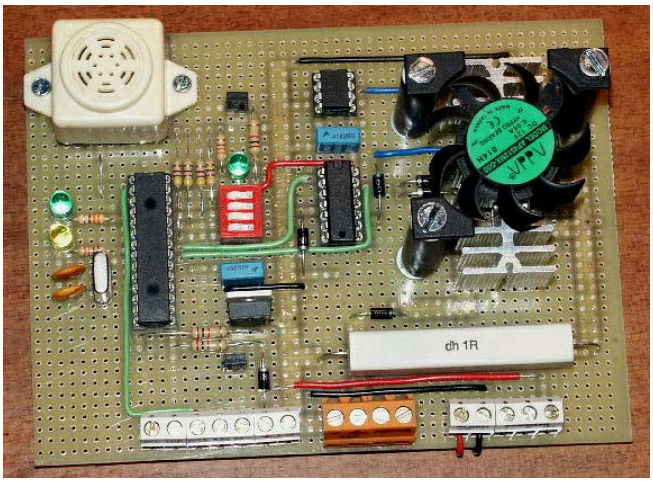
Existen 4 entradas (RB2-RB5) conectadas a un switch-dip cuádruple. Solo las dos inferiores (RB2, RB3) son utilizadas por el firmware actual (versión 4) y permitirán seleccionar diferentes direcciones $I^2C$ para evitar coincidencia de

direccionamiento con otros dispositivos (entre 01111000 [0x78] y 01111110 [0x7E]).
Dos de las salidas (RB0 y RB1) se utilizarán para el control del servomotor. Para limpiar dichas señales de ruidos indeseados, se han filtrado a través de dobles inversores Schmith Trigger (74LS14)

Finalmente los pines MCLR, RB6 y RB7 se han llevado a un conector triple externo (X1 en esquema) con el fin de permitir la programación del chip una vez montado en placa. Un jumper (JP3 en esquema) permite desconectar MCLR de la resistencia de PullUp en el momento de la programación

## *Sección de potencia*

La sección de potencia permitirá dar al servomotor la corriente que requiere para su funcionamiento. Para ello se han utilizado sendos drivers TC4422 de Microchip que permiten consumos pico de hasta 9 amperios. Dichos drivers están protegidos de las corrientes de autoinducción del motor por un puente de 4 diodos (1N4007).




Es importante tener en cuenta que la masa utilizada por la sección de potencia es diferente a la masa utilizada por el resto de las secciones. La razón es que ambas masas están separadas por la resistencia cementada de 1Ω que nos permite medir el consumo de corriente del servomotor. Por ello existe una pequeña diferencia de potencial correspondiente a la caída de tensión en esta resistencia. Debido a esto y para evitar corrientes inversas de los drivers hacia la sección de control en los flancos de bajada, se utilizan dos diodos (1N4007) de protección.

## *Sección de ventilación*

Para proteger el sistema de sobrecalentamientos, existe una sección de ventilación activada por la sección de control.

Básicamente el sistema está formado por un pequeño ventilador y el driver TC4425 de Microchip. Cuando la sección de control identifica que la temperatura ha superado los 46ºC, activa RA2. Su señal es filtrada por una puerta inversora Schmith Trigger y llevada al driver mencionado. Dicho driver activa el ventilador.

Una vez que la sección de control considera que la temperatura ha bajado a límites adecuados (<40ºC), detiene el ventilador.

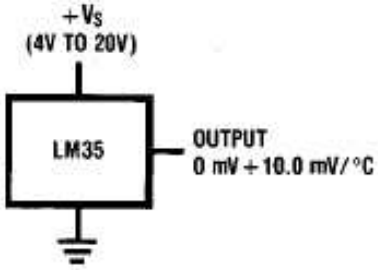
Cuando la temperatura sube en exceso (54º), la sección de control detendrá el servomotor, pero la sección de ventilación seguirá activa a la vez que se activa la alarma.

## *Sensores*

El sistema utiliza básicamente dos sensores:

**Sensor de temperatura** (LM35): Este sensor produce una variación de potencial en su salida de 10mV por grado centígrado. En la forma en que

se ha colocado en el circuito, permite un rango de medida de +2ºC hasta 150ºC, aunque existen otras configuraciones que permiten medidas de –55ºC hasta 150ºC (Consultar hoja técnica del fabricante para más detalles).




Este sensor está unido al radiador de los drivers de potencia del servomotor. Por tanto solo mide la temperatura de dichos drivers y no del servomotor u otros componentes de la circuitería.

**Sensor de corriente**: En realidad no es más que la resistencia de 1Ω que se ha comentado anteriormente. Esta resistencia separa la masa de la sección de potencia y servomotor de la del resto del sistema. Es una resistencia cementada resistente al sobrecalentamiento en caso de alto consumo de corriente por parte del servomotor. Como sabemos por la ley de ohm, la caída de tensión en una resistencia es directamente proporcional a la corriente que circula por ella. (V=IxR). Si además la resistencia es de 1Ω, V=Ix1=I, es decir que el valor de caída de potencial es idéntico a la corriente que circula por la resistencia, e igual a la corriente que circula por el servomotor (menos una pequeña corriente consumida en el driver). La realidad es que es difícil una precisión perfecta en las resistencias para que sean de 1Ω exacto, pero en cualquier caso el sistema nos resulta igualmente válido siempre que midamos previamente al montaje el valor de la resistencia y lo consideremos en la fórmula de ohm para un conocimiento exacto de la corriente circulante.


La masa de la sección de control y la sección de potencia están separadas por esta resistencia, por lo que la entrada al conversor A/D será precisamente la masa de la sección de potencia, que variará según la corriente circulante.

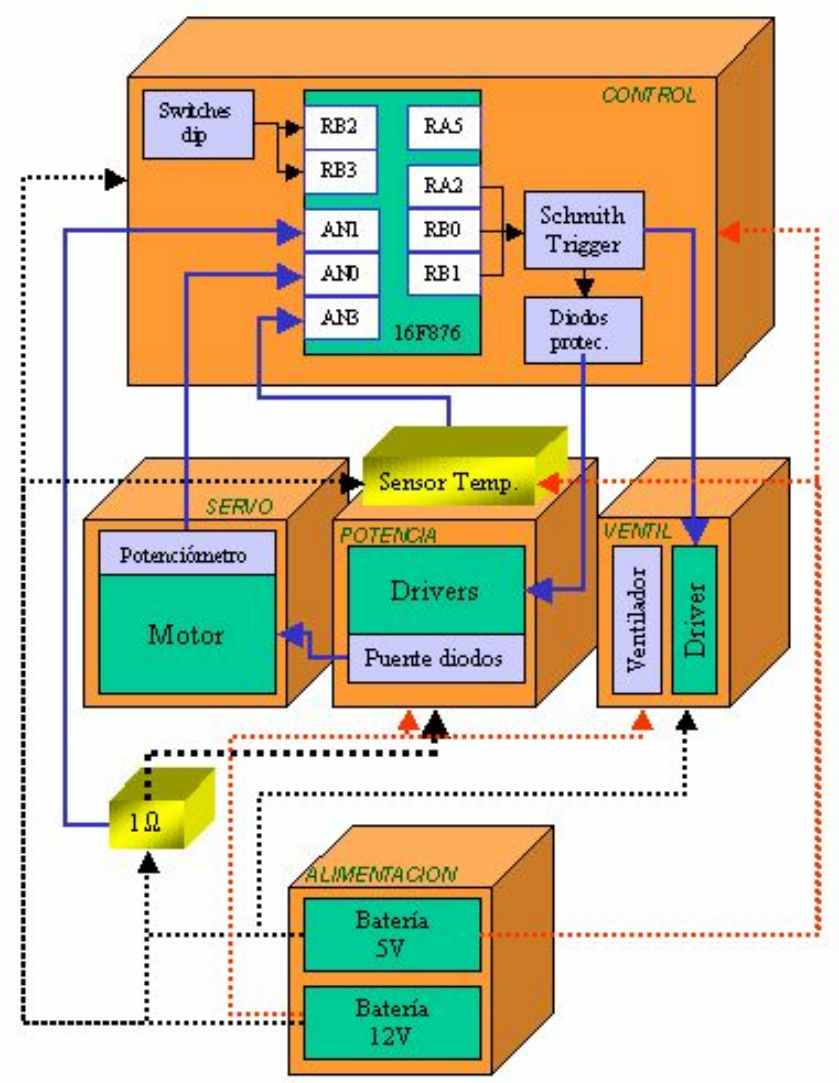
Al utilizar una resistencia de 1Ω, la corriente máxima circulante puede ser de 5 amperios. Si supera este valor se produciría una caída de potencial superior a 5v que al entre en el conversor A/D podría dañar la sección de control. Por eso, aunque el tipo de motor utilizado tiene un consumo de 1 amperio en estado de bloqueo, es conveniente instalar un fusible de 4-5 amperios entre el polo negativo de la fuente de alimentación y la masa de la circuitería.

**Potenciómetro**: Como ya se ha comentado con anterioridad, el potenciómetro es utilizado como divisor de tensión para que permita medir la posición del eje del servomotor en cada momento.

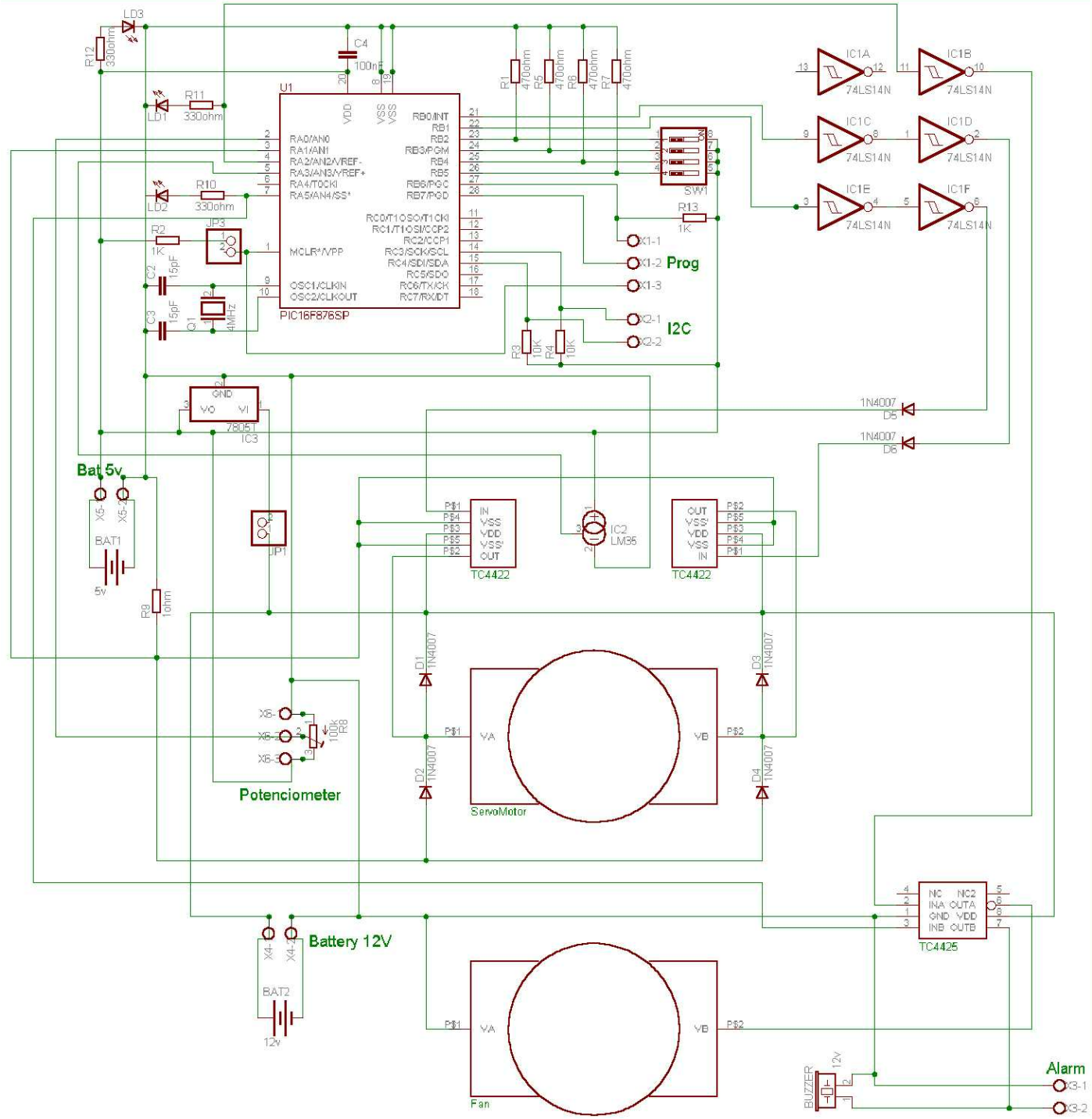
## *Sección de Alimentación*

El circuito diseñado permite utilizar dos fuentes de alimentación, de 5v y 12v. La masa de ambas fuentes de alimentación será común para obtener las mismas referencias. La fuente correspondiente a 12v alimentará el servomotor, la sección de potencia y la de ventilación, mientras que la de 5v alimentará la sección de control, el sensor de temperatura y el potenciómetro (divisor de tensión) del servomotor. Es costumbre habitual separar las fuentes de alimentación de las áreas de potencia y control para evitar que las variaciones provocadas por las áreas de potencia afecten a las áreas de control. En cualquier caso este circuito incluye la posibilidad de alimentarlo exclusivamente con una fuente de alimentación de 12v. Para ello se ha dispuesto un jumper (JP1 en esquema) que permite o no obtener los 5 v necesarios a través de un regulador de tensión 7805. En las pruebas realizadas no se han percibido ruidos de importancia en la circuitería de control.

A continuación se muestra un esquema por bloques del sistema descrito. Las líneas de trazado discontinuo indican las líneas de alimentación. Como se observará hay una línea negra más gruesa que las demás. Esta indica el otro plano de masas diferente al común y que es utilizado por lo unidad de potencia.



El esquema eléctrico completo se muestra a continuación.



# Diseño del software

El software ha sido desarrollado en ensamblador para PIC16F876.

El módulo está configurado para producir una interrupción cada vez que se produce un evento $I^2C$. Al atender dicha interrupción comprueba la información que le llega al microcontrolador y procede en consecuencia.

## *Registros*

El módulo contiene una serie de registros que pueden ser accedidos desde un master por $I^2C$. Algunos de estos registros son solo de lectura, mientras que otros son de lectura y escritura. Caso especial es uno de los registros, denominado "Estado", que tiene algunos bits que solo son de lectura y otros que son de lectura/escritura (R/W)

Mediante el manejo de estos registros se obtiene toda la funcionalidad del módulo. A continuación describimos dichos registros:

- **PosAct y PosNew:** En realidad son accedidos desde el master como un único registro. Cuando es accedido en modo escritura, el dato se escribe sobre PosNew y cuando es accedido como lectura, el dato devuelto es el contenido de PosAct. PosNew indica la posición que se desea que tenga el eje del servomotor, mientras que PosAct indica la posición real del eje en el instante en que es accedido el registro. El rango de valores aceptables para el posicionamiento del eje está entre 0+HalfDeadBand+1 y 255-HalfDeadBand-1, no obstante, en caso de solicitar una posición fuera de este rango, el programa forzará la posición a los límites indicados para evitar comportamientos incorrectos. Esto implica que en todo momento se podrán seleccionar

  (255-HalfDeadBand-1) - (0+HalfDeadBand+1) +1

  es decir,

  254-2xHalfDeadBand posiciones.

  La posición por defecto que mantendrá el módulo al arrancar será la que tenga en ese momento, no variándola hasta que le llegue tal orden por $I^2C$.

- **HalfDeadBand**: Es un registro R/W. Indica la mitad del tamaño de la banda muerta (DeadBand), que es una franja de error aceptable en el posicionamiento del eje del servomotor. Sin esta franja el servomotor estaría constantemente tratando de posicionarse en el lugar exacto, impidiéndoselo la inercia del motor. El rango de valores válidos está entre 0 y 255. El valor por defecto que utilizará el módulo si no se le indica otro será 3.

- **Corriente**: Es un registro accedido solo en modo lectura. Permite calcular el consumo de corriente de la sección de potencia y servomotor. La medida considerada en realidad en este registro en la caída de potencial en la resistencia de 1Ω mencionada en la sección de Diseño Electrónico. La resolución tomada por el conversor A/D es de 19,5mV/bit. Es decir que por ejemplo, un valor de este registro de 37, implicaría una caída de tensión en la resistencia de 37x19,5mV= 721,5mV. Es importante conocer el valor real de la resistencia si se desea tener valores precisos de consumo. Si la resistencia realmente tiene 1Ω, el consumo será de 721,5mA, pero si el valor real de la resistencia es por ejemplo de 1,5Ω, el valor de consumo de corriente según la ley

de ohm será I = V/R = 721,5mV/1,5Ω = 481mA

- **Temperatura**: Es un registro de solo lectura. Indica la temperatura de la sección de potencia del servomotor. Es utilizada internamente para la activación del sistema de ventilación o bloqueo de la unidad en caso de sobrecalentamiento. La resolución usada en la conversión A/D es de 5mV/bit, que según las especificaciones del sensor lineal de temperatura utilizado corresponde a 1/2 grado/bit. Ha de tenerse en cuenta adicionalmente que la configuración establecida para el sensor hace que su rango de valores esté entre 2ºC y 150ºC, por lo que una medición de 0v corresponde a 2ºC. Así un valor en este registro de por ejemplo 37 corresponde con un valor en grados centígrados de ºC = (Temperatura/2)+2 = (37/2)+2 = 20,5º

- **Estado**: Se trata de un registro especial en el que el bit de mayor peso es R/W y el resto son solo de lectura. Los valores de los bits son los siguientes:

    - ***Bit 0***: Ventilador On/Off. Solo lectura.
    - ***Bit 1***: servomotor detenido por exceso de temperatura On/Off. Solo lectura.
    - ***Bits 2 y 3***: Estado posicionamiento servo. Si ambos On, está posicionado. Si solo uno encendido indica que aún no está posicionado y el sentido de giro . Solo lectura.
    - ***Bits 4, 5 y 6***: No usados
    - ***Bit 7***: Indica si el servomotor ha sido detenido por orden $I^2C$. Tipo R/W

- **Versión**: Indica la versión del firmware. Es un registro de solo lectura.

## *Cuerpo del programa*

Básicamente el funcionamiento del programa consiste en un bucle en el que mientras le llega un comando $I^2C$, trata de mantener el eje del servomotor en la posición indicada por "PosNew", a la vez que actualiza los valores de los demás registros. Inicialmente "PosNew" tendrá el mismo valor que "PosAct", por lo que si no se le indica por $I^2C$ que varíe su posición, se mantendrá de forma activa en la posición que estaba.

El chequeo del valor del sensor de temperatura y consiguiente actualización de su registro asociado, se hace junto con un control de márgenes de temperatura según se muestra a continuación:

- Si el sensor indica un valor de temperatura superior a 46ºC, se activa la sección de ventilación hasta que baje de 40ºC. (RA2 activado). El sistema sigue funcionando normalmente.
- Si el sensor indica un valor de temperatura superior a 54ºC, se detiene la sección de potencia y por tanto el servomotor hasta que baje de 40ºC. Esto se hace para proteger la unidad de daños críticos. (RA2 y RA5 activados. Este último indica estado de alarma y puede ser utilizado para activar sistemas de aviso). La sección de ventilación y la actualización de los valores de registro sigue activa, así como las funcionalidades del bus $I^2C$.

## *Comandos $I^2C$*

Básicamente las órdenes que llegarán del Master al Slave serán de lectura y escritura en los registros mencionados anteriormente. En la primera transmisión se indica el número de registro al que se quiere acceder, bien sea para

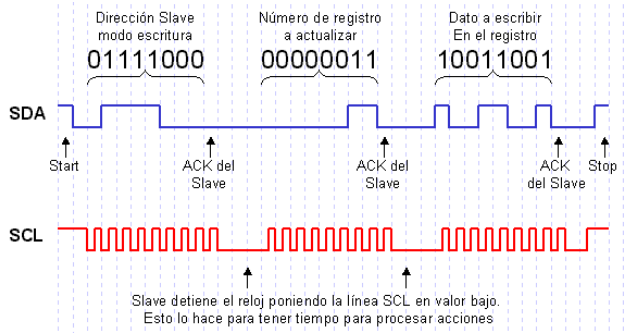
lectura o escritura. Dicha numeración es la siguiente:

1. Registro de Estado.
2. No usado.
3. Posición. En lectura "PosAct" en escritura "PosNew"
4. Temperatura
5. Corriente
6. HalfDeadBand
7. Revisión Firmware.

A continuación se muestra el formato de los comandos $I^2C$ utilizados para el intercambio de datos entre Master y Slave a nivel de lógica digital.

### Escritura registro

El Master envía tres bytes. El primero contiene la dirección del Slave. El segundo, el número de registro a actualizar. El tercero, el dato a escribir en el registro.




### Lectura Registro

Se hacen dos transferencias de dos bytes cada uno. En la primera el Master envía la dirección del Slave y el número de registro a leer. En la segunda transmisión el Master envía la dirección del Slave (pero con bit de lectura) y el Slave envía el contenido del registro escogido.


## *Código fuente*

A continuación se incluye el código fuente del servomotor, que actúa tal como se ha comentado en modo Slave.

```
; SVD01_04
;
; Programa servocontrol (Slave)
; Por: Alejandro Alonso Puig
; Fecha: 10/9/2003
; Controlador: 16F876
; Función:
; Permite control en bucle cerrado de la posición de un motoreductor con
; entrada de feedback por potenciometro. Básicamente es el programa que
; consigue el funcionamiento como servomotor.
; La comunicación con el exterior se hace mediante bus I2C.
; Adicionalmente se mide el consumo de corriente y temperatura
; Tiene medidas de autoprotección con activación de una señal para ventilación
; en caso de superar una temperatura dada y parada de servomotor en caso de
; superar una temperatura límite.
; Shitches dip para especificar la dirección i2c que se utilizará
```

```
        list            p=16F876
        include         "P16F876.INC"


;Definición de macros

        #define Ventilador      PORTA,2 ;Bit de activación de ventilador
        #define AlarmaTemp      PORTA,5 ;Bit de activación de alarma. El módulo se detiene
                                        ;por exceso de temperatura.
        #define Version         d'04'   ;Versión del programa

;Definición de variables

        cblock  0x20

        HalfDeadBand    ;valor de mitad de banda de precisión de posicionamiento
                        ;cuanto menor sea, más precisa es la posición
                        ;DeadBand=2*HalfDeadBand
        PosAct          ;Posición actual del servo según Potenciómetro (0 a 255)
        PosNew          ;Posicion deseada del servo (0+HalfDeadBand+1 a 255-HalfDeadBand-1)
        PosNewL         ;PosNewL y PosNewH son los extremos inferior y superior..
        PosNewH         ;..a PosNew para establecer el margen según "DeadBand"
        PosMax          ;Valores máximo y mínimo que se puede dar a "PosAct"...
        PosMin          ;..en función del HalfDeadBand. Básicamente son..
                        ;(0+HalfDeadBand+1 a 255-HalfDeadBand-1)

        Corriente       ;Consumo de corriente del módulo (ver proc ChkCurrent)
        Temperatura     ;Temperatura del módulo de potencia (ver proc ChkTemperatura)
        Estado          ;Registro especial de estado del módulo
                        ;       Bit0: Ventilador On/Off
                        ;       Bit1: servomotor detenido por exceso de temperatura On/Off
                        ;       Bits2 y 3: Estado posicionamiento servo. Si ambos On, está
                        ;                  posicionado. Si solo uno encendido indica que aún
                        ;                  no está posicionado y el sentido de giro.
                        ;       Bit7: Indica si el servomotor ha sido detenido por orden i2c


        DirNodo         ;Dirección I2C de este slave (4 posibles direcciones segun switches dip)
        MensajeIn       ;Contendrá el dato recibido por I2C del master
        MensajeOut      ;Contendrá el dato a enviar por I2C al master
        BkStatus        ;Backup del registro STATUS
        BkW             ;Backup W
        Temp            ;Variable Temporal usada para evaluación de eventos I2C
        Registro        ;Registro accedido por i2c:
                        ;       1: Registro de Estado. Variable "Estado"
                        ;       2: No usado. Reservado para velocidad en proximas versiones
                        ;       3: Posición. En lectura "PosAct" en escritura "PosNew"
                        ;       4: Temperatura
                        ;       5: Corriente
                        ;       6: HalfDeadBand
                        ;       7: Revisión Firmware. "Version"
        StatI2C         ;Registro intermedio para identificar el estado i2c:
                        ;       0: Si On --> llegó dato Registro (por tanto si llega llega otro
                        ;                    dato, será para sobreescribir contenido registro dado)


        BDel0           ;Usada en retardos
        BDel1           ;Usada en retardos
        BDel2           ;Usada en retardos
        Pausa           ;Usada para hacer pausas con subr "HacerTiempo"

        endc                    ;Fin de definiciones



        org     0
        goto    INICIO
        org     5

;--------------------------------------------------------------------------------
Interrupcion            ;RUTINA DE INTERRUPCIÓN. Se ocupa de los eventos I2C
;--------------------------------------------------------------------------------

        ;Guardamos copia de algunos registros
```

```
        movwf   BkW             ;Hace copia de W
        movf    STATUS,W        ;Hace copia de registro de estado
        banksel PORTA
        movwf   BkStatus


        ;Chequeamos si la interrupción es por evento I2C. En caso positivo llamamos
        ;a la rutina de proceso del evento
        banksel PIR1
        btfss   PIR1,SSPIF      ;Ha ocurrido un evento SSP? (I2C)
        goto    IntNoSSP        ;No. entonces será por otra cosa. Saltamos.
        call    SSP_Handler     ;Si. Procesamos el evento. Si se reciben ordenes, quedarán
                                ;registradas en "MensajeIn". Se enviarán las ordenes
                                ;guardadas en "MensajeOut".
        banksel PIR1
        bcf     PIR1,SSPIF      ;Limpiamos el flag
        goto    Rest

IntNoSSP        ;Aquí se gestionan interrupciones que no son por SSP

;..........
; En caso de necesitarse, poner aquí la rutina de gestión de interrupciones
; que no sean por bus I2C
;..........


Rest    ;Restauramos las copias de los registros
        movf    BkStatus,W      ;Restaura las copias de registros
        movwf   STATUS          ;registro de estado
        movf    BkW,W           ;registro W

        retfie


;--------------------------------------------------------------------------------


INICIO          ;Inicio del cuerpo del programa

        banksel TRISB           ;Apunta a banco 1
        movlw   b'00111100'     ;Salida excepto B2, B3, B4 y B5 que tiene switches dip
        movwf   TRISB           ;

        ;Configuración para uso de conversor A/D
        banksel ADCON1
        movlw   b'00001011'     ;Establece puerta A como salida excepto RA0 ...
        movwf   TRISA           ;..(Potenciómetro), RA1 (medición corriente) y RA3 (Temperatura)
        movlw   b'00000100'     ;PORTA: RA0, RA1, RA3 entrada analog. Resto dig. Justif ADRESH
        movwf   ADCON1          ;

        ;Inicialización variable de posición del servomotor
        banksel PosAct
        call    ChkPosAct       ;Identificación posición actual del servo..
        movf    PosAct,W        ;..y la guardamos en PosNew..
        movwf   PosNew          ;..para que el servo no se mueva...

        ;Establecemos valor de DeadBand por defecto. Se puede modificar por i2c
        movlw   d'3'
        movwf   HalfDeadBand

        ;Establecemos valor registro especial de estado por defecto
        movlw   b'10000000'
        movwf   Estado

        ;Configuración para gestión i2c
        call    init_i2c_Slave ;Configuración para uso de i2c
        banksel INTCON
        bsf     INTCON,GIE      ;Activamos las interrupciones

        banksel PORTA
        clrf    PORTA
        clrf    PORTB
        clrf    StatI2C
```

```
BUCLE   ;Bucle principal del programa

StSt    ;Chequeo bit de start/stop del servomotor en registro especial de estado
        btfsc   Estado,7        ;On:servomotor funciona, Off:servomotor se detiene
        goto    StON            ;Está On. Seguimos
        call    PararMotor      ;Está Off. Paramos servomotor,...
        call    ChkPosAct       ;..pero seguimos midiendo variables...
        call    ChkCurrent      ;..por si son solicitadas por i2c
        call    ChkTemperatura ;Además el ventilador seguira activo si es necesario
        goto    StSt            ;y nos mantenemos en bucle hasta que se active por i2c

StON

        ;Chequeo Temperatura
        call    ChkTemperatura

        ;Chequeo de consumo eléctrico
        call    ChkCurrent

        ;Verificamos valor de PosNew para que no sobrepase los límites:
        ;(0+HalfDeadBand+1 a 255-HalfDeadBand-1)
        ;Si PosNew > 255-HalfDeadBand-1 then PosNew=255-HalfDeadBand-1
        ;Si PosNew < HalfDeadBand+1 then PosNew=HalfDeadBand+1
        movf    HalfDeadBand,W
        addlw   d'1'
        movwf   PosMin          ;0+HalfDeadBand+1
        movlw   d'254'
        movwf   PosMax
        movf    HalfDeadBand,W
        subwf   PosMax,F        ;255-HalfDeadBand-1

        ;Checkea si PosNew > PosMax
        movf    PosMax,W        ;
        subwf   PosNew,W        ;Resta/compara con "PosNew"
        btfsc   STATUS,Z        ;Son iguales (Z=1)??
        goto    OkSigue         ;Si. Está en el límite de posición, pero es correcto
        btfsc   STATUS,C        ;No. Mayor (C=0)??
        goto    FuerzaMax       ;Si. (PosNew > PosMax) Forzamos que no pase del límite
        goto    ChkMenos        ;No, (PosNew < PosMax)

ChkMenos        ;Checkea si PosNew < PosMin
        movf    PosMin,W        ;
        subwf   PosNew,W        ;Resta/compara con "PosNew"
        btfsc   STATUS,Z        ;Son iguales (Z=1)??
        goto    OkSigue         ;Si. Está en el límite de posición, pero es correcto
        btfsc   STATUS,C        ;No. Mayor (C=0)??
        goto    OkSigue         ;Si. (PosNew > PosMin) está ok: PosMin<PosNew<PosMax
        goto    FuerzaMin       ;No, (PosNew < PosMin) Forzamos que no pase del límite

FuerzaMax       ;Fuerza a que la posición no pase del máximo
        movf    PosMax,w
        movwf   PosNew
        goto    OkSigue

FuerzaMin       ;Fuerza a que la posición no pase del mínimo
        movf    PosMin,W
        movwf   PosNew
        goto    OkSigue

OkSigue

        ;Cálculo PosNewH y PosNewL en función del valor de PosNew
        movf    PosNew,W
        movwf   PosNewH
        movwf   PosNewL
        movf    HalfDeadBand,W
        addwf   PosNewH,F
        subwf   PosNewL,F

        ;Identificación posición actual del servo
        call    ChkPosAct


ChkMayor        ;Checkea si PosAct > PosNewH --> Giro antihorario
        movf    PosNewH,W       ;
        subwf   PosAct,W        ;Resta/compara con "PosAct"
```

```
        btfsc   STATUS,Z        ;Son iguales (Z=1)??
        goto    OkPosic         ;Si. Está en el límite de DeadBand pero se da por bueno
        btfsc   STATUS,C        ;No. Mayor (C=0)??
        goto    GiroAntihora    ;Si. (PosAct > PosNewH) Activamos motor
        goto    ChkMenor        ;No, (PosAct < PosNewH)

ChkMenor        ;Checkea si PosAct < PosNewL --> Giro antihorario
        movf    PosNewL,W       ;
        subwf   PosAct,W        ;Resta/compara con "PosAct"
        btfsc   STATUS,Z        ;Son iguales (Z=1)??
        goto    OkPosic         ;Si. Está en el límite de DeadBand pero se da por bueno
        btfsc   STATUS,C        ;No. Mayor (C=0)??
        goto    OkPosic         ;Si. (PosAct > PosNewL) está en Deadband:PosNewL< PosAct < PosNewH
        goto    GiroHora        ;No, (PosAct < PosNewL) Activamos motor

OkPosic         ;Paramos motor
        call    PararMotor
        goto    BUCLE

GiroHora        ;Activamos motor sentido Horario y actualizamos valor registro especial Estado
        bsf     PORTB,0
        bcf     PORTB,1
        bsf     Estado,2
        bcf     Estado,3
        goto    BUCLE

GiroAntihora    ;Activamos motor sentido Antihorario y actualizamos valor registro especial Estado
        bcf     PORTB,0
        bsf     PORTB,1
        bcf     Estado,2
        bsf     Estado,3
        goto    BUCLE




        goto    BUCLE




;*******************************************************************************
; SUBRUTINAS
;*******************************************************************************


;-------------------------------------------------------------------------------
ChkPosAct               ;Identifica la posición actual del servo mediante la
                        ;la conversión A/D del valor dado por el potenciómetro
                        ;y lo deja en la variable "PosAct"
;-------------------------------------------------------------------------------

        ;Tomaremos datos de ADRESH
        banksel ADCON1
        bcf     ADCON1,ADFM

        banksel ADCON0
        movlw   b'11000001'     ;PORTA-RA0: osci interno, canal-0, activar captura
        movwf   ADCON0          ;
        movlw   d'1'            ;Pausa para que de tiempo al condensador interno...
        movwf   Pausa           ;..a capturar el valor analógico
        call    HacerTiempo
        bsf     ADCON0,GO       ;Comenzar conversión A/D
AD_W1   btfsc   ADCON0,GO_DONE  ;Conversión finalizada?
        goto    AD_W1
        movf    ADRESH,W
        movwf   PosAct          ;Guarda en variable de posición actual
        return


;-------------------------------------------------------------------------------
ChkCurrent              ;Obtiene el consumo de corriente del módulo servomotor
                        ;mediante conversión A/D de la caida de potencial de una
                        ;resistencia de 1 ohm. El valor obtenido se almacena en la
                        ;variable "Corriente" de manera que el consumo en amperios
```

```
                        ;se puede calcular con la fórmula siguiente:
                        ;         Amperios=(Corriente x 0.02)/R
                        ;siendo R la resistencia de medición, que en principio es de
                        ;1ohm, pero que conviene medir para que los calculos sean exactos
                        ;La máxima corriente que puede medir es 255x0.02=5.1 amperios
                        ;Si la corriente va a ser superior, habra que reducir el valor
                        ;de la resistencia de manera que Max Amperios x R < 5
;-------------------------------------------------------------------------------


        ;Tomaremos datos de ADRESH
        banksel ADCON1
        bcf     ADCON1,ADFM

        banksel ADCON0
        movlw   b'11001001'     ;PORTA-RA1: osci interno, canal-1, activar conversion
        movwf   ADCON0          ;
        movlw   d'1'            ;Pausa para que de tiempo al condensador interno...
        movwf   Pausa           ;..a capturar el valor analógico
        call    HacerTiempo
        bsf     ADCON0,GO       ;Comenzar conversión A/D
AD_W2   btfsc   ADCON0,GO_DONE ;Conversión finalizada?
        goto    AD_W2
        movf    ADRESH,W
        movwf   Corriente       ;Guarda en variable de medición de corriente
        return



;-------------------------------------------------------------------------------
ChkTemperatura          ;Obtiene la temperatura del módulo de potencia del servomotor
                        ;mediante conversión A/D de la salida del sensor LM35
                        ;y la almacena en la variable "Temperatura"
                        ;La temperatura en °C del módulo será igual a:
                        ;       °C=(Temperatura/2)+2
;-------------------------------------------------------------------------------


        ;Tomaremos datos de ADRESL para tener una precisión de medio grado por bit
        banksel ADCON1
        bsf     ADCON1,ADFM

        banksel ADCON0
        movlw   b'11011001'     ;PORTA-RA3: osci interno, canal-3, activar conversion
        movwf   ADCON0          ;
        movlw   d'1'            ;Pausa para que de tiempo al condensador interno...
        movwf   Pausa           ;..a capturar el valor analógico
        call    HacerTiempo
        bsf     ADCON0,GO       ;Comenzar conversión A/D
AD_W3   btfsc   ADCON0,GO_DONE ;Conversión finalizada?
        goto    AD_W3
        banksel ADRESL
        movf    ADRESL,W
        banksel Temperatura
        movwf   Temperatura     ;Guarda en variable de medición de temperatura

        ;Los drivers utilizados tienen un rango de funcionamiento entre 0° y 70°C
        ;Checkeamos si ha superado 46°C, con lo que activaríamos ventilador hasta
        ;que baje de 40°C
        movlw   d'88'           ;46°C=factor 88  --->(88/2)+2=46
        subwf   Temperatura,W   ;Resta/compara con "Temperatura"
        btfsc   STATUS,Z        ;Son iguales (Z=1)??
        goto    VentiladorOn    ;Si. Activamos ventilador
        btfsc   STATUS,C        ;No. Mayor (C=0)??
        goto    VentiladorOn    ;Si. Límite de temperatura superado
                                ;No, verificamos si menor que 40°
        movlw   d'76'           ;40°C=factor 76  --->(76/2)+2=40
        subwf   Temperatura,W   ;Resta/compara con "Temperatura"
        btfsc   STATUS,Z        ;Son iguales (Z=1)??
        goto    FinChk          ;Si. no hacemos nada. Si ventilador activado, continuará así
        btfsc   STATUS,C        ;No. Mayor (C=0)??
        goto    FinChk          ;Si. no hacemos nada. Si ventilador activado, continuará así
        goto    VentiladorOff   ;No, ha bajado de 40°, podemos desactivar ventilador


VentiladorOn
```

```
        bsf     Ventilador      ;Activamos ventilador
        bsf     Estado,0        ;..y registro especial de estado

        ;Checkeamos si ha superado 54°C, con lo que detendríamos el módulo
        ;para evitar sobrecalentamiento hasta que baje de 40°C
        movlw   d'104'          ;54°C=factor 104  --->(104/2)+2=54
        subwf   Temperatura,W   ;Resta/compara con "Temperatura"
        btfsc   STATUS,Z        ;Son iguales (Z=1)??
        goto    AlarmOn         ;Si. Activamos procedimiento de alarma por sobrecalentamiento
        btfsc   STATUS,C        ;No. Mayor (C=0)??
        goto    AlarmOn         ;Si. Límite de temperatura superado
                                ;No, todo en orden
        goto    FinChk


VentiladorOff
        bcf     Ventilador      ;Paramos ventilador
        bcf     Estado,0        ;..y registro especial de estado
        goto    FinChk


AlarmOn ;Procedimiento ante sobrecalentamiento del módulo

        ;Paramos motor
        call    PararMotor

        ;Activamos bit de alarma por sobrecalentamiento
        bsf     AlarmaTemp
        bsf     Estado,1        ;..y registro especial de estado

ChkT    ;Entramos en bucle hasta que baje de 40°C
        banksel ADCON0
        movlw   b'11011001'     ;PORTA-RA3: osci interno, canal-3, activar conversion
        movwf   ADCON0          ;
        movlw   d'1'            ;Pausa para que de tiempo al condensador interno...
        movwf   Pausa           ;..a capturar el valor analógico
        call    HacerTiempo
        bsf     ADCON0,GO       ;Comenzar conversión A/D
AD_W4   btfsc   ADCON0,GO_DONE ;Conversión finalizada?
        goto    AD_W4
        banksel ADRESL
        movf    ADRESL,W
        banksel Temperatura
        movwf   Temperatura     ;Guarda en variable de medición de temperatura

        movlw   d'76'           ;40°C=factor 76  --->(76/2)+2=40
        subwf   Temperatura,W   ;Resta/compara con "Temperatura"
        btfsc   STATUS,Z        ;Son iguales (Z=1)??
        goto    ChkT            ;Si. continuamos en bucle hasta que se enfrie
        btfsc   STATUS,C        ;No. Mayor (C=0)??
        goto    ChkT            ;Si. continuamos en bucle hasta que se enfrie
        bcf     AlarmaTemp      ;No, ha bajado de 40°, podemos desactivar condicion alarma
        bcf     Ventilador      ;Paramos ventilador
        bcf     Estado,0        ;..y registro especial de estado
        goto    FinChk


FinChk
        return



;-------------------------------------------------------------------------------
PararMotor              ;Detiene el servomotor
;-------------------------------------------------------------------------------
        bsf     PORTB,0
        bsf     PORTB,1
        bsf     Estado,2
        bsf     Estado,3
        return



;-------------------------------------------------------------------------------
init_i2c_Slave          ;Inicializa valores para uso de I2C en Slave
                        ;Ha de ser llamado tras definir TRISC (de ser necesario)
```

```
;--------------------------------------------------------------------------------

        ;Guardamos copia de algunos registros
        movwf   BkW             ;Hace copia de W
        movf    STATUS,W        ;Hace copia de registro de estado
        banksel PORTA
        movwf   BkStatus

        ;Establecemos dirección del esclavo segun switches dip (B2 y B3)
        movlw   b'01111000'
        movwf   DirNodo
        btfsc   PORTB,2
        bsf     DirNodo,2
        btfsc   PORTB,3
        bsf     DirNodo,1

        ;Configuramos I2C
        banksel TRISC           ; Pasamos a direccionar Banco 1
        movlw   b'00011000'     ; Establece líneas SDA y SCL como entradas...
        iorwf   TRISC,F         ;..respetando los valores para otras líneas.
        bcf     SSPSTAT,CKE     ; Establece I2C input levels
        bcf     SSPSTAT,SMP     ; Habilita slew rate
        bsf     SSPCON2,GCEN    ; Habilita direccionamiento global
        banksel DirNodo
        movf    DirNodo,W       ; Dirección esclavo
        banksel SSPADD
        movwf   SSPADD          ;
        banksel SSPCON                ; Pasamos a direccionar Banco 0
        movlw   b'00110110'     ; Slave mode, SSP enable,
        movwf   SSPCON          ;
        bcf     PIR1,SSPIF      ; Limpia flag de eventos SSP
        bcf     PIR1,7          ; Limpia bit. Mandatorio por Datasheet

        ;Configuración para interrupciones por evento I2C
        banksel PIE1
        bsf     PIE1,SSPIE
        bsf     INTCON,PEIE

        ;Restauramos las copias de los registros
        movf    BkStatus,W      ;Restaura las copias de registros
        movwf   STATUS          ;registro de estado
        movf    BkW,W           ;registro W

        return


; ------------------------------------------------------------------------------------------
SSP_Handler     ; Este manejador controla cada evento SSP (I2C) acontecido.
                ; El código que se muestra abajo chequea 5 posibles estados.
                ; Cada uno de los 5 estados SSP son identificados haciendo
                ; XOR de los bits del registro SSPSTAT con mascaras de bits
                ; predeterminadas. Una vez que el estado ha sido identificado
                ; se llevan a cabo las acciones pertinentes. Los estados
                ; indefinidos son considerados como estados de error.

                ; State 1: Operación de escritura I2C, ultimo byte era de dirección.
                ; SSPSTAT bits: S = 1, D_A = 0, R_W = 0, BF = 1

                ; State 2: Operación de escritura I2C, ultimo byte era de datos.
                ; SSPSTAT bits: S = 1, D_A = 1, R_W = 0, BF = 1

                ; State 3: Operación de lectura I2C, ultimo byte era de dirección.
                ; SSPSTAT bits: S = 1, D_A = 0, R_W = 1, BF = 0

                ; State 4: Operación de lectura I2C, ultimo byte era de datos.
                ; SSPSTAT bits: S = 1, D_A = 1, R_W = 1, BF = 0

                ; State 5: Reset lógico del Slave I2C por NACK del master.
                ; SSPSTAT bits: S = 1, D_A = 1, R_W = 0, BF = 0

; ------------------------------------------------------------------------------------------


        banksel SSPSTAT
        movf    SSPSTAT,W       ; Obtiene el valor de SSPSTAT
        andlw   b'00101101'     ; elimina los bits no importantes SSPSTAT.
```

```
        banksel Temp
        movwf   Temp            ; para chequeo posterior.

State1:                         ; Operación de escritura, ultimo byte ha sido
        movlw   b'00001001'     ; de dirección, el buffer está lleno.
        banksel Temp
        xorwf   Temp,W          ;
        btfss   STATUS,Z        ; Estamos en el primer estado?
        goto    State2          ; No, checkeamos siguiente estado
        call    ReadI2C         ; SI. Hacemos un read SSPBUF (para vaciar buffer).
                                ; El Hardware se ocupa de mandar Ack
        return

State2:                         ; Operación de escritura, ultimo byte ha sido
        movlw   b'00101001'     ; de datos, el buffer está lleno.
        banksel Temp
        xorwf   Temp,W
        btfss   STATUS,Z        ; Estamos en el segundo estado?
        goto    State3          ; NO, checkeamos siguiente estado
        call    ReadI2C         ; SI, Tomamos el byte del SSP.

        ;Aquí tenemos en W el valor del dato recibido
        movwf   MensajeIn

        btfsc   StatI2C,0       ;Chequeamos si ya sabíamos el registro a actualizar y es un dato
        goto    LoSabemos       ;Lo sabíamos, es un dato
        movf    MensajeIn,W     ;No lo sabíamos, nos acaba de llegar
        movwf   Registro
        bsf     StatI2C,0
        return

LoSabemos       ;Sabemos el registro y tenemos el dato, actualizamos dato del registro
        call    UpdateReg
        bcf     StatI2C,0       ;Limpiamos flag
        return

State3:                         ; Operación de lectura, ultimo byte ha sido
        movlw   b'00001100'     ; de dirección, el buffer está vacío
        banksel Temp
        xorwf   Temp,W
        btfss   STATUS,Z        ; Estamos en el tercer estado?
        goto    State4          ; NO, checkeamos siguiente estado
                                ; SI

        btfsc   StatI2C,0       ;Chequeamos si ya sabemos el registro a leer
        goto    LoSabemosR      ;Lo sabemos,
        movlw   0               ;No lo sabemos, devolvemos un cero por defecto
        call    WriteI2C        ;escribimos el byte en SSPBUF
        return
LoSabemosR      ;Sabemos el registro a leer. Lo leemos y enviamos el dato
        call    ReadReg
        movf    MensajeOut,W
        call    WriteI2C        ; SI, escribimos el byte en SSPBUF
        bcf     StatI2C,0       ; Limpiamos flag de número de registro recibido
        return

State4:                         ; Operación de lectura, ultimo byte ha sido
        movlw   b'00101100'     ; de datos, el buffer está vacío
        banksel Temp
        xorwf   Temp,W
        btfss   STATUS,Z        ; Estamos en el cuarto estado?
        goto    State5          ; NO, checkeamos siguiente estado
                                ; SI. Operación no admitida.
        movlw   0               ; devolvemos un cero por defecto
        call    WriteI2C        ; escribimos el byte en SSPBUF
        return

State5:
        movlw   b'00101000'     ; Se ha recibido un NACK mientras se transmitían...
        banksel Temp
        xorwf   Temp,W          ; ..datos al master. Lo lógica del Slave..
        btfss   STATUS,Z        ; ..se resetea en este caso. R_W = 0, D_A = 1
        goto    I2CErr          ; y BF = 0
        return                  ; Si no estamos en State5, entonces es
                                ; que algo fue mal
```

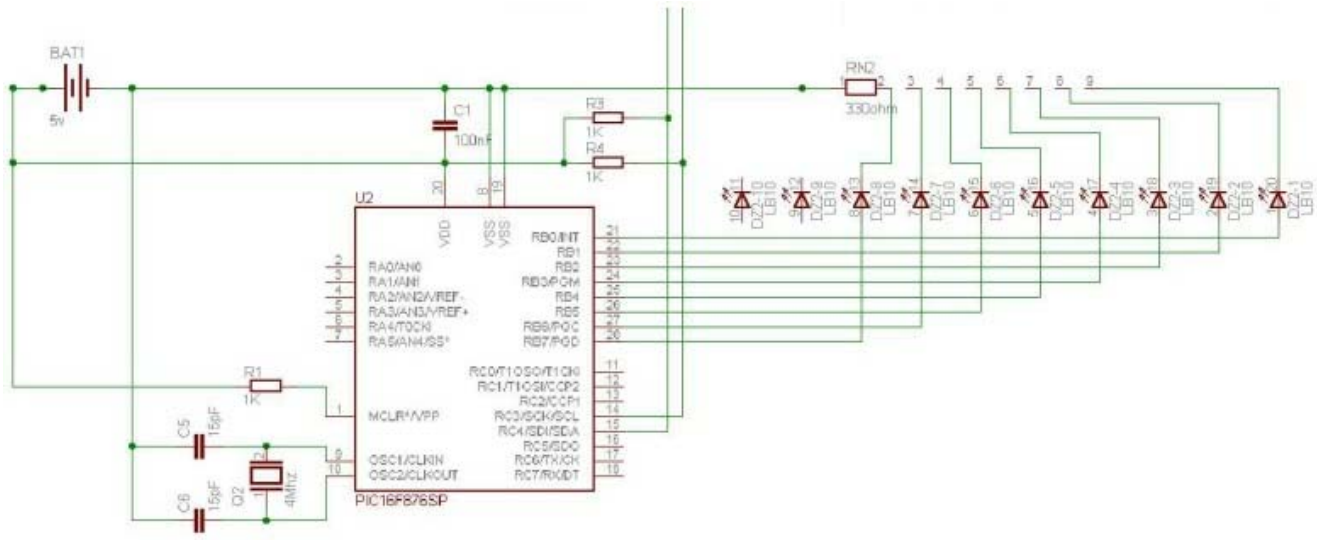
```
I2CErr nop                  ; Algo fue mal
       return


;--------------------------------------------------------------------
WriteI2C       ;Usada por SSP_Handler para escribir datos en bus I2C
;--------------------------------------------------------------------

       banksel SSPCON
       movwf   SSPBUF        ; Escribe el dato en W
       bsf     SSPCON,CKP    ; Libera el reloj
       return

;--------------------------------------------------------------------
ReadI2C        ;Usada por SSP_Handler para escribir datos en bus I2C
;--------------------------------------------------------------------

       banksel SSPBUF
       movf    SSPBUF,W      ; Toma el byte y lo guarda en W
       return


;--------------------------------------------------------------------
UpdateReg             ;Actualiza Registro ordenado por I2C
;--------------------------------------------------------------------

       ;Procedemos a actuar según la orden recibida del Master. Haremos un Pseudo CASE
       ;que actualice solo los registros escribibles e ignore los que no se puedan escribir


M_01   ;Estado
       movlw   d'1'          ;
       xorwf   Registro,W    ;
       btfss   STATUS,Z      ; Es este el registro a actualizar?
       goto    M_03          ; No, chequeamos siguiente caso
                             ; Si. procedemos a actualizar el registro (solo bits R/W)
       bsf     Estado,7      ; Flag 7: On/Off servomotor
       btfss   MensajeIn,7   ; Simplemente establecemos el mismo valor de bit...
       bcf     Estado,7      ; ...que nos ha llegado a "MensajeIn"
       return                ;Regresamos a la espera de una nueva orden

M_03   ;Posición. En Escritura es PosNew
       movlw   d'3'          ;
       xorwf   Registro,W    ;
       btfss   STATUS,Z      ; Es este el registro a actualizar?
       goto    M_06          ; No, chequeamos siguiente caso
       movf    MensajeIn,W   ; Si. procedemos a actualizar el registro
       movwf   PosNew
       return                ;Regresamos a la espera de una nueva orden

M_06   ;HalfDeadBand
       movlw   d'6'          ;
       xorwf   Registro,W    ;
       btfss   STATUS,Z      ; Es este el registro a actualizar?
       goto    M_Error       ; No, chequeamos siguiente caso
       movf    MensajeIn,W   ; Si. procedemos a actualizar el registro
       movwf   HalfDeadBand
       return                ;Regresamos a la espera de una nueva orden


M_Error;No es un registro conocido o es de solo lectura. Se ignora.

       return



;--------------------------------------------------------------------
ReadReg        ;Leemos Registro solicitado por I2C
;--------------------------------------------------------------------

       ;Haremos un Pseudo CASE que lea el registro solicitado y lo deje en MensajeOut


M_01R  ;Estado
       movlw   d'1'          ;
```

```
       xorwf   Registro,W     ;
       btfss   STATUS,Z       ; Es este el registro a leer?
       goto    M_03R          ; No, chequeamos siguiente caso
       movf    Estado,W       ; Si. procedemos a leer el registro
       movwf   MensajeOut
       return                 ;Regresamos a la espera de una nueva orden

M_03R  ;Posición. En Lectura es PosAct
       movlw   d'3'           ;
       xorwf   Registro,W     ;
       btfss   STATUS,Z       ; Es este el registro a leer?
       goto    M_04R          ; No, chequeamos siguiente caso
       movf    PosAct,W       ; Si. procedemos a leer el registro
       movwf   MensajeOut
       return                 ;Regresamos a la espera de una nueva orden

M_04R  ;Temperatura
       movlw   d'4'           ;
       xorwf   Registro,W     ;
       btfss   STATUS,Z       ; Es este el registro a leer?
       goto    M_05R          ; No, chequeamos siguiente caso
       movf    Temperatura,W  ; Si. procedemos a leer el registro
       movwf   MensajeOut
       return                 ;Regresamos a la espera de una nueva orden

M_05R  ;Corriente
       movlw   d'5'           ;
       xorwf   Registro,W     ;
       btfss   STATUS,Z       ; Es este el registro a leer?
       goto    M_06R          ; No, chequeamos siguiente caso
       movf    Corriente,W    ; Si. procedemos a leer el registro
       movwf   MensajeOut
       return                 ;Regresamos a la espera de una nueva orden

M_06R  ;HalfDeadBand
       movlw   d'6'           ;
       xorwf   Registro,W     ;
       btfss   STATUS,Z       ; Es este el registro a leer?
       goto    M_07R          ; No, chequeamos siguiente caso
       movf    HalfDeadBand,W        ; Si. procedemos a leer el registro
       movwf   MensajeOut
       return                 ;Regresamos a la espera de una nueva orden

M_07R  ;Version
       movlw   d'7'           ;
       xorwf   Registro,W     ;
       btfss   STATUS,Z       ; Es este el registro a leer?
       goto    M_ErrorR       ; No, chequeamos siguiente caso
       movlw   Version        ; Si. procedemos a leer el registro (en este caso una constante)
       movwf   MensajeOut
       return                 ;Regresamos a la espera de una nueva orden

M_ErrorR       ;No es un registro conocido. Se devuelve valor 0
       clrf    MensajeOut
       return



;-------------------------------------------------------------------------------------
HacerTiempo    ;realiza una pausa del numero de centesimas de segundo especificadas en "Pausa"
               ;El tiempo real es aproximado, dependiendo del número de interrupciones
               ;que se producan.
;-------------------------------------------------------------------------------------

       movf    Pausa,W        ;Coloca el valor de pausa en BDel2...
       movwf   BDel2          ;...para no alterar su contenido

;.........................................................
; Generado con PDEL ver SP  r 1.0  el 24/02/03 Hs 18:31:22
; Descripcion: Delay 10000 ciclos (1 centésima de segundo)
;.........................................................
BCiclo  movlw     .8        ; 1 set numero de repeticion  (B)
        movwf     BDel0     ; 1 |
BLoop1  movlw     .249      ; 1 set numero de repeticion  (A)
        movwf     BDel1     ; 1 |
BLoop2  nop                 ; 1 nop
```

```
        nop                 ; 1 ciclo delay
        decfsz    BDel1, 1  ; 1 + (1) es el tiempo 0  ? (A)
        goto      BLoop2    ; 2 no, loop
        decfsz    BDel0,  1 ; 1 + (1) es el tiempo 0  ? (B)
        goto      BLoop1    ; 2 no, loop
BDelL1  goto BDelL2         ; 2 ciclos delay
BDelL2  nop                 ; 1 ciclo delay
;.....................................................
       decfsz  BDel2,F       ;Repite tantas veces el ciclo de una decima de segundo...
       goto    BCiclo        ;..como se lo indique BDel2
        return              ; 2+2 Fin.


Fin
       END
```

# Ejemplo de Master

El siguiente sería un ejemplo de montaje electrónico para el Master que controlaría dicho servomotor.




A continuación se incluye un ejemplo de código para el Master, que hace que el servo se mueva de un extremo a otro con una pausa de unos 2 segundos entre cambios de posición. Asimismo solicita al servomotor (Slave) el dato de consumo de corriente y lo muestra por los leds conectados al puerto B. El código para controlar el Servo ha de incluirse en el cuerpo del programa (donde el bucle MLoop).

```
; MSVD04
; Master para control Servo SVD01_04
; Por: Alejandro Alonso Puig
; Fecha: 10/9/2003
; Controlador: 16F876
; Función:
; Controla por I2C el módulo servomotor SVD01 v4
; Como ejemplo, mueve el servomotor de un lado al otro y va mostrando el valor del
; registro de consumo de corriente por barrera de leds conectada
; al puerto B
;
; Las subrutinas definidas para el intercambio de datos con el módulo SVD01 (Servomotor)
; son los siguientes:
;
```

```
; LECTURA DE REGISTROS
; --------------------
;
; ReadPosSVD01        Obtiene por i2c la posición del módulo de servomotor
;                     y la deja en la variable "PosActSVD01"
; ReadTempSVD01              Lee el valor de temperatura del módulo SVD01 y la deja
;                     en "TempSVD01"
; ReadCurrSVD01              Lee el valor de consumo del módulo SVD01
;                     y lo deja en la variables "CurrSVD01"
; ReadStatSVD01              Lee el registro de estado del módulo SVD01 y lo deja
;                     en "StatSVD01"
; ReadHDBSVD01        Obtiene por i2c el registro HalfDeadBand del módulo de
;                     servomotor y la deja en la variable "HDBSVD01"
; ReadVerSVD01        Obtiene por i2c la versión de firmware del módulo de
;                     servomotor y la deja en la variable "VerSVD01"
;
; ESCRITURA DE REGISTROS
; ----------------------
;
; SetPosSVD01         Ordena por i2c al módulo de servomotor que se
;                     coloque en la posición establecida por "PosNewSVD01"
; SetStatSVD01        Ordena por i2c al módulo de servomotor la actualización
;                     de su registro especial de estado según el valor de
;                     "StatSVD01". Ha de tenerse en cuenta que solo se actualizarán
;                     aquellos bits que sean R/W. Los demás serán ignorados.
; SetHDBSVD01         Ordena por i2c al módulo de servomotor que almacene
;                     en su registro HalfDeadBand el valor dado por "HDBSVD01"



        list            p=16F876
        include         "P16F876.INC"


;Definición de constantes

        #define ClockValue      d'9'    ;(100khz) valor para cálculo de vel. I2C que pasará a SSPADD
        #define DirSVD01        b'01111000'     ;Dirección Módulo servomotor (01111000)

        ;Códigos de registro del módulo SVD01
        #define SVD01Stat       d'1'    ;Registro de estado
        #define SVD01Posic      d'3'    ;Posición
        #define SVD01Temp       d'4'    ;Temperatura
        #define SVD01Curr       d'5'    ;Corriente
        #define SVD01HDB        d'6'    ;HalfDeadBand
        #define SVD01Ver        d'7'    ;Versión del firmware


;Definición de variables

        cblock  0x20
        MensajeIn       ;Contendrá el dato recibido por I2C del slave
        MensajeOut      ;Contendrá el dato a enviar por I2C al slave
        DirSlave        ;Dirección del Slave
        BkStatus        ;Backup del registro STATUS
        BkW             ;Backup W

        BDel0           ;Usada en retardos
        BDel1           ;Usada en retardos
        BDel2           ;Usada en retardos
        Pausa           ;Usada en para hacer pausas con subr "HacerTiempo"


        ;Registros de módulo SVD01
        PosActSVD01     ;Variable de posición actual de servomotor según dato recibido por i2c
        PosNewSVD01     ;Posición que se desea que tenga la servomotor (2 a 253)
        StatSVD01       ;Registro de estado
        TempSVD01       ;Temperatura
        CurrSVD01       ;Corriente
        HDBSVD01        ;HalfDeadBand
        VerSVD01        ;Versión del firmware


        endc            ;Fin de definiciones
```

```
        org     0
        goto    INICIO
        org     5


;-------------------------------------------------------------------------------



INICIO          ;Inicio del cuerpo del programa

        banksel TRISA           ;Apunta a banco 1
        movlw   b'00011111'     ;Entrada (switches). Solo 5 inferiores
        movwf   TRISA           ;
        movlw   b'00000000'     ;Salida (Leds)
        movwf   TRISB           ;
        banksel PORTB           ;Apunta a banco 0
        clrf    PORTB           ;Limpia puerto B
        clrf    PORTA           ;Limpia puerto A

        call    init_i2c_Master       ;Configuración para uso de i2c

        clrf    MensajeIn
        clrf    MensajeOut

        movlw   d'10'           ;Pausa de 10 centésimas de segundo para que en...
        movwf   Pausa           ;...el arranque de tiempo a los slaves a quedar...
        call    HacerTiempo     ;..configurados adecuadamente.


;*******************************************************************************
;A partir de aquí ha de incluirse el código con el que controlar el servomotor
;El control sobre el servomotor se realizará mediante llamadas a las diferentes
;rutinas de lectura y escritura de registros comentadas al principio del programa.
;*******************************************************************************

MLoop

        ;Proceso para resetear el slave en caso de bloqueo de algún tipo
        call Send_RStart
        call Send_Stop
        banksel PORTB

        ;Ordena movimiento a posición concreta
        movlw   d'255'
        movwf   PosNewSVD01
        call    SetPosSVD01

        ;pausa de 50 centésimas de segundo
        movlw   d'50'
        movwf   Pausa
        call    HacerTiempo

        ;Lee registro de corriente del módulo servomotor
        call    ReadCurrSVD01
        movf    CurrSVD01,W
        movwf   PORTB           ;muestra en leds de portb

        ;pausa de 200 centésimas de segundo (2 segundos)
        movlw   d'200'
        movwf   Pausa
        call    HacerTiempo

        ;Ordena movimiento a posición concreta
        movlw   d'0'
        movwf   PosNewSVD01
        call    SetPosSVD01

        ;pausa de 200 centésimas de segundo (2 segundos)
        movlw   d'200'
        movwf   Pausa
        call    HacerTiempo
```

```
        goto    MLoop




;*******************************************************************************
; SUBRUTINAS
;*******************************************************************************



;-------------------------------------------------------------------------------
ReadPosSVD01            ;Obtiene por i2c la posición del módulo de servomotor
                        ;y la deja en la variable "PosActSVD01"
;-------------------------------------------------------------------------------

        banksel DirSlave
        movlw   DirSVD01
        movwf   DirSlave
        movlw   SVD01Posic      ;Registro 3 de SVD01
        movwf   MensajeOut
        call    Send_Start      ;Envía condición de inicio
        banksel DirSlave
        call    Enviar          ;Envía el dato al Slave
        call    Send_RStart     ;Envía condición de reinicio
        banksel DirSlave
        call    Recibir         ;Toma dato del Slave...
        movf    MensajeIn,W     ;...y lo guarda en...
        movwf   PosActSVD01     ;...la variable de registro correspondiente
        call    Send_Stop       ;Envía condición de stop
        banksel DirSlave

        return


;-------------------------------------------------------------------------------
SetPosSVD01             ;Ordena por i2c al módulo de servomotor que se
                        ;coloque en la posición establecida por "PosNewSVD01"
;-------------------------------------------------------------------------------

        banksel DirSlave
        movlw   DirSVD01
        movwf   DirSlave
        movlw   SVD01Posic      ;Registro 3 de SVD01
        movwf   MensajeOut
        call    Send_Start      ;Envía condición de inicio
        banksel DirSlave
        call    Enviar          ;Envía el dato al Slave
        movf    PosNewSVD01,W   ;Se deja en W para que...
        call    Send_Byte       ;...Send_Byte lo envíe por i2c
        call    Send_Stop       ;Envía condición de stop
        banksel DirSlave
        return



;-------------------------------------------------------------------------------
ReadTempSVD01           ;Lee el valor de temperatura del módulo SVD01 y la deja
                        ;en TempSVD01
;-------------------------------------------------------------------------------

        ;Leemos temperatura en SVD01
        banksel DirSlave
        movlw   DirSVD01
        movwf   DirSlave
        movlw   SVD01Temp       ;Registro 4 de SVD01 (Registro de temperatura)
        movwf   MensajeOut
        call    Send_Start      ;Envía condición de inicio
        banksel DirSlave
        call    Enviar          ;Envía el dato al Slave
        call    Send_RStart     ;Envía condición de reinicio
        banksel DirSlave
        call    Recibir         ;Toma dato del Slave...
        movf    MensajeIn,W     ;...y lo guarda en...
```

```
        movwf   TempSVD01       ;...la variable correspondiente
        call    Send_Stop       ;Envía condición de stop
        banksel DirSlave

        return



;*******************************************************************************
ReadCurrSVD01  ;Lee el valor de consumo del módulo SVD01
                        ;y lo deja en la variables CurrSVD01
;*******************************************************************************

        banksel DirSlave
        movlw   DirSVD01
        movwf   DirSlave
        movlw   SVD01Curr       ;Registro 5 de SVD01 (Registro de Corriente)
        movwf   MensajeOut
        call    Send_Start      ;Envía condición de inicio
        banksel DirSlave
        call    Enviar          ;Envía el dato al Slave
        call    Send_RStart     ;Envía condición de reinicio
        banksel DirSlave
        call    Recibir         ;Toma dato del Slave...
        movf    MensajeIn,W     ;...y lo guarda en...
        movwf   CurrSVD01       ;...la variable correspondiente
        call    Send_Stop       ;Envía condición de stop
        banksel DirSlave

        return



;*******************************************************************************
ReadStatSVD01           ;Lee el registro de estado del módulo SVD01 y lo deja
                        ;en StatSVD01
;*******************************************************************************

        ;Leemos registro de estado en MD03_1
        banksel DirSlave
        movlw   DirSVD01
        movwf   DirSlave
        movlw   SVD01Stat       ;Registro 1 de SVD01 (registro de estado)
        movwf   MensajeOut
        call    Send_Start      ;Envía condición de inicio
        banksel DirSlave
        call    Enviar          ;Envía el dato al Slave
        call    Send_RStart     ;Envía condición de reinicio
        banksel DirSlave
        call    Recibir         ;Toma dato del Slave...
        movf    MensajeIn,W     ;...y lo guarda en...
        movwf   StatSVD01       ;...la variable de registro correspondiente
        call    Send_Stop       ;Envía condición de stop
        banksel DirSlave

        return



;*******************************************************************************
SetStatSVD01            ;Ordena por i2c al módulo de servomotor la actualización
                        ;de su registro especial de estado según el valor de
                        ;"StatSVD01". Ha de tenerse en cuenta que solo se actualizarán
                        ;aquellos bits que sean R/W. Los demás serán ignorados.
;*******************************************************************************

        banksel DirSlave
        movlw   DirSVD01
        movwf   DirSlave
        movlw   SVD01Stat       ;Registro 1 de SVD01 (Estado)
        movwf   MensajeOut
        call    Send_Start      ;Envía condición de inicio
        banksel DirSlave
        call    Enviar          ;Envía el dato al Slave
```

```
        movf    StatSVD01,W     ;Se deja en W para que...
        call    Send_Byte       ;...Send_Byte lo envíe por i2c
        call    Send_Stop       ;Envía condición de stop
        banksel DirSlave
        return


;*******************************************************************************
ReadHDBSVD01            ;Obtiene por i2c el registro HalfDeadBand del módulo de
                        ;servomotor y la deja en la variable "HDBSVD01"
;*******************************************************************************

        banksel DirSlave
        movlw   DirSVD01
        movwf   DirSlave
        movlw   SVD01HDB        ;Registro 6 de SVD01
        movwf   MensajeOut
        call    Send_Start      ;Envía condición de inicio
        banksel DirSlave
        call    Enviar          ;Envía el dato al Slave
        call    Send_RStart     ;Envía condición de reinicio
        banksel DirSlave
        call    Recibir         ;Toma dato del Slave...
        movf    MensajeIn,W     ;...y lo guarda en...
        movwf   HDBSVD01        ;...la variable de registro correspondiente
        call    Send_Stop       ;Envía condición de stop
        banksel DirSlave

        return


;*******************************************************************************
SetHDBSVD01             ;Ordena por i2c al módulo de servomotor que almacene
                        ;en su registro HalfDeadBand el valor dado por "HDBSVD01"
;*******************************************************************************

        banksel DirSlave
        movlw   DirSVD01
        movwf   DirSlave
        movlw   SVD01HDB        ;Registro 6 de SVD01
        movwf   MensajeOut
        call    Send_Start      ;Envía condición de inicio
        banksel DirSlave
        call    Enviar          ;Envía el dato al Slave
        movf    HDBSVD01,W      ;Se deja en W para que...
        call    Send_Byte       ;...Send_Byte lo envíe por i2c
        call    Send_Stop       ;Envía condición de stop
        banksel DirSlave
        return


;*******************************************************************************
ReadVerSVD01            ;Obtiene por i2c la versión de firmware del módulo de
                        ;servomotor y la deja en la variable "VerSVD01"
;*******************************************************************************

        banksel DirSlave
        movlw   DirSVD01
        movwf   DirSlave
        movlw   SVD01Ver        ;Registro 6 de SVD01
        movwf   MensajeOut
        call    Send_Start      ;Envía condición de inicio
        banksel DirSlave
        call    Enviar          ;Envía el dato al Slave
        call    Send_RStart     ;Envía condición de reinicio
        banksel DirSlave
        call    Recibir         ;Toma dato del Slave...
        movf    MensajeIn,W     ;...y lo guarda en...
        movwf   VerSVD01        ;...la variable de registro correspondiente
        call    Send_Stop       ;Envía condición de stop
        banksel DirSlave

        return
```

```
;-------------------------------------------------------------------------------
init_i2c_Master              ;Inicializa valores para uso de I2C en Master
                      ;Ha de ser llamado tras definir TRISC y un valor para
                      ;ClockValue. Para frecuencia SCL=Fosc/(4x(ClockValue+1))
;-------------------------------------------------------------------------------

       ;Guardamos copia de algunos registros
       movwf   BkW            ;Hace copia de W
       movf    STATUS,W       ;Hace copia de registro de estado
       banksel PORTA
       movwf   BkStatus

       ;Configuramos I2C
       banksel TRISC          ; Pasamos a direccionar Banco 1
       movlw   b'00011000'    ; Establece líneas SDA y SCL como entradas...
       iorwf   TRISC,f        ;..respetando los valores para otras líneas.
       movlw   ClockValue     ; Establece velocidad I2C segun...
       movwf   SSPADD         ; ...valor de ClockValue
       bcf     SSPSTAT,6      ; Establece I2C input levels
       bcf     SSPSTAT,7      ; Habilita slew rate
       banksel SSPCON               ; Pasamos a direccionar Banco 0
       movlw   b'00111000'    ; Master mode, SSP enable, velocidad segun...
       movwf   SSPCON         ; ... Fosc/(4x(SSPADD+1))
       bcf     PIR1,SSPIF     ; Limpia flag de eventos SSP
       bcf     PIR1,7         ; Limpia bit. Mandatorio por Datasheet

       ;Restauramos las copias de los registros
       movf    BkStatus,W     ;Restaura las copias de registros
       movwf   STATUS         ;registro de estado
       movf    BkW,W          ;registro W

       return


; ----------------------------------------------------------------------------------------
Enviar ;Envía un mensaje (comando) almacenado en "MensajeOut" al Slave cuya dirección
       ;se ha de encontrarse en la variable "DirSlave"
;-----------------------------------------------------------------------------------------


       ;Guardamos copia de algunos registros
       movwf   BkW            ;Hace copia de W
       movf    STATUS,W       ;Hace copia de registro de estado
       banksel PORTA
       movwf   BkStatus

StEnv
       banksel DirSlave
       movf    DirSlave,W     ;Dirección esclavo
       call    Send_Byte      ;Envía dirección y orden de escritura
       call    WrtAckTest     ;Verifica llegada ACK
       banksel SSPCON2
       bcf     SSPCON2,ACKSTAT       ;limpia flag ACK
       xorlw   1
       btfss   STATUS,Z       ;Chequea si llegó ACK
       goto    SigueEnv       ;Si. Seguimos con envío dato
       call    Send_Stop      ;No. Reintentamos envío
       call    Send_Start
       goto    StEnv

SigueEnv
       banksel MensajeOut
       movf    MensajeOut,W   ;Lo deja en W para que la subrutina Send_Byte lo envíe

       call    Send_Byte      ;envía por i2c

       ;Restauramos las copias de los registros
       movf    BkStatus,W     ;Restaura las copias de registros
       movwf   STATUS         ;registro de estado
       movf    BkW,W          ;registro W

       return
```

```
; ------------------------------------------------------------------------------------------
Recibir ;Solicita dato al Slave cuya dirección ha de encontrarse en la variable
        ;"DirSlave" y lo mete en "MensajeIn".
;------------------------------------------------------------------------------------------

        ;Guardamos copia de algunos registros
        movwf   BkW             ;Hace copia de W
        movf    STATUS,W        ;Hace copia de registro de estado
        banksel PORTA
        movwf   BkStatus

StRec
        banksel DirSlave
        movf    DirSlave,W      ;Dirección esclavo
        iorlw   b'00000001'     ;con orden de lectura
        call    Send_Byte       ;Envía dirección y orden de lectura
        call    WrtAckTest      ;Verifica llegada ACK
        banksel SSPCON2
        bcf     SSPCON2,ACKSTAT       ;limpia flag ACK
        xorlw   1
        btfsc   STATUS,Z        ;Chequea si llegó ACK
        goto    StRec           ;No. Reintentamos envío
                                ;Si. Leemos dato
        call    Rec_Byte        ;Recibe dato por i2c y lo mete en "MensajeIn"

        ;Restauramos las copias de los registros
        movf    BkStatus,W      ;Restaura las copias de registros
        movwf   STATUS          ;registro de estado
        movf    BkW,W           ;registro W

        return



; ------------------------------------------------------------------------------------------
Send_Start      ;Envía condición de start
;------------------------------------------------------------------------------------------

        banksel SSPCON2
        bsf     SSPCON2,SEN     ; Envía Start
        call    CheckIdle       ;Espera fin evento
        return


; ------------------------------------------------------------------------------------------
Send_RStart     ;Envía condición de Repeated Start
;------------------------------------------------------------------------------------------

        banksel SSPCON2
        bsf     SSPCON2,RSEN    ; Envía Repeated Start
        call    CheckIdle       ;Espera fin evento
        return


; ------------------------------------------------------------------------------------------
Send_Ack        ;Envía Ack
;------------------------------------------------------------------------------------------

        banksel SSPCON2
        bcf     SSPCON2,ACKDT   ; acknowledge bit state to send (ack)
        bsf     SSPCON2,ACKEN   ; Inicia secuencia de ack
        call    CheckIdle       ;Espera fin evento
        return


; ------------------------------------------------------------------------------------------
Send_Nack       ;Envía Nack para finalizar recepción
;------------------------------------------------------------------------------------------

        banksel SSPCON2
        bsf     SSPCON2,ACKDT   ; acknowledge bit state to send (not ack)
        bsf     SSPCON2,ACKEN   ; Inicia secuencia de nack
        call    CheckIdle       ;Espera fin evento
        return
```

```
; ------------------------------------------------------------------------------------
Send_Stop      ;Envía condición de stop
;------------------------------------------------------------------------------------

       banksel SSPCON2
       bsf     SSPCON2,PEN    ;Activa secuencia de stop
       call    CheckIdle      ;Espera fin evento
       return


; ------------------------------------------------------------------------------------
Send_Byte      ;Envía el contenido de W por i2c
;------------------------------------------------------------------------------------

       banksel SSPBUF                ; Cambia a banco 0
       movwf   SSPBUF         ; inicia condicion de escritura
       call    CheckIdle      ;Espera fin evento
       return



; ------------------------------------------------------------------------------------
Rec_Byte       ;Recibe dato por i2c y lo mete en "MensajeIn"
;------------------------------------------------------------------------------------

       banksel SSPCON2        ; Cambia a banco 1
       bsf     SSPCON2,RCEN   ; genera receive condition
       btfsc   SSPCON2,RCEN   ; espera a que llegue el dato
       goto    $-1
       banksel SSPBUF         ; Cambia a banco 0
       movf    SSPBUF,w       ; Mueve el dato recibido ...
       movwf   MensajeIn      ; ...  a MensajeIn
       call    CheckIdle      ;Espera fin evento
       return


; ------------------------------------------------------------------------------------
CheckIdle      ;Chequea que la operación anterior termino y se puede proceder con
               ;el siguiente evento SSP
;------------------------------------------------------------------------------------

       banksel SSPSTAT               ; Cambia a banco 1
       btfsc   SSPSTAT, R_W   ; Transmisión en progreso?
       goto    $-1
       movf    SSPCON2,W
       andlw   0x1F           ; Chequeamos con mascara para ver si evento en progreso
       btfss   STATUS, Z
       goto    $-3            ; Sigue en progreso o bus ocupado. esperamos
       banksel PIR1           ; Cambia a banco 0
       bcf     PIR1,SSPIF     ; Limpiamos flag
       return


;------------------------------------------------------------------------------------
WrtAckTest     ;Chequea ack tras envío de dirección o dato
               ;Devuelve en W 0 o 1 dependiendo de si llegó (0) o no (1) ACK
;------------------------------------------------------------------------------------

       banksel SSPCON2        ; Cambia a banco 1
       btfss   SSPCON2,ACKSTAT ;Chequea llegada ACK desde slave
       retlw   0              ;llegó ACK
       retlw   1              ;no llegó ACK


;------------------------------------------------------------------------------------
HacerTiempo    ;realiza una pausa del numero de centesimas de segundo especificadas en "Pausa"

;------------------------------------------------------------------------------------

       movf    Pausa,W        ;Coloca el valor de pausa en BDel2...
       movwf   BDel2          ;...para no alterar su contenido

;.........................................................
; Generado con PDEL ver SP  r 1.0  el 24/02/03 Hs 18:31:22
; Descripcion: Delay 10000 ciclos (1 centésima de segundo)
;.........................................................
```

```
BCiclo  movlw     .8        ; 1 set numero de repeticion  (B)
        movwf     BDel0     ; 1 |
BLoop1  movlw     .249      ; 1 set numero de repeticion  (A)
        movwf     BDel1     ; 1 |
BLoop2  nop                 ; 1 nop
        nop                 ; 1 ciclo delay
        decfsz    BDel1, 1  ; 1 + (1) es el tiempo 0  ? (A)
        goto      BLoop2    ; 2 no, loop
        decfsz    BDel0,  1 ; 1 + (1) es el tiempo 0  ? (B)
        goto      BLoop1    ; 2 no, loop
BDelL1  goto BDelL2         ; 2 ciclos delay
BDelL2  nop                 ; 1 ciclo delay
;.....................................................
       decfsz  BDel2,F       ;Repite tantas veces el ciclo de una decima de segundo...
       goto    BCiclo        ;..como se lo indique ADel2
        return              ; 2+2 Fin.


END
```

# Líneas futuras

El modelo de servocontrol presentado permite mejoras considerables, como la reducción de tamaño y sobre todo un control más preciso de la posición del servo mediante la reducción de velocidad al acercarse a la posición seleccionada, con lo que se evitará el balanceo de ajuste que puede acontecer con valores de "HalfDeadBand" bajos y escasa carga.

Indudablemente una evolución de los servomotores de radiocontrol hacia los servomotores $I^2C$ facilitaría enormemente el control de cantidades grandes de servos en sistemas complejos y evitaría el uso inútil de puertos y capacidad de proceso de los microcontroladores master.

Son este tipo de servomotores y no los de radiocontrol los que realmente requiere la robótica. No solo por su facilidad de control y su reducida necesidad de proceso, sino también por la amplia información para control en bucles cerrados que provee.