ESCUELA POLITÉCNICA DEL EJECITO

FACULTAD DE INGENIERIA ELECTRONICA

PROYECTO DE GRADO PARA LA OBTENCION DEL TITULO EN INGENIERIA ELECTRONICA

"DISEÑO Y CONSTRUCCIÓN DE MÓDULOS PERIFÉRICOS PARA EL SISTEMA DE DESARROLLO DE PIC´S SDP-877 MP-SDP877"

MYRIAM ELIZABETH CISNEROS TORRES

SANGOLQUI – ECUADOR

ENERO 2005

AGRADECIMIENTOS

El agradecimiento mas importante es para mi Padre Dios, aquel Ser que me dio la fuerza, el optimismo, la capacidad y la sabiduría necesaria no solo para realizar este trabajo, sino para culminar mi carrera. Fue El quien me levanto cuando muchas veces caí, fue El quien me dio aliento para recuperarme y mirar adelante en momentos difíciles de mi vida y fue El quien puso en mi camino a personas maravillosas e importantes para mi.

A mis padres por su eterno e incondicional apoyo, por todos sus consejos, por su preocupación, por esa dedicación hacia sus hijos que hoy en día es muy difícil de encontrar, por acompañarme en toda la carrera, por estar a mi lado en momentos felices, tristes, emocionantes y decepcionantes, pero siempre brindándome su sonrisa y sus brazos para refugiarme.

A mi Abuelita preciosa, Orfelina, por el inmenso y eterno amor que me brinda, porque a pesar de no saber de temas de electrónica siempre estuvo junto a mi, dándome alas para soñar y luchar por lo que quiero, por impulsarme tan solo con su cariño, con sus abrazos y besos a culminar uno de mis primeros sueños, el obtener el titulo por el cual siempre soñé.

A mi abuelita Eloisa, por el apoyo que me brindo, por su ayuda y sus sabios consejos. Ella es una persona que con ejemplo me ha demostrado lo fuertes y valientes que podemos ser las mujeres en todos los aspectos.

A mi hermana, Lorena, por ser mi confidente, mi amiga, mi respaldo, mi fuerza y una motivación para mi. Por ayudarme a reponerme de mis malos momentos y celebrar conmigo los buenos. Por cuidarme y quererme de una manera tan especial y que hasta hoy no había entendido.

A mi familia en general, a mis tíos, mis primos y demás, por apoyarme a pesar de la distancia física que nos separa, por cada llamada, por cada consejo, por esa preocupación y apoyo para alcanzar mis metas.

A mis amigos, por compartir junto a mi 5 maravillosos años, en los que no solamente tuvimos momentos de risas, también hubieron días tristes en los que unos a otros nos dábamos consejos y nos refugiábamos en nuestra gran amistad. Por apoyarme y por la tranquilidad que me transmitían cuando estaban junto a mi.

A todas aquellas personas que ya no están conmigo aquí en la Tierra, a mi abuelito Luis, a mi amiga Yolita, grandes seres humanos, que alguna vez soñaron conmigo y compartieron mis anhelos, porque al recordarlos me llenaba de fuerza, por ser una motivación muy fuerte para mi, porque no podía decepcionarlos y aunque ya no pueda verlos... yo los siento junto a mi siempre, los siento felices y orgullosos de lo que he logrado.

A los Ingenieros Víctor Proaño y Hugo Ortiz, por su apoyo y colaboración en la realización de esta tesis.

DEDICATORIA

A mi Padre Dios, por ser mi fuerza, mi esperanza, mi guía y mi respaldo.

A mi papi, Luis, por su dedicación y entrega a sus hijas, por luchar contra todo y todos para darme todo lo que necesito, no solo material sino moral, espiritual y sentimentalmente.

A mi mamita preciosa, Estela, mi guía, mi confidente, mi motivación. Por ser una madre como pocas en el mundo, entregada y abnegada con sus hijas, porque no solamente nos ha entregado su vida, sus sueños, sus esperanzas y su amor, sino que también nos ha ensenado el respeto, la lealtad, la honestidad, la perseverancia y a luchar por lo que queremos a pesar de los obstáculos que se presenten.

A mi abuelita Orfelina, por su amor, por cada palabra, por cada beso, por cada abrazo, en los que no solamente me transmitía amor sino fuerza y motivación.

A mi hermana Lorena, por su ayuda y apoyo, por ser siempre mi protectora y por luchar junto a mi durante todo estos años, en general por ser mi ángel guardián.

A mis tíos, por su preocupación y su animo, por enseñarme con su propio ejemplo que las cosas que uno quiere, se las puede conseguir si se trabaja duro por ellas.

A mi abuelito Luis, a quien hubiera deseado abrazar y besar este día, para compartir con el mi alegría, pero Dios no lo quiso así. A el, que a pesar de no poder ver, abrazar o besar sigue junto a mi apoyándome y dándome fuerzas. Porque en un sueño me hablaste y me dijiste que hoy estarías aquí... este trabajo es para ti.

A mi amiga Yolita, que tampoco pudo estar conmigo durante este tiempo, pero que vive en mi corazón, ella es una llama y una voz que me anima a seguir adelante y luchar por lo que quiero.

A mis amiguitos, con quienes compartí maravillosos e inolvidables momentos, por ser esa fuerza que me impulsaba a seguir adelante, por ser mi apoyo, por escucharme, por compartir mis problemas, por estar junto a mi cuando mas los necesitaba.

A un grupo de amigos, que llegaron a convertirse en mis hermanos, seres humanos con tanta calidad humana que son capaces de irradiar el día mas nublado con sus palabras y con cada ocurrencia. Por iluminar mis días grises y traer un brillo especial a mi vida: Pablito, Joe, Daniel, Cristina, Christian G, Fito, Christian S, Jenny, Ramiro, Alex, David, Rita, Rommel, Raquel, Roberto.

PROLOGO

El propósito del presente trabajo es realizar un estudio y elaborar una guía de prácticas de dos temas específicos de mucha actualidad en el estudio de microcontroladores que son: la red de comunicación I2C y los microcontroladores RfPIC.

Existe dificultad en disponer de laboratorios actualizados en el área de microcontroladores, lo cual se debe principalmente al continuo desarrollo de la tecnología y la aparición de dispositivos cada vez mas sofisticados que ameritan ser analizados y estudiados. Debido a este hecho, en la facultad de Ingeniería Electrónica de la ESPE hacen falta laboratorios actualizados en el área de microcontroladores que integren módulos de aplicación práctica. Una forma posible de resolver el problema es construir sistemas que permitan acceder al estudio de los dispositivos nuevos, sin la pérdida de tiempo que se producen por malas conexiones o dificultad de encontrar los elementos electrónicos en el mercado nacional. En base a este precepto, se ha implementado ya un sistema de desarrollo de microcontroladores SDP877 el cual cumple funciones básicas para el estudio de las aplicaciones de microcontroladores.

Como un complemento del sistema SDP877, en este proyecto se realiza el diseño y construcción de módulos de aplicación práctica, cuyo fin será demostrar al estudiante otras utilidades que se pueden obtener de los microcontroladores.

Cuando se dispone de un sistema de desarrollo la principal ventaja que se obtiene es la disminución del tiempo de realización de proyectos, al disponer de herramientas que resuelven la parte básica de los problemas que se encuentran en las diversas aplicaciones.

El principal objetivo del sistema construido **MP-SDP877** es facilitar a los alumnos la realización de aplicaciones prácticas de la red I2C y los RfPIC, sin embargo, se deben indicar algunos otros objetivos que han sido cumplidos con el desarrollo del proyecto:

- Diseñar y construir módulos periféricos para el Sistema de desarrollo de microcontroladores SDP877.

- Diseñar y construir un módulo para manejo de la interfaz I2C con posibilidad de interconexión al Sistema de desarrollo para microcontroladores.

- Realizar un estudio y análisis de los microcontroladores rfPIC para establecer posibles aplicaciones mediante un Kit de Desarrollo.

- Disponer de una herramienta que facilite la utilización de los microcontroladores en aplicaciones prácticas.

El **CAPITULO I** hace un análisis de los módulos transmisores y receptores del rfKIT. Se tratarán las características mas importantes de los PIC's transmisor y receptor. Además se explicará el funcionamiento de las tarjetas.

En el **CAPITULO II** se describe el diseño para el manejo de la red de comunicaciones I2C, se realizará un análisis del bus, junto a las características y requerimientos de la misma.

El esquema electrónico, la configuración de los puertos de los PIC's a emplear en la comunicación I2C, así como los diagramas de conexión se indican en el **CAPITULO III**.

El **CAPITULO VI** describe las prácticas realizadas usando el MP-SDP877 y el kit de transmisión RF. Se presentan diagramas de flujo, explicación de los programas y el código fuente.

Las conclusiones y recomendaciones del proyecto realizado se indican en el **CAPITULO V**.

INDICE

Contenido **Pág.**

ESQUEMA ELECTRONICO

CAPITULO 4

PRÁCTICAS Y APLICACIÓNES PARA EL MÓDULO PERIFERICO

PRÁCTICAS REALIZADAS

DIAGRAMA ELECTRONICO Y CIRCUITO IMPRESO DE LA PLACA DEL MP-SDP877 Y DEL KIT DE DESARROLLO DE RF PIC'S

ANEXO 2

DIAGRAMAS DE FLUJO DE LA APLICACIÓN AN736, "AN I2C NETWORK PROTOCOL FOR ENVIRONMENTAL MONITORING"

ANEXO 3

CODIGO FUENTE DEL SOFTWARE DE LA APLICACIÓN AN736, "AN I2C NETWORK PROTOCOL FOR ENVIRONMENTAL MONITORING"

ANEXO 4

CODIGO FUENTE DEL SOFTWARE DE LA TRANSMISION-RECEPCION CON RF PIC'S

ANEXO 5

HOJAS DE DATOS TECNICOS DE LOS COMPONENTES

INDICE DE FIGURAS

INDICE DE TABLAS

GLOSARIO

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

CAPITULO 1

TRANSMISIÓN Y RECEPCIÓN CON RFPIC

Para la presente aplicación se ha empleado el kit de desarrollo rfPIC, correspondiente a la empresa Microchip. A continuación se hará un estudio al mencionado kit y luego se procederá a realizar una explicación detallada del programa que genera la aplicación.

1.1 Módulo Transmisor

MICROCONTROLADORES RFPIC CON TRANSMISOR RF

1.1.1 Información General

Los elementos microcontroladores rfPIC integran la potencia de los PICmicro® de Microchip con las capacidades de comunicación gíreles (inalámbrica), para aplicaciones de RF con baja potencia. Los elementos favoritos en el mercado son de Microchip, los PICmicro, con arquitectura RISC que son fáciles de programar y familiares para los diseñadores alrededor de todo el mundo. Los elementos ofrecen un pequeño encapsulado y componentes externos de menor tamaño, a fin de aprovechar espacio dentro de una aplicación (Ilustración 1).

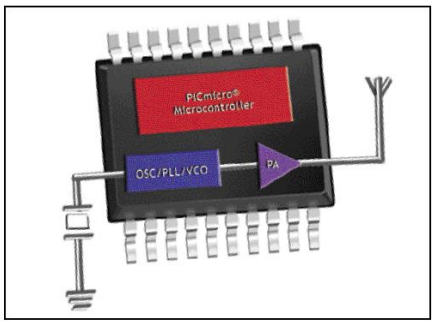

Ilustración 1 rfPIC 12F675

El rfPIC12F675 es un diseño transmisor ASK, de bajo costo y alto desempeño de radio UHF que se usa como el PIC rf12F675K para frecuencias de 315 MHz y el rf12F675F para 433.92 MHz.

El diseño del módulo es aplicable para:

- Comando y control remoto, de forma inalámbrica.
- Entrada remota sin ningún tipo de llave.
- Sistemas de seguridad.
- Aplicaciones telemétricas de baja potencia.

1.1.2 Descripción.

El rfPIC12F675 (Ilustración 2) es un módulo de transmisión único que puede ser usado en diferentes formas. El módulo transmisor muestra diferentes aspectos destacados del PIC transmisor, el RF12F675.

El módulo transmisor consiste de:

- 2 pulsadores conectados a GP3 y GP4
- 2 potenciómetros conectados a GP0 y GP1
- habilitador RF (RF_{EN}) conectado a GP5
- Datos ASK ($DATA_{ASK}$) conectados a GP2

- Zócalo opcional de 8 pines para la versión de PIC12F675 con encapsulado de 8 pines

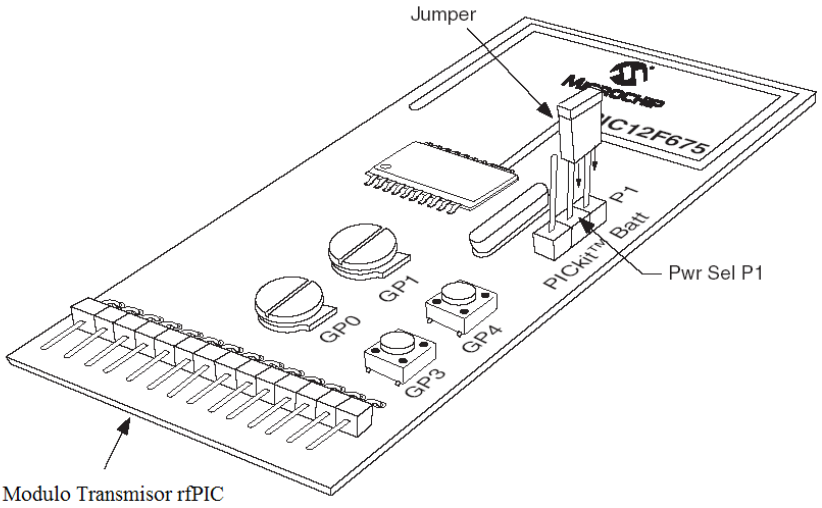

Ilustración 2 Estructura del Módulo Transmisor

1.1.3 Requerimientos.

El jumper P1, selecciona una de las dos fuentes de poder para el rfPIC12F675.

En el PICkit Stater Kit, se ubica el jumper para permitir al módulo transmisor ser energizado desde el Kit principal.

NOTA: Cuando se programa el módulo transmisor en el PICkit Satarter Kit, el jumper Pwr Sel P1 debe estar ubicado en la posición de los pines 1 y 2.

Con el jumper en la posición de pines 1 y 2, se permite al módulo transmisor ser energizado con la batería de litio. Cuando se energiza desde la batería, el módulo transmisor puede ser empleado en forma portable.

1.1.4 Programación del rfPIC12F675

El rfPIC12F675 puede ser programado gracias al PICKit 1 Flash Starter Kit.

Simplemente deben seguirse los siguientes pasos:

Paso 1: Remover el PIC16F676 o el PIC12F676 del zócalo del PICkit Starter Kit.

Paso 2: Conectar el módulo transmisor en la cabecera de expansión del PICkit Starter Kit, J3 (Ilustración 3)

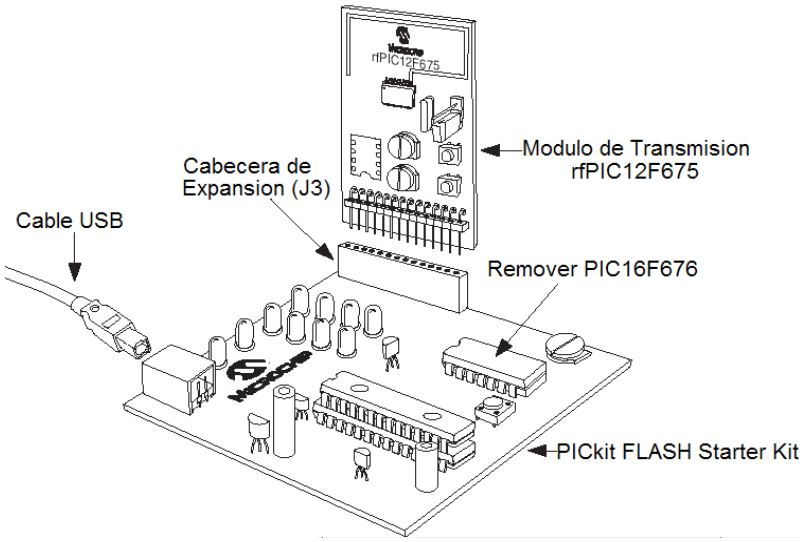

Ilustración 3 Forma de Programar el módulo rfPIC12F675

Paso 3: el rfPIC12F675 del módulo transmisor, se convierte ahora en el elemento a programar.

1.2 Microcontrolador Flash rfPIC12f675 Con Transmisor ASK/FSK

1.2.1 Características técnicas.

- Velocidad de operación: oscilador interno de 4MHz, calibrado a ±1%
- Capacidad de interrupción
- 8 niveles de pila
- Modos de direccionamiento directo e indirecto

1.2.2 Características generales.

- Memoria:
  Memoria FLASH de programa 1024 x 14 palabras
  Memoria EEPROM de datos 128 x 8 bytes
  Memoria SRAM de datos 64 x 8 bytes
  100 000 escrituras en FLASH
  1 000 000 escrituras en EEPROM
  Retención de datos FLASH/EEPROM mayor a 40 años
- Protección de código programable
- 6 pines de entrada/salida, weak pull-ups y detección de cambio en pin
- Comparador análogo de 16 niveles de referencia interna
- Conversor análogo/digital 10 bits, 4 canales
- Timer0 con 8 bits timer/contador con 8 bits de prescaler
- Timer1 con 16 bits timer/contador con 3 bits de prescaler
- 5 us de reacción del SLEEP con 3V de alimentación

1.2.3 Características Secundarias.

- Consumo bajo de energía
- 14 mA transmitiendo +6dBm a 434 MHz
- 4 mA transmitiendo -15dBm a 434 MHz

- 500 uA, 4.0 MHz INTOSC (Oscilador interno)
- Rango de voltaje para operación 2.0 a 5.5 V
- Rango de temperatura industrial.

1.2.4 Transmisor UHF ASK/FSK

- Tasa de transmisión de datos ASK 0-40 Kbps
- Tasa de transmisión de datos FSK 0-40 Kbps
- Potencia de salida +10 dBm a –12 dBm en 4 pasos
- Consumo de potencia ajustable para transmisión

1.2.5 Aplicaciones

- Sistemas de seguridad remoto
- Sistemas de alarma automotriz
- Puertas de condominios y de garaje
- Sistemas de alarma regulares
- Sistemas de alarma contra robos
- Acceso a edificios
- Sensores wíreless (inalámbricos)

1.2.6 Configuración de Pines

El integrado 12F675 posee 20 pines distribuidos de la siguiente manera:

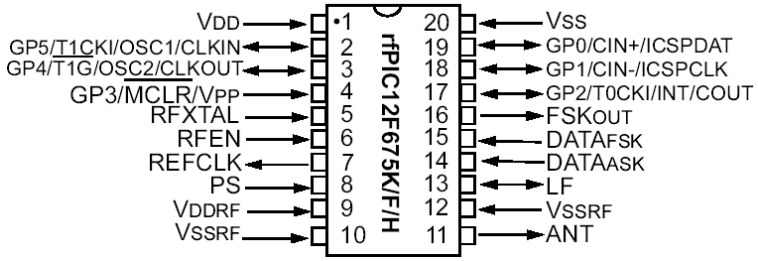

Ilustración 4 Distribución de los pines del integrado

1.2.7 Transmisión ASK/FSK de UHF

1.2.7.1 Operación del transmisor

El transmisor es un integrado completo de UHF con ASK/FSK que consiste de un cristal oscilador, Lazo cerrado de fase (PLL), un amplificador de potencia con salida de colector abierto y un modo de control lógico.

Existen tres variaciones de este elemento para optimizar su desempeño en las bandas de frecuencia mas comúnmente utilizadas (véase la Tabla 1).

Elemento	Frecuencia	Modulación
RfPIC12F675K	290-350 MHz	ASK/FSK
RfPIC12F675F	390-450 MHz	ASK/FSK
RfPIC12F675H	850-930 MHz	ASK/FSK

Tabla 1 Detalle de bandas de frecuencia de los rfPIC's

La estructura interna del transmisor es mostrada en la Ilustración 5**.** Un oscilador Colpitts genera una frecuencia de referencia establecida por el cristal adjunto. El oscilador controlador de voltaje (VCO) convierte el voltaje en el pin LF a frecuencia. Esta frecuencia es dividida para 32 y comparada con la referencia del cristal. La señal de salida VCO es amplificada por PA, cuya única salida dirige a la antena de usuario.

Los componentes externos requieren un cristal para establecer la frecuencia de transmisión.

Las dos señales de control del microcontrolador son conectadas externamente para una máxima flexibilidad en el diseño. El rfPIC12F675 es capaz de transmitir datos por ASK o FSK.

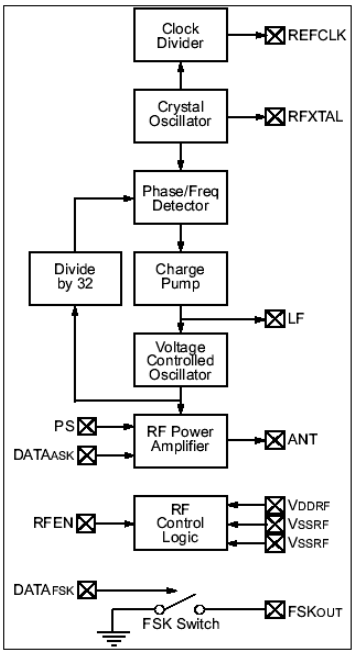

Ilustración 5 Diagrama de Bloques del Transmisor

El rfPIC12F675 es un elemento emisor de radio frecuencia (RF). Los elementos wireless RF son gobernados por una agencia estatal reguladora. Por ejemplo, en los Estados Unidos es el Comité de Comunicaciones Federal (FCC) y en Europa es la Administración Europea Postal y de Telecomunicaciones (CEPT). Es responsabilidad del diseñador el asegurarse que sus productos estén de acuerdo a las reglas y regulaciones del país de uso.

Los elementos RF requieren un nivel de implementación correcto, en función de conocer los requerimientos regulatorios. Se requiere conocer el plano del terreno para reducir emisiones de radiofrecuencias no deseadas.

1.2.7.2 Voltaje de Alimentación (VDDRF, VSSRF)

Los pines VDDRF y VSSRF son los que proveen voltaje y tierra, respectivamente al transmisor.

Estos pines de energía están separados de los pines de alimentación y tierra del microcontrolador.

Los dos pines VSSRF pueden ser colocados a tierra con la menor cantidad de conexiones posibles. La tierra del microcontrolador también debe ser conectada a la misma referencia de tierra del RF. Sin embargo la fuente VDDRF puede ser diferente a la polarización del microcontrolador, con tal que las entradas RFEN y DATA estén dentro de los límites de las especificaciones.

Consideraciones Generales.

- *Proveer baja impedancia en las conexiones de polarización y tierra para minimizar las emisiones no deseadas.*
- *No se deberá compartir las vías PCB con otras conexiones a tierra.*
- *Se deberá filtrar la señal VDDRF con un filtro RC si el ruido del microcontrolador excede los límites regulatorios.*

1.2.7.3 Cristal Oscilador

El cristal oscilador del transmisor es un oscilador Colpitts que provee una frecuencia de referencia al PLL. Esta es independiente del oscilador del microcontrolador. Un cristal externo es conectado al pin XTAL. La frecuencia de transmisión es arreglada y determinada por la frecuenta del cristal, en base a la ecuación 1.

$$f_{transmision} = f_{RFXTAL} \; x \; 32$$

Ecuación 1 Frecuencia de transmisión

Debido a la selección flexible de la frecuencia de transmisión, la frecuencia del cristal resultante tendrá un valor que no será estándar. Entonces para ciertas frecuencias el diseñador deberá consultar a una empresa productora de cristales y obtener los valores de los cristales que comúnmente manufacturan.

Para la modulación ASK, el cristal podrá ser conectado directamente desde el RFXTAL a tierra, o en serie con un condensador para mejorar la frecuencia. La Ilustración 6 muestra la manera en que el cristal debe ser conectado y en la Tabla 2 se indicará la forma en que la frecuencia de algunos cristales típicos cambian con la capacitancia.

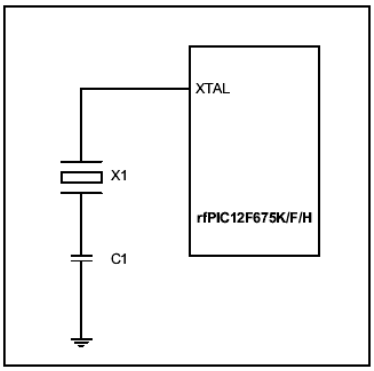

Ilustración 6 Circuito del cristal ASK

C1	Frecuencia Predicha (MHz)	PPM desde 13.55 MHz	Frecuencia de Transmisión (MHz) ($32 * f_{XTAL}$)
22 pF	13.551438	+ 106	433.646
39 pF	13.550563	+ 42	433.618
100 pF	13.549844	- 12	433.595
150 pF	13.549672	- 24	433.5895
470 pF	13.549548	- 33	433.5856
1000 pF	13.549344	- 48	433.579
Nota 1: Condiciones de Operación Estándar, T_A = 25 C, RFEN = 1, V_{DDRF} = 3V, f_{XTAL}=13.55 MHz			

Tabla 2 Frecuencia aproximada del oscilador XTAL vs. Capacitancia (modo ASK)

El oscilador es habilitado cuando la entrada RFEN esta en alto. El cristal se toma aproximadamente 1 ms para empezar a oscilar.

Los cristales de alta frecuencia inician su funcionamiento mas rápidamente que los cristales de bajas frecuencias.

1.2.7.4 Modulación ASK

En la modulación ASK, el dato es transmitido variando la potencia de salida. El pin DATAASK habilita el PA, haciendo de esta forma que la señal de salida RF se active y se desactive.

Un receptor simple usando un filtro y un diodo detector de picos es capaz de capturar el dato.

Un receptor mucho mas avanzado tal como el rfRXD0420 puede incrementar considerablemente el rango y reducir la susceptibilidad a la interferencia.

En el modo ASK, los pines DATAFSK y FSKOUT no son usados y se recomienda que ambos sean conectados a tierra.

Un ejemplo de un circuito ASK típico es mostrado en la Ilustración 7. El capacitor C1 puede ser reemplazado por uno más pequeño a fin de simplificar el diseño del transmisor si el receptor tiene un ancho de banda grande.

Para un receptor de banda angosta, el capacitor C1 podría necesitar ser reemplazado por un capacitor para sintonizar el transmisor a la frecuencia exacta.

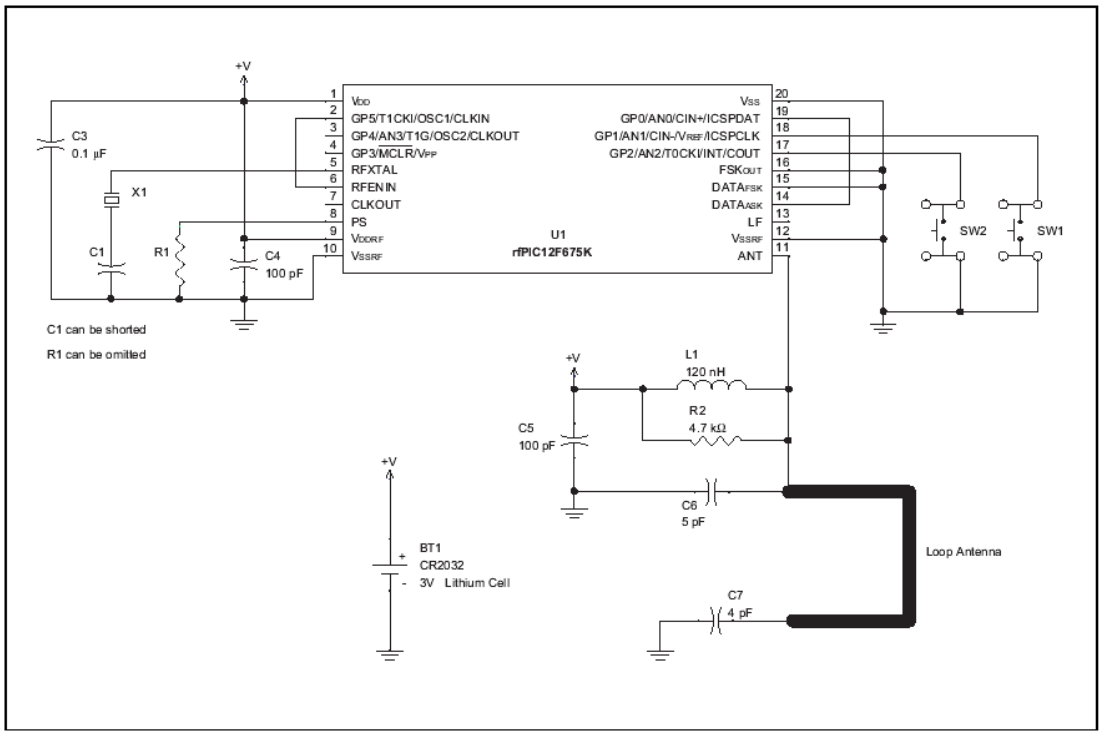

Ilustración 7 Esquema del transmisor ASK típico

1.2.7.5 Modulación FSK

En la modulación FSK el dato a transmitir es enviado mediante la variación de frecuencia en la salida. Esto es realizado cargando al cristal de referencia con un condensador extra para llevarlo a una frecuencia ligeramente mas baja.

Al conmutar la capacitancia con la señal del dato, se produce una variación del transmisor entre dos frecuencias.

Estas dos frecuencias básicas de cristal son entonces multiplicadas por 32 para obtener la frecuencia de transmisión RF.

Contrariamente a la frecuencia de transmisión ASK, la frecuencia central FSK no es transmitida actualmente. Este es el punto medio artificial entre las dos frecuencias transmitidas, calculado con la ecuación 2.

$$f_c = \frac{f_{max} + f_{min}}{2}$$

Ecuación 2 Frecuencia FSK central

Otro parámetro de gran importancia en FSK es la frecuencia de desviación de la frecuencia de transmisión. Esta mide que tan lejos la frecuencia se alejará de la frecuencia central. Esta desviación es calculada usando la ecuación 3.

$$\Delta f = \frac{f_{max} - f_{min}}{2}$$

Ecuación 3 Variación de la frecuencia de Transmisión

Un receptor FSK deberá especificar su valor óptimo de desviación. La desviación debe ser mayor a la tasa de transmisión de los datos/4. La mínima desviación es usualmente limitada por la exactitud de la frecuencia del transmisor y los componentes receptores. La máxima desviación es usualmente limitada por las características del cristal transmisor.

Un condensador extra y un switch interno son añadidos al diseño ASK para construir un transmisor FSK, como se muestra en la Ilustración 8. El capacitor C1 en serie con el cristal determina la máxima frecuencia.

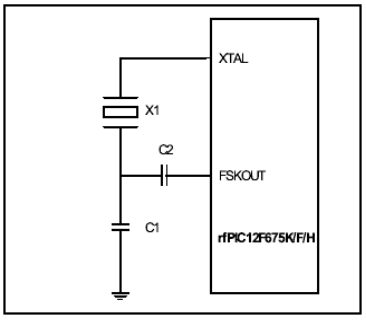

Ilustración 8 Circuito de Cristal FSK

Con el pin DATAFSK en alto, el pin FSKOUT esta abierto y el capacitor C2 no afecta de ninguna manera a la frecuencia. Cuando el pin DATAFSK cambia a bajo, el pin FSKOUT se conecta a tierra, y el C2 es ubicado en paralelo con C1. La suma de los dos elementos lleva a la frecuencia de oscilación a un valor mas bajo, como el mostrado en la Ilustración 9.

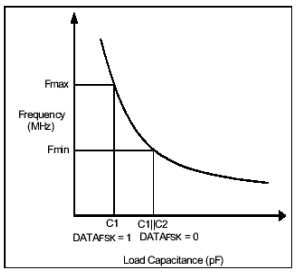

Ilustración 9 Variación de la frecuencia de Transmisión

Para conocer el diseño de un modulador FSK se puede observar en la ilustración 9-a.

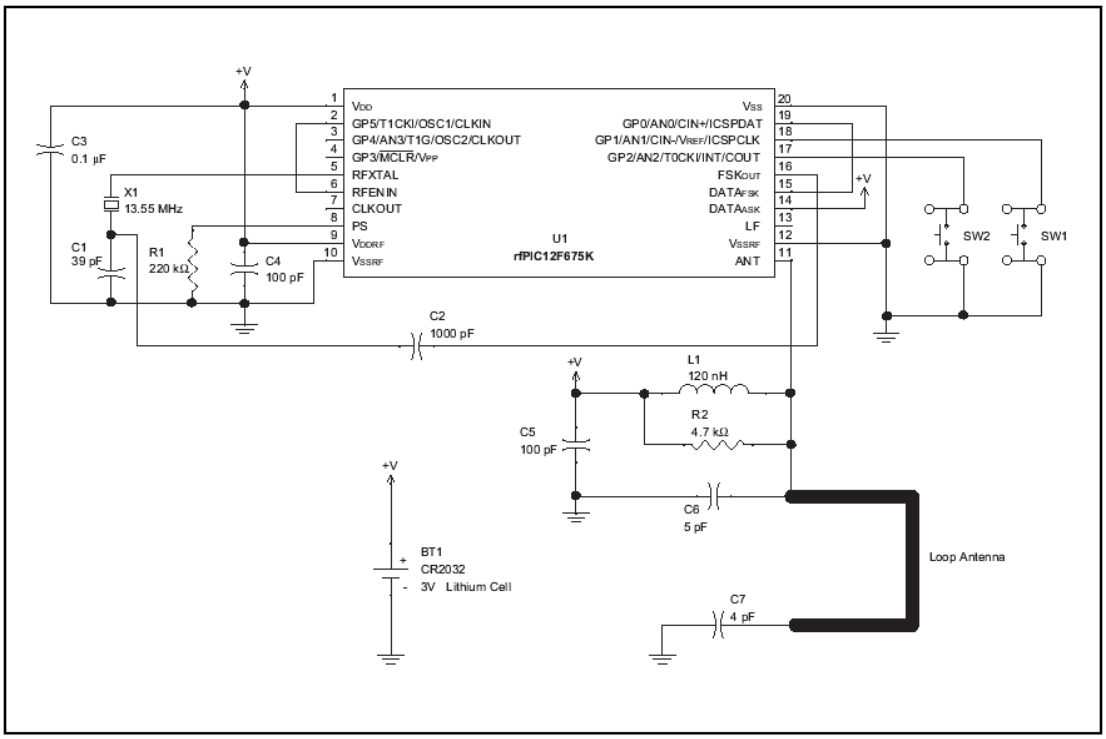

Ilustración 9-a Esquema del transmisor FSK típico

1.2.7.6 Salida de Reloj

La salida de reloj esta disponible al microcontrolador o a otros circuitos que requieran una frecuencia exacta de referencia.

Esta señal puede ser usada para corregir el oscilador interno RC para diseños de sistemas que requieren una exactitud en la sincronización de bits. La salida REFCLK puede ser conectada directamente a los pines T0CKI o al T1CKI.

La frecuencia de la salida REFCLK es el valor del cristal oscilador dividido para 4 en el rfPIC12F675K y en el rfPIC12F675F. Para el rfPIC12F675H el cristal oscilador es dividido para 8.

Consideraciones Generales.

Mantener los trazos del reloj cortos y angostos y tan lejos como sea posible del resto de trazos para reducir la capacitancia y el flujo de corriente asociada. Si el trazo REFCLK debe pasar cerca del cristal y los nodos LF, entonces debe ser cubiertos con conexiones de tierra.

El beneficio más importante es la conservación de la potencia de la batería. Otra razón es hacerlo más fácil para pasar los limites regulatorios. Y una tercera razón es para reducir la interferencia a otras comunicaciones en el espectro compartido de RF.

Las antenas pequeñas e ineficientes podrían requerir un alto nivel de potencia, mayor que en las antenas grandes y eficientes.

1.2.7.7 Desactivación de la Salida por bajo voltaje

El transmisor rfPIC12F675 tiene una construcción de desactivación por bajo voltaje centrado cerca de 1.85V. Si la fuente de voltaje alcanza un valor menor a este voltaje, el amplificador de voltaje es desactivado para prevenir transmisiones no controladas.

1.3 Módulo Receptor

1.3.1 Características del Módulo Receptor

El módulo receptor rfRXD0420 (Ilustración 10) es un diseño de radio ASK, de bajo costo, con alto rendimiento UHF de corto rango.

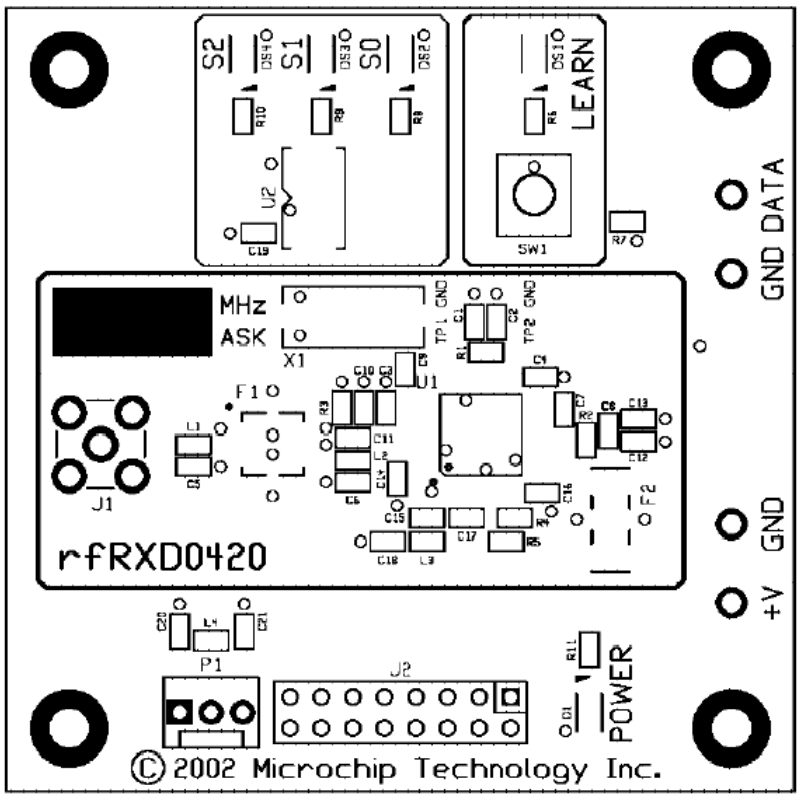

Ilustración 10 Módulo receptor rfRXD0420

El diseño es adecuado para:

- Comando y control remoto, vía wireless.
- Entrada remota sin ningún tipo de llave.
- Sistemas de seguridad.
- Aplicaciones telemétricas de baja potencia.

Las especificaciones para el módulo receptor son:

- Un único canal, ajustado a una frecuencia de 315 MHz y 433.92 MHz.
- Modulación ASK
- Tasa de transmisión: 4800 baudios

El rfRXD0420 es un módulo receptor único, que puede ser empleado de diferentes formas.

- Puede ser conectado a la cabecera de expansión del PICKit FLASH STARTER KIT con fines demostrativos y de desarrollo.

- El módulo receptor puede ser instalado en cualquier prototipo de prueba, demostración o propósitos de desarrollo.

Una descripción de los pines del módulo receptor rfRXD0420 UHF ASK/FSK se indica en la tabla 3.

Pin	Descripción
1-10	Sin conexión
11	Receptor de Datos de ingreso
12	Sin conexión
13	Alimentación: 2.5 – 5.5 VDC
14	Tierra
ANT	Conexión de la antena

Tabla 3 Funciones de los Pines del módulo receptor rfRXD0420

La conexión de la antena se realiza mediante un orificio de 0.055 pulgadas. Una antena de alambre simple de diámetro AWG 24, puede ser construida e insertada en el receptáculo. La longitud del cable dependerá de la frecuencia.

$$\lambda(metros) = {}^{c}\!/_{f}\ (Hertz)$$

donde:

$$c = 3x10^{8} = velocidad\ de\ la\ luz\ (m/s)$$
$$f = frecuencia\ de\ recepcion\ (Hertz)$$
$$\lambda = longitud\ de\ onda\ (metros)$$

La longitud del cable de la antena en pulgadas puede ser encontrado, para una frecuencia dada, aplicando la siguiente formula:

$$longitud\ de\ cable\ de\ la\ antena(pu\lg adas) = 2952.8 / f(MHz)$$

Alternativamente el pin que actúa como terminal de conexión de la antena puede ser removido y se puede realizar una conexión diferente. Por ejemplo, puede ser conectado un cable coaxial a la parte frontal de la placa y aterrizado en la placa por la parte posterior.

1.3.2 Descripción del rfRXD0420

El rfRXD0420 es un receptor de corta distancia de bajo costo, con un diseño compacto, que necesita de pocos elementos externos para convertirse en un módulo receptor completo. Cubre frecuencias de 300 MHz a 450 MHz.

Este rfPIC puede manejar modulaciones ASK, FSK y FM, siendo compatible con todos los elementos de las series de los transmisores RF y rfHCS.

Entre sus características destacan:

- Estabilidad de alta frecuencia sobre variaciones de temperatura y voltaje.
- Ganancia LNA ajustable para mejorar rangos de medición.
- Ancho de banda IF seleccionable mediante filtro cerámico externo. La frecuencia IF esta en el rango de 455 KHz a 21.4 MHz. Esto facilita el uso de filtros cerámicos disponibles en el mercado.
- Indicador de Fuerza de la señal recibida (RSSI) para detección ASK.
- Amplio rango de voltaje de alimentación.
- Emplea ASK para recepción de datos digitales.

- Modulación FM para recepción de señales análogas.
- Demodula FSK/FM usando un detector de cuadratura, también llamado detector de coincidencia de fase.

El rfRXD0420 es un dispositivo que contiene:

- Amplificador de bajo ruido (LNA) con ganancia seleccionable.
- Mezclador para conversiones de la señal RF a IF (frecuencia intermedia) seguido de un amplificador IF.
- Cristal oscilador
- Divisor de realimentación: en la versión 420 se divide para 16 y en la versión 920 se divide para 32.
- Amplificador limitador de IF para limitar y amplificar la señal IF y la RSSI.
- Demodulador, consiste de un detector de fase (Mixer2) y un amplificador, creando un detector de cuadratura para demodular la señal IF en aplicaciones FSK y FM.
- Amplificador operacional, OPA, que puede ser configurado como comparador de decisiones de datos ASK o FSK , o como un filtro para modulación FM.

La configuración de sus pines se presenta en la ilustración inferior. (Ver datasheet en ANEXO 5)

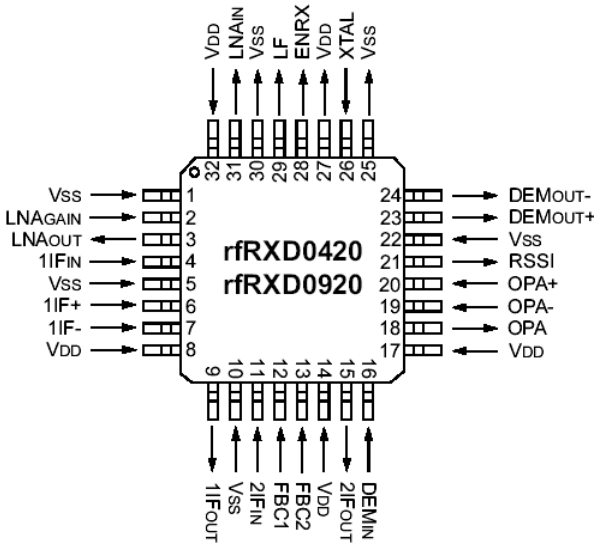

Ilustración 11 Distribución de pines del rfRXD0420

1.3.3 Desempeño del rfRXD0420 con señales ASK

En un circuito que maneja modulación ASK, el amplificador limitador IF con RSSI es usado como un detector ASK. La señal RSSI es filtrada luego de ser detectada y después comparada con un voltaje de referencia para determinar si la señal RF ingresada es un cero o un uno lógico. El voltaje de referencia puede ser configurado como un nivel de referencia dinámico, determinado por la señal RF ingresada o por un nivel predeterminado.

1.3.3.1 Filtraje post detector RSSI

La señal RSSI es filtrada para remover altas frecuencias y ruido, para asistir en el proceso de toma de decisiones del comparador e incrementar la sensibilidad del receptor. El filtro pasa bajo de la señal RSSI es un filtro RC creado por la impedancia de salida RSSI de 36 KΩ y el condensador C1. La constante de tiempo (RC=τ) del filtro RC depende del período de la señal y de cuando se realiza la decisión de la señal.

1.3.3.2 Periodo de la señal

Una sensibilidad óptima del receptor con una distorsión de pulso razonable ocurre cuando la constante de tiempo del filtro RC esta entre 1 y 2 veces el período de la señal. Si la constante de tiempo del filtro RC se establece con un valor muy corto el beneficio en el filtraje de ruido es pequeño. Sin embargo, si la constante tiene un valor muy grande, los pulsos de datos llegan a ser alargados causando interferencia entre símbolos.

1.3.3.3 Decisión de la señal

Si la decisión del bit ocurre en el centro del período de la señal, entonces la constante de tiempo debe colocarse en un valor menor o igual a la mitad del periodo de la señal. La ilustración 12 indica este método. El trazo superior representa la señal recibida de on-off, el trazo inferior muestra la señal RSSI luego de su paso por el filtro RC.

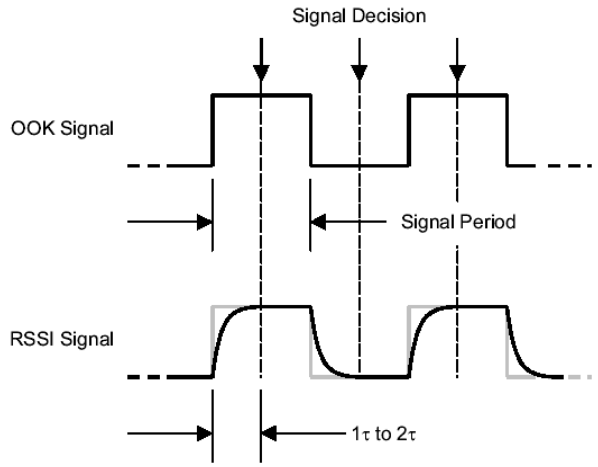

Ilustración 12 Bit de decisión ocurre en el centro del periodo

Si la decisión del bit ocurre cerca del final del periodo de la señal, entonces la constante de tiempo puede ser establecida con un valor menor o igual al periodo de la señal. La ilustración 13 muestra este método.

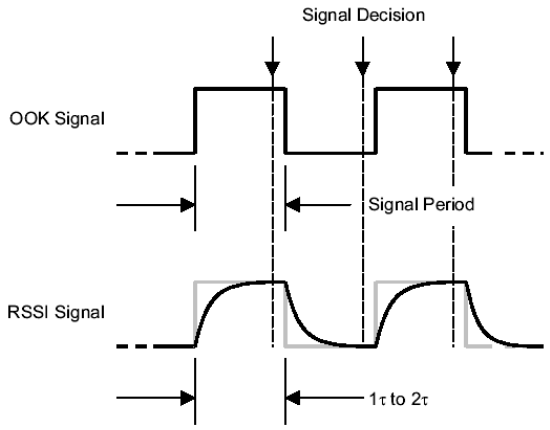

Ilustración 13 Bit de decisión ocurre cerca del final del periodo de la señal

Una vez que el tiempo de decisión de la señal y el período de la señal son conocidos, el condensador C1 puede ser seleccionado. Luego que C1 sea seleccionado, el diseñador puede observar la señal RSSI con un osciloscopio y realizar las pruebas correspondientes.

1.3.3.4 Comparador

El amplificador operacional interno esta configurado como un comparador. La señal RSSI es aplicada al Terminal OPA+, correspondiente al pin 20 y luego comparado con un voltaje de referencia en OPA- (pin 19) para determinar el nivel lógico de la señal recibida. El voltaje de referencia puede ser dinámico o estático.

La elección de referencias de voltaje Dinámicas vs. Estáticas depende en parte de la relación de unos lógicos vs. ceros lógicos. Una vez obtenida la relación, el voltaje de referencia dinámico puede ser generado con un simple filtro pasabajo. La ventaja del voltaje de referencia dinámico es el incremento de sensibilidad del receptor comparada con un voltaje de referencia preestablecido.

La elección de un voltaje de referencia estático depende en parte del contenido de DC del dato. Esto es, que el dato tiene un número disparejo de

unos lógicos versus ceros lógicos. La desventaja del voltaje de referencia estático es el decremento de sensibilidad del receptor, comparado con el voltaje de referencia dinámico.

1.3.3.5 Voltaje de referencia dinámico

El voltaje de referencia dinámico puede ser obtenido promediando la señal recibida con un filtro pasabajo. El ejemplo de un circuito de aplicación ASK es mostrado en la Ilustración 7, en ella el filtro pasabajo esta formado por R1 y C2. La salida del filtro pasabajo es alimentada a OPA-.

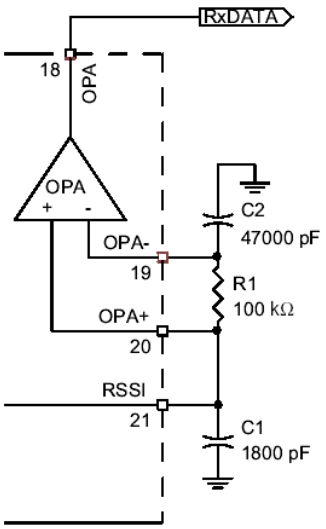

Ilustración 14 Sección del rfRXD0420, R1 y C2

El establecimiento de la constante de tiempo R1-C2 depende de la relación de unos y ceros lógicos, y un compromiso entre la estabilidad y el tiempo de reacción del receptor. Si la señal recibida tiene un número parejo de unos lógicos versus ceros lógicos, la constante de tiempo puede ser relativamente pequeña.

Así, el voltaje de referencia puede reaccionar rápidamente a cambios en la amplitud de la señal recibida y diferencias en los transmisores.

Si la constante de tiempo es muy grande, el voltaje de referencia será más estable. Sin embargo, el receptor no podrá reaccionar tan rápido a una señal dada.

Para la selección de valores de R1 y C2 (Ilustración 14), primero se empezará con la constante de tiempo entre 10 y 100 veces la tasa de la señal. Segundo, observar el voltaje de referencia contra la señal RSSI para determinar si los valores son los adecuados. La ilustración 15 es una pantalla de osciloscopio que captura una señal RF de onda cuadrada modulada. El trazo superior es el dato de salida de OPA (pin 18). Los dos trazos inferiores son la señal RSSI y el voltaje de referencia generado.

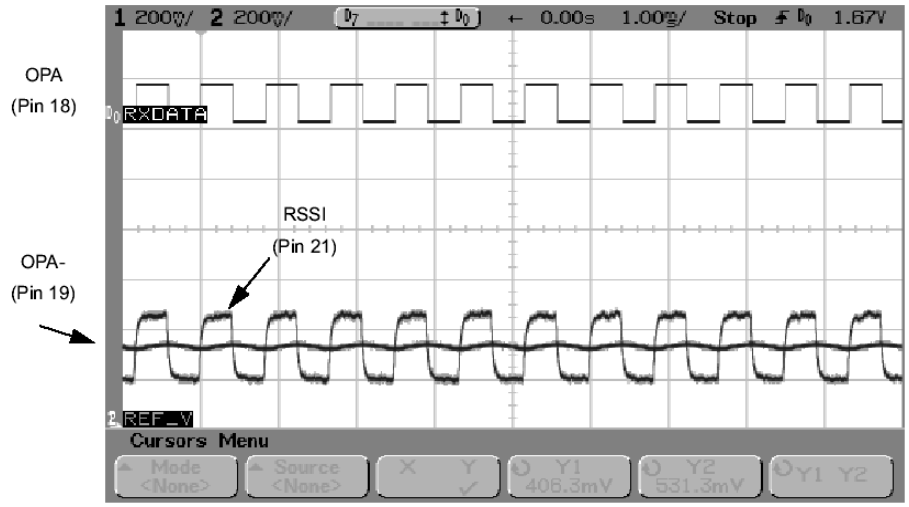

Ilustración 15 Comparación entre RSSI y el voltaje de referencia

1.3.4 Desempeño del rfRXD0420 con señales FSK

1.3.4.1 Consideraciones del filtraje IF

El ancho de banda del filtro IF esta determinado en función de:

- Modulación (ASK,FSK o FM)

- Ancho de banda de la señal
- Tolerancia de temperatura y frecuencia soportada por los elementos transmisores y receptores.

El ancho de banda de las señales FSK es dos veces la desviación de la frecuencia pico mas el ancho de banda de la señal. Por ejemplo si la tasa de transmisión es 2400 bps en codificación Manchester, el ancho de banda de la señal es 4800 baudios ó 1200 Hz, y si la desviación de frecuencia pico es 24 KHz, el mínimo ancho de banda del filtro IF se encontrara luego de sumar a este valor las tolerancias de frecuencia y temperatura de los componentes transmisor y receptor.

$$IF\ BW_{\min} = (2x2400) + (2x24000)$$
$$IF\ BW_{\min} = 52800Hz$$

Las señales FSK son más sensibles a retardos del filtro IF. Entonces, debe ser usado un filtro con mínimas variaciones de retardo. Como una alternativa se puede emplear un filtro con un ancho de banda mayor al requerido ya que las variaciones de retardo en el centro de la banda serán relativamente constantes.

1.3.4.2 Detector FSK

La sección del demodulador (DEMOD) consiste de un detector de fase (Mixer2) y amplificador, creando un detector de cuadratura, también conocido como detector de coincidencia de fase para demodular las señales en aplicaciones con modulación FSK y FM. La señal de fase viene directamente de la salida del amplificador limitador IF al Mixer2.

La señal de cuadratura es creada por un circuito externo a la salida del amplificador limitador IF (pin 15 – $2IF_{OUT}$), acoplado al pin 16, DEM_{IN}.

1.3.4.3 Discriminador LC

El circuito externo puede ser construido con un capacitor y un inductor. Este tipo de circuitos produce una señal de salida excelente. Sin embargo, uno de los elementos, L o C, debe ser sintonizable. La ilustración inferior indica un circuito discriminador usando un condensador variable.

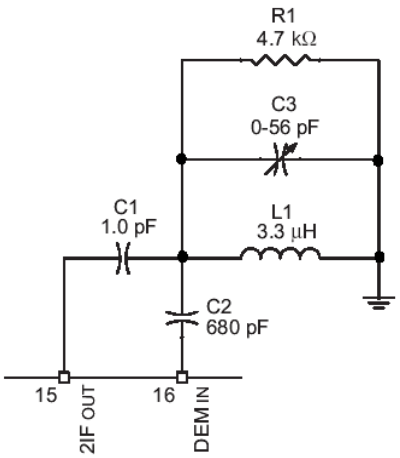

Ilustración 16 Circuito Discriminador LC

1.3.4.4 Discriminador cerámico

También puede ser construida una solución sin variaciones en sus elementos, como se muestra en la ilustración inferior, mediante un discriminador cerámico.

El discriminador cerámico actúa como un circuito sintonizado a la frecuencia IF. El condensador en paralelo, C3, sintoniza el resonador cerámico. El gran desempeño de este circuito permite que a pequeñas variaciones de frecuencia se habilite una gran salida del detector. Sin embargo, las desviaciones de frecuencia que se presentan requieren de mejores tolerancias de frecuencia por parte de los componentes transmisor y receptor.

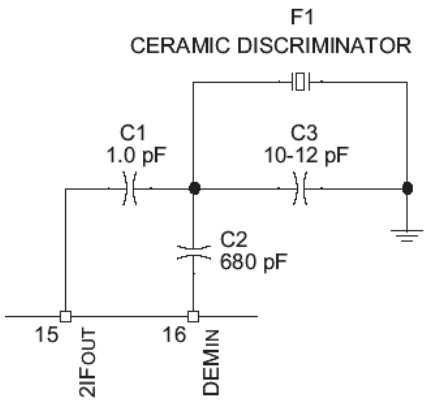

Ilustración 17 Circuito de Discriminador cerámico

1.3.4.5 Filtraje de post detector

Se debe tener cuidado al seleccionar el valor de los condensadores C1 y C2, de tal forma que la salida del detector no se distorsione y la sensibilidad de receptor se incremente. Estos valores son elegidos dependiendo de la tasa de transmisión de la señal.

Generalmente si la tasa de transmisión de la señal es rápida, entonces la constante de tiempo del filtro puede ser pequeña. Por otro lado, si la tasa de transmisión es pequeña, la constante de tiempo del filtro puede ser un valor alto.

1.3.4.6 Comparador

La salida del amplificador DEMOD (DEMOUT+ y DEMOUT-, Pines 23 y 24), depende de la desviación pico de la señal FSK o FM y del circuito de sintonización externo.

Las salidas DEMout+ y DEMout- son salidas de alta impedancia con capacidad de corriente de solo 20 uA. La capacitancia en estos pines limita la tasa máxima de datos. La salida de voltaje nominal de estos pines es 1.2V.

CAPITULO 2

PROTOCOLO DE RED I2C

2.1 Introducción

Los sistemas de comunicación de red, en la actualidad crecen rápidamente tanto en tamaño como en complejidad. Estos sistemas tienen algunos subsistemas integrados de alta velocidad con parámetros de operación críticos y que deben proveer un servicio de confiabilidad extrema. Para mantener el desempeño de estos sistemas, se deben desarrollar sistemas de monitoreo adecuados, de tal forma que una falla que pueda llegar a generar otra falla potencial y más peligrosa, pueda ser identificada rápidamente. Sin embargo, este monitoreo debe ser desarrollado de manera económica para mantener los sistemas con bajos costos.

Para minimizar las fallas del sistema e incrementar la flexibilidad, estos sistemas de red de comunicación deben tener características modulares y de componentes fáciles de conseguir en el mercado. Cada componente en el sistema, contiene típicamente múltiples subsistemas que pueden incluir reguladores DC/DC, microprocesadores de alta velocidad, FPGA's , etc. Algunos de los parámetros de los sistemas de monitoreo incluyen la fuente de poder, el voltaje de salida, la corriente de la fuente de alimentación, la temperatura del elemento, la temperatura ambiente y la velocidad de funcionamiento.

Una red es requerida para que todos los datos que el sensor registra sean recogidos y alimente a la computadora central para monitoreo y análisis. Debido a que algunos sensores son localizados muy cercanos uno de otro, el bus I2C ofrece una solución que puede ser implementada con un costo mínimo en

hardware. Además, los microcontroladores de bajo costo (MCU's) con un amplio rango de elementos periféricos y una interfaz I2C están disponibles fácilmente en el mercado.

Para que el bus I2C sea una solución efectiva para una red de sensores, un protocolo adecuado es requerido, uno que prevenga los errores de sistema del bus que pueda afectar a los datos del sensor. El propósito de este documento es definir un protocolo de red, que sea fácil de adaptarse a la mayoría de las aplicaciones de red. El protocolo del bus debe ser inmune a condiciones adversas de la red, tal como un nodo de red en mal funcionamiento.

2.2 Especificaciones del bus I2C

La interfaz física del bus consiste de dos líneas de drenaje abierto, una para la señal de reloj (SCL) y una línea para datos (SDA).

Las líneas SDA y SCL son puestas en alto mediante las resistencias conectadas a la línea de V_{DD}. El bus debe tener una configuración maestro y algunas configuraciones esclavo, o puede tener múltiples elementos maestros. El elemento maestro es responsable de generar la señal de reloj para los elementos esclavos.

El protocolo I2C soporta incluso un modo de direccionamiento de 7 o 10 bits, permitiendo 128 0 1024 elementos físicos conectados al bus, respectivamente. En la práctica, las especificaciones del bus reservan ciertas direcciones, así que pocas direcciones quedan disponibles.

Por ejemplo, el modo de direccionamiento de 7 bits permite el uso de 112 direcciones. Todos los datos transferidos en el bus son inicializados por el elemento maestro y se convierten en ocho bits en ese instante, siendo el primero el bit más significativo (MSB).

No existe límite para la cantidad de datos que pueden ser enviados en una sola transferencia. El protocolo I2C incluye un mecanismo de handshaking.

Luego de la transferencia de 8 bits, un noveno pulso de reloj es enviado por el maestro. En este momento el elemento transmisor en el bus libera la línea SDA y el elemento receptor en el bus reconoce el envío del dato por parte del transmisor. La señal ACK (SDA se mantiene en bajo) es enviada en caso de que el mensaje no sea recibido satisfactoriamente. Una señal NACK también es usada para finalizar la transmisión de datos luego de que el último byte sea recibido. De acuerdo a las especificaciones del bus I2C, todos los cambios registrados en la línea SDA deben ocurrir mientras el estado de la línea SCL sea bajo. Esta restricción permite que solo dos condiciones únicas sean detectadas en el bus: una secuencia de START (S) y una secuencia de STOP (P). La secuencia de START ocurre cuando el elemento maestro presiona para que la señal de SDA se mantenga en bajo, mientras la línea SCL se mantiene en alto. La secuencia de START notifica a todos los elementos esclavos del bus, que los bytes de direcciones están a punto de ser transmitidos. La secuencia de STOP ocurre cuando la línea SDA esta en alto, mientras la línea SCL se mantenga en alto y ésta termine la transmisión.

Los elementos esclavos del bus podrían resetear su lógica de recepción luego de que la secuencia de STOP haya sido detectada.

El protocolo I2C también permite una condición de START repetida (RESTART - R_S), la cual permite que el elemento maestro ejecute la secuencia de START sin que sea precedida de una secuencia de STOP. El RESTART es útil, por ejemplo, cuando el elemento maestro cambia de una operación de escritura a una operación de lectura y no libera el control del bus.

Una transmisión I2C típica de escritura puede proceder de la forma en la que se muestra en la Ilustración 18. En este ejemplo el elemento maestro escribirá dos bytes a un elemento esclavo. La transmisión es iniciada cuando el maestro inicia una condición de START en el bus. Luego el maestro envía un

byte de dirección al esclavo. Los siete dígitos más altos contienen la dirección del esclavo. Los bits menos significativos (LSB) del byte de dirección especifican la operación del I2C, tanto para lectura (LSB=1) o escritura (LSB=0).

En el noveno pulso de reloj, el maestro libera la línea SDA, así el esclavo puede conocer acerca de la recepción. Si el byte de dirección fue recibido por el esclavo y era la dirección correcta, el esclavo responde con una señal ACK, manteniendo la línea SDA en bajo. Asumiendo que ACK fue recibida, el maestro envía los bytes de datos.

En el noveno pulso de reloj luego de cada byte de datos, el esclavo responde con un ACK. Luego del último byte de datos, una señal de NACK es enviada por el esclavo al maestro para indicar que no existen mas bytes para ser transmitidos.

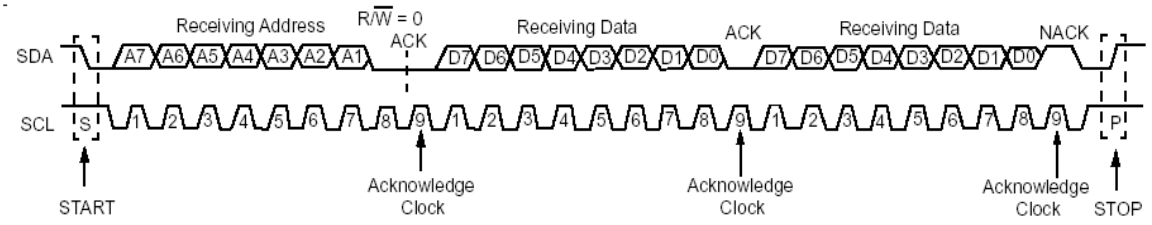

Ilustración 18 Típica transmisión de escritura I2C (dirección 7 bits)

Luego del pulso de NACK, el maestro inicia la condición de STOP para liberar el bus.

Una operación de lectura es efectuada en forma similar a la operación de escritura, y esta es mostrada en la Ilustración 19. En este caso, el bit R/W en el byte de dirección es establecido para indicar la operación de lectura.

Luego de que el byte de dirección sea recibido, el elemento esclavo envía un pulso ACK y mantiene la línea SCL en bajo. Al mantener la línea SCL, el esclavo puede tomar todo el tiempo que necesite para preparar el dato a ser transmitido de regreso al maestro. Cuando el esclavo esta listo, este libera SCL y el elemento maestro registra el dato que viene del buffer del esclavo.

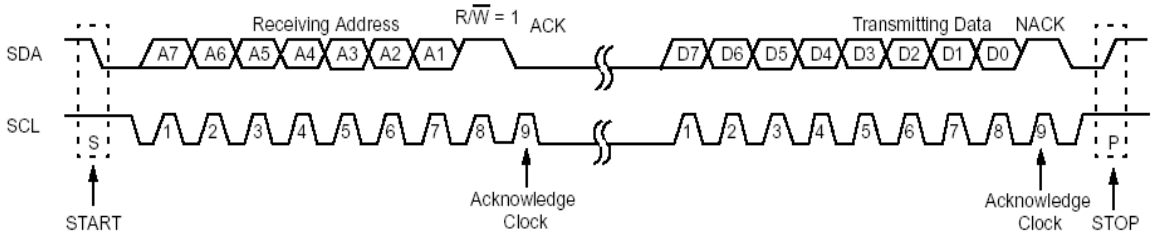

Ilustración 19 Típica transmisión de lectura I2C (dirección 7 bits)

En el noveno pulso de reloj, el esclavo libera la línea SDA y guarda el valor del bit ACK recibido del maestro. Si el pulso ACK fue recibido, el esclavo debe preparar el siguiente byte de datos a ser transmitido. Si un NACK fue recibido, la transmisión del dato es completa. En este caso, el esclavo resetea su nivel lógico de recepción I2C y espera a la siguiente condición de START.

Para muchos elementos periféricos I2C, tal como una memoria EEPROM no volátil, una operación de escritura y de lectura I2C son hechas en sucesión. Por ejemplo, la operación de escritura especifica la dirección a ser leída y la operación de lectura obtiene el byte de datos. Si el elemento maestro no libera el bus luego de que la dirección de memoria es escrita en el elemento, una secuencia repetida de START es generada para leer el contenido de la dirección de memoria.

2.3 Cálculos de diseño para el bus I2C

Cuando se diseña una red I2C, el numero de elementos en el bus, las características físicas del cableado del bus y la longitud del bus debe ser considerada. Estas variables determinan la cantidad total de carga capacitiva en el bus, que las especificaciones del bus I2C limitan a 400 pF.

Si las características eléctricas del cableado usado para el bus I2C son conocidas, entonces es fácil determinar la capacitancia total del bus. Todo lo que se necesita es determinar la contribución de capacitancia de cada elemento en el bus. Si la capacitancia de cada elemento no es conocida, entonces se estimara un valor de 10pF para cada uno de ellos.

Otra forma de encontrar la capacitancia total del bus es tomar valores preliminares para las resistencias de pull up y analizar el tiempo en el que actúan en el bus, mediante el uso de un osciloscopio.

Para la mayoría de aplicaciones, 2000Ω seria un buen valor de inicio para las resistencias de pull up. El tiempo de acción es el tiempo que la señal toma para ir desde 10% a un 90% del valor final. Entonces la capacitancia total del bus puede ser determinada usando la Ecuación 4.

$$C_{BUS} = \frac{t_R}{2.2 \cdot R}$$

Ecuación 4 Calculo de la Capacitancia del Bus

Luego, las especificaciones para el tiempo de acción del bus I2C deben ser conocidas, el cual depende de la frecuencia del bus. Para modos de alta velocidad (400KHz), el tiempo de acción máximo es 300ns. Para modos estándar (100KHz), el tiempo de acción máximo es 1μs. La Ecuación 4 puede ser modificada para encontrar el valor requerido de resistencias de pull up como se muestra en la Ecuación 5.

$$R_{PULL-UP} = \frac{t_R}{2.2 \cdot C_{BUS}}$$

Ecuación 5 Calculo de la resistencia de pull up

Las especificaciones I2C limitan la cantidad de corriente en el bus a 3mA, que indirectamente coloca un limite en el valor de las resistencias de pull up. Así, para un bus de 5V, la resistencia de pull up mínima que puede ser usada es 5V/3mA o aproximadamente 1600Ω.

2.3.1 Ejemplo de Calculo de diseño

Como un diseño ejemplo, las características del cable que fue usado para probar la aplicaron, será utilizado para los siguientes cálculos. Un cable de longitud de 24 pies fue usado para conectar dos PIC's 16F877 con resistencias de pull up de 200Ω en las líneas SDA y SCL.

La línea SCL fue observada con un osciloscopio y el tiempo de acción fue determinado con un valor de 464ns. La capacitancia del cable, por pie, es calculada en el Ejemplo 1.

$$C_{CABLE} = \frac{464ns}{(2.2)(200\Omega)(24\,pies)} = 44\,pF / pie$$

Ejemplo 1 Calculo de la capacitancia del alambre

La longitud máxima del bus que puede ser usada con este cable, sin extensiones, es calculada en el Ejemplo 2.

$$L_{MAX} = \frac{400\,pF}{44\,pF / pie} = 9.1 pies$$

Ejemplo 2 Calculo de la longitud máxima del bus

Note que el calculo de esta longitud también excluye los efectos de la capacitancia del elemento y puede ser reducida un poco en la práctica.

Para mas cálculos, se debe asumir que la longitud del bus esta especificada como 3 pies. Usando el cable elegido para este ejemplo, se puede establecer la capacitancia del bus como 3 x 44pF/pulgada o 132 pF. Ahora se debe elegir la frecuencia del bus, la cual es arbitrariamente seleccionada con un valor de 100 KHz. Usando el tiempo de acción máximo para una frecuencia de bus de 100 KHz, el valor de las resistencias de pull up es calculada en el Ejemplo 3.

$$R_{PULL-UP} = \frac{1\mu s}{(2.2)(132\,pF)} = 3400\Omega$$

Ejemplo 3 Calculo de la resistencia de pull up

Una resistencia de pull up con un valor de 3400Ω, podrá proveer aproximadamente 1.5 mA al bus, valor que no viola el máximo de corriente establecida.

La Tabla 4 muestra la longitud máxima del bus en base a la frecuencia del bus, la corriente límite del bus y las características del cable. Sin embargo se necesita calcular la longitud máxima del bus para la aplicación específica, esta tabla de datos proporcionará una idea aproximada de lo que se desea lograr.

Capacitancia del Bus = 44 pF/pie	Resistencia de Pull up	Frecuencia del Bus = 100 KHz	Frecuencia del Bus = 400 KHz
		Máxima longitud del Bus	Máxima longitud el Bus
Sin extensión de bus	1600Ω	6 pies	1.8 pies
Con circuito para extensión IC	160Ω	60 pies	18 pies

Nota: la longitud del bus esta limitada por la elección de los valores de resistencia de pull up, que no deben exceder el máximo de corriente que puede circular por el bus I2C de acuerdo a las especificaciones.

Tabla 4 Datos para las máximas longitudes del bus

2.4 Fallas de hardware

En un ambiente de monitoreo distribuido, los elementos esclavos pueden ser cambiados en el bus para reemplazar sistemas con fallas, para mantenimiento regular o incluso para pruebas.

El hardware de la aplicación puede variar dependiendo de los requerimientos del sistema, pero ciertos aspectos de hardware pueden ser implementados en cada sistema para asegurar que los errores mínimos son introducidos en el bus I2C, cuando un nuevo elemento es insertado o removido. El conector de hardware elegido debe proveer la energía necesaria al sistema.

Así como un nodo esclavo es conectado al bus I2C, la primera conexión física realizada debe ser la conexión a tierra, de tal forma que cualquier tipo de energía se descarga en el sistema de tierra, la segunda conexión puede ser la energía al nodo esclavo.

La cantidad de capacitancia total en la fuente de energía debe ser considerada y se deben instalar resistencias limitadoras de corriente. Las líneas SDA y SCL deben ser la última conexión a realizar en el sistema, es una buena idea el colocar resistencias pequeñas en serie en las líneas SDA y SCL. Estas resistencias limitarán la cantidad de corriente que puede circular a través de los pines de entrada/salida durante el encendido. La Ilustración 20 muestra un diagrama de bloque de la conexión física del bus.

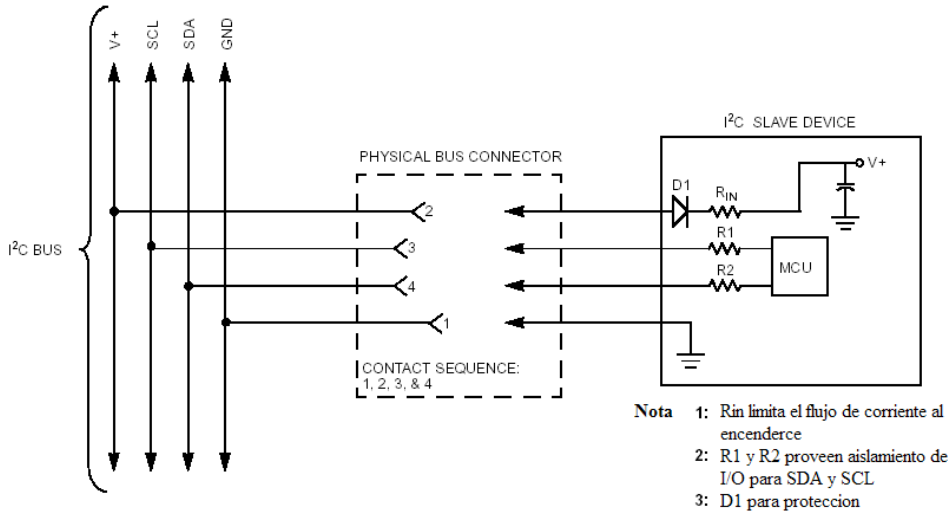

Ilustración 20 Diagrama de Bloques de la Conexión física del Bus I2C

2.5 Reloj de Tiempo Real

2.5.1 Características.

- El reloj de tiempo real (RTC) cuenta segundos, minutos, horas, días del mes, meses, días de la semana y años que puede llegar hasta el 2100.
- 56 bytes de NVRAM para almacenamiento de datos.
- Dos cables de interfaz serial.
- Señal de salida de onda cuadrada programable.
- Detección automática de falla de energía.
- Rango de temperatura de –40°C a +85°C
- Disponible en encapsulado DIP de 8 pines

2.5.2 Descripción de los pines

PIN	Descripción
Vcc	Fuente de poder primaria
X1,X2	Conexión de cristal de 32.768 KHz.

V_{BAT}	Voltaje de entrada de la batería de +3V
GND	Tierra
SDA	Datos Serial
SCL	Reloj Serial
SQW/OUT	Salida de onda cuadrada.

Tabla 5 Descripción de los pines del RTC

2.5.3 Descripción

El Reloj de Tiempo Real es un reloj/calendario, codificado en binario y decimal (BCD), que dispone de 56 bytes de NVRAM. Las direcciones y los datos son transferidos serialmente por medio de dos cables en un bus bidireccional.

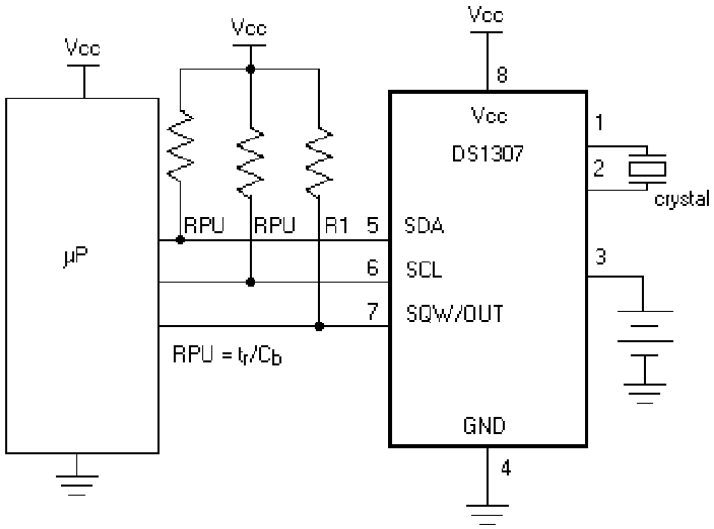

Ilustración 21 Circuito de Operación típico

El reloj/calendario provee información de segundos, minutos, horas, días, fechas, meses y años. La fecha de fin de mes es ajustada automáticamente para los meses que tienen menos de 31 días incluyendo las correcciones para los años bisiestos.

El reloj opera en formato de 24 horas o 12 horas con un indicador de AM/PM. Además el dispositivo detecta fallas de la fuente de alimentación y automáticamente conmuta con la batería.

2.5.4 Operación

El RTC (Real Time Clock) opera como un elemento esclavo en un bus serial. El acceso se lo obtiene mediante la implementación de una condición de START y estableciendo un código de identificación del elemento seguido de la dirección de registro.

Los registros subsecuentes pueden ser accedidos secuencialmente hasta que una condición de STOP sea ejecutada. Cuando el voltaje V_{CC} cae por debajo de un valor de 1.25 x V_{BAT}, el elemento termina un acceso en proceso y resetea el contador de dirección del mismo.

Las entradas al dispositivo no serán reconocidas en este momento para prevenir que datos erróneos sean escritos en el elemento. Cuando V_{CC} cae por debajo de V_{BAT}, el elemento conmuta a un modo de baja corriente.

El dispositivo conmuta de la batería a V_{CC} cuando V_{CC} tiene un valor mayor a V_{BAT} + 0.2V y reconoce las entradas cuando V_{CC} es mayor que 1.25 x V_{BAT}. El diagrama de bloques de la Ilustración 21 muestra los elementos principales del RTC serial.

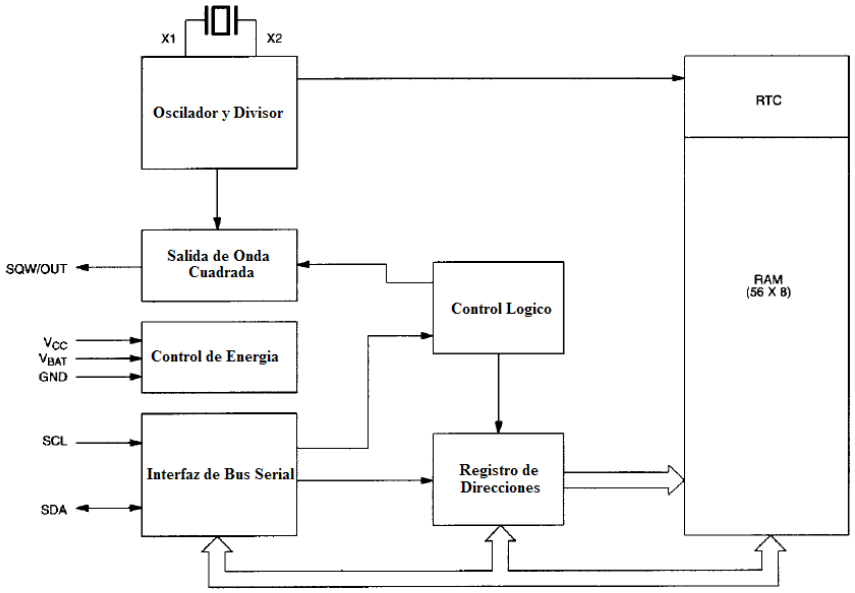

Ilustración 22 Diagrama de Bloques del Integrado

2.5.5 Descripción de las señales

V_{CC}, GND._ La energía de DC se provee al elemento en estos pines. V_{CC} es la entrada correspondiente a +5V. Cuando se aplican 5V en sus limites normales, el elemento es totalmente accesible y los datos pueden ser escritos y leídos. Cuando una batería de 3V se conecta al dispositivo y V_{CC} esta por debajo de 1.25 x V_{BAT}, las lecturas y escrituras están restringidas.

V_{BAT}._ Es la entrada de batería de litio, que corresponde a un valor estándar de 3V. El voltaje de la batería debe mantenerse entre 2.0V y 3.5V para una operación adecuada. El voltaje de protección de escritura a la cual el acceso al RTC y la RAM están negadas esta establecido por la circuitería interna como 1.25 x V_{BAT}.

SCL (Serial Clock Input)._ SCL es usado para sincronizar el movimiento de los datos en la interfaz serial.

SDA (Serial Data Input/Output)._ SDA es el pin de entrada/salida para los dos cables de interfaz serial. El pin SDA es de drenaje abierto por lo que requiere de una resistencia externa de pull up.

SQW/OUT (Square Wave/Output Driver)._ Cuando esta habilitado, el bit SQWE es colocado en 1, el pin SQW/OUT es la salida de frecuencias de onda cuadrada de 1Hz, 4KHz, 8KHz, 32KHz. Este pin también es de drenaje abierto por lo que necesita una resistencia externa de pull up. SQW/OUT. SQW/OUT puede operar tanto con V_{CC} como con el voltaje de la batería.

X1, X2._ Son las conexiones para un cristal de cuarzo de 32.768KHz. La circuitería del oscilador interno esta diseñada par operar con un cristal, teniendo una capacitancia de carga especifica de 12.5 pF.

El RTC también puede ser manejado por un oscilador externo de 32.768KHz, en esta configuración el pin X1 esta conectado a la señal del oscilador externo y el pin X2 es flotante.

2.5.6 Exactitud del Reloj

La exactitud del reloj depende de la exactitud del cristal y de la compatibilidad entre la carga capacitiva del oscilador y la carga capacitiva del cristal elegido. Un error adicional puede presentarse en la frecuencia del cristal causada por los cambios violentos de temperatura.

2.5.7 Mapa de direcciones de RTC y RAM

El mapa de direcciones de los registros del RTC y de la RAM es mostrado en la Ilustración 23. Los registros RTC están localizados en las direcciones de memoria 00h a la 07h. Los registros de la RAM están localizados en las direcciones de memoria 08h a la 3Fh. Durante un acceso de múltiples bytes,

cuando el puntero de direcciones alcanza la 3Fh, el ultimo espacio de la RAM, esta salta a la localidad 00h, el inicio de los registros del reloj.

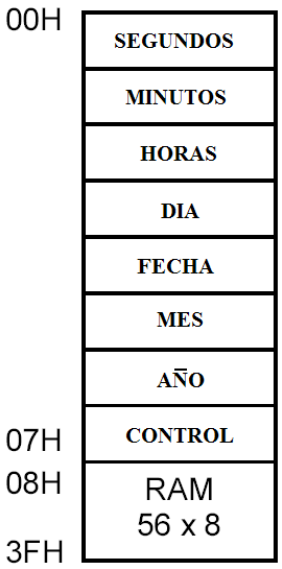

Ilustración 23 Mapa de Direcciones

2.5.8 Reloj y calendario

La información de tiempo y de calendario es obtenida mediante la apropiada lectura de los bytes de los registros. Los registros del RTC son mostrados en la Ilustración 24. El tiempo y el calendario son establecidos o inicializados escribiendo los bytes de los registros apropiados. El contenido de los registros de tiempo y calendario están en formato BCD.

El bit 7 del registro 0 es el bit de habilitación de reloj (CH). Cuando el bit esta en 1, el oscilador es deshabilitado. Cuando el bit esta en 0, el oscilador es habilitado.

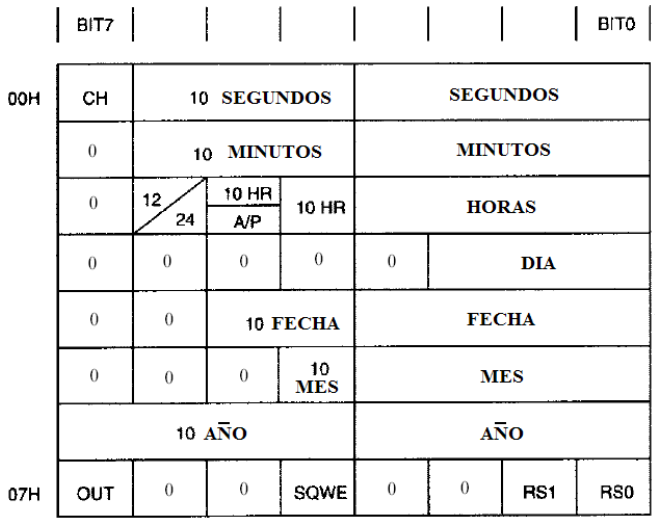

Ilustración 24 Registros del RTC

Debe notarse que el estado inicial de los registros no esta definido, por lo cual es importante habilitar el oscilador (bit CH=0) durante la configuración inicial.

El RTC puede correr tanto en modo de 24 horas como en el de 12 horas. El bit 6 del registro de horas es definido como el bit selector de modo de presentación de 12 o 24 horas. Cuando esta en alto, se selecciona el modo de 12 horas. En este modo, el bit 5 es el bit de AM/PM, cuando esta en alto indica formato PM. En el modo de 24 horas, el bit 5 es el segundo bit de las siguientes 10 horas (20 – 23 horas)

En el inicio, la hora actual es transferida a unos registros secundarios. La información es leída de estos registros mientras el reloj continúa funcionando. Esto elimina la necesidad de volver a leer los registros en caso de actualización de los registros principales durante su lectura.

2.5.9 Registro de control

Este registro es empleado para controlar la operación del pin SQW/OUT.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

OUT	0	0	SQWE	0	0	RS1	RS0

OUT (Output control): Este bit controla el nivel de salida del pin SQW/OUT cuando la salida de onda cuadrada esta deshabilitada.

SQWE (Square Wave Enable): Este bit habilita la salida del oscilador si se coloca en '1' lógico. La frecuencia de salida de la onda cuadrada depende del valor de los bits RS0 y RS1. Con la onda cuadrada de salida establecida en 1Hz, los registros del reloj se actualizan en la zona que la onda baja.

RS (Rate Select): Estos bits controlan la frecuencia de salida de la onda cuadrada cuando esta ha sido habilitada. La Tabla 6 indica las frecuencias que pueden ser seleccionadas con estos bits.

RS1	RS2	Frecuencia de Salida SQW
0	0	1 Hz
0	1	4.096 KHz
1	0	8.192 KHz
1	1	32.768 KHz

Tabla 6 Frecuencias de Salida de onda cuadrada

2.6 Bus de datos serial de 2 cables

El RTC soporta un bus bidireccional de dos cables y un protocolo de transmisión de datos. Un elemento que envía datos al bus es definido como transmisor y el elemento que lo recibe se conoce como receptor. El maestro es aquel que controla el mensaje y los controlados son los llamados esclavos. El bus debe ser controlado por un elemento maestro que es el encargado de generar la señal de reloj (SCL), de controlar el acceso al bus y generar las condiciones de START y STOP.

El RTC opera como un miembro esclavo en el bus de dos cables. Una configuración de bus usando un protocolo de dos cables se muestra en la Ilustración 25.

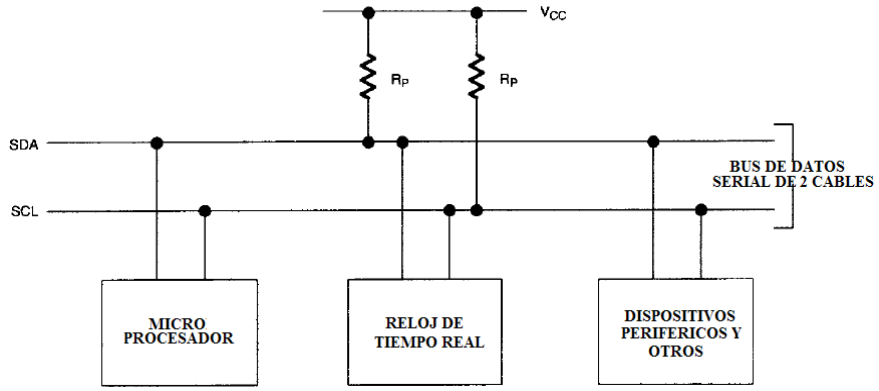

Ilustración 25 Configuración típica del bus de dos cables

No se debe olvidar las siguientes características:

- La transferencia de datos podría ser inicializada solo cuando el bus no esta ocupado.

- Durante la transmisión de datos, la línea de datos debe mantenerse estable siempre que la línea de reloj este en alto.

Los cambios en la línea de datos mientras la línea de reloj esta en alto, pueden ser interpretados como señales de control.

En concordancia, las siguientes condiciones del bus han sido definidas:

Bus no ocupado (Bus not busy): tanto la línea de datos y de reloj se mantienen en alto.
Inicio transferencia de datos (Start data transfer): Un cambio en el estado de la línea de datos, de alto a bajo, mientras el reloj esta en alto, define una condición de START.

Detener transferencia de datos (Stop data transfer): Un cambio en el estado de la línea de datos de bajo a alto, mientras el reloj esta en alto, define una condición de STOP.

Datos Validos (Data valid): El estado de la línea de datos representa datos validos cuando, luego de una condición de START, la línea de datos es estable por el periodo de duración en alto de la señal de reloj. Los datos en la línea deben ser cambiadas durante el periodo en bajo de la señal de reloj. Existe un pulso de reloj por cada bit de datos. Cada transferencia de datos es iniciada con una condición de START y termina con una condición de STOP.

El numero de bytes de datos transferidos entre las condiciones de START y STOP no es limitado, y es determinado por el elemento maestro. La información es transferida por bytes y el receptor indica que se recibió el mensaje (Acknowledge), mediante un noveno bit. En las especificaciones del bus, se indican dos modos de transferencia: regular (100KHz) y rápido (400 KHz).

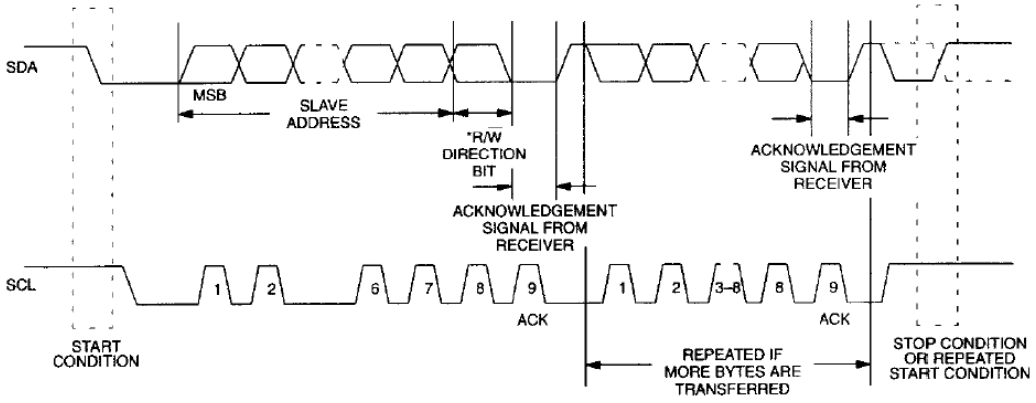

Ilustración 26 Transferencia de datos en el Bus serial de 2 cables

Dependiendo del estado del bit R/W, existen dos tipos de transferencia posible:

Transferencia de datos de un transmisor maestro a un receptor esclavo._ El primer byte transmitido por el maestro es la dirección del esclavo. Luego le sigue el numero de bytes de datos. El esclavo retorna un bit de ACK luego de cada byte recibido. En el dato, primero se transfiere el MSB.

Transferencia de datos de un transmisor esclavo a un receptor maestro._ El primer byte, que corresponde a la dirección del esclavo, es transmitida por el maestro. Luego el esclavo retorna un bit de ACK. Este bit es seguido de la transmisión del esclavo con el numero de bytes de datos. El maestro retorna un bit de ACK luego de recibir todos los bytes anteriores al ultimo byte. Al final del ultimo byte recibido, una señal de 'not acknowledge' es retornada.

El elemento maestro genera todos los pulsos de reloj y las condiciones de START y STOP. La transferencia se termina con una condición de STOP o con una condición repetida de START. Desde que la señal repetida de START es también el inicio de la siguiente transferencia, el bus no será liberado. Los datos son transmitidos con el MSB primero.

El RTC puede operar en los siguientes dos modos:

Modo de esclavo receptor (RTC modo de escritura)._ Los datos y la señal de reloj son transmitidos a través de las líneas SDA y SCL. Luego de que cada byte es recibido un bit de ACK es transmitido. Las condiciones de START y STOP son reconocidas como el inicio y fin de una transmisión serial. El reconocimiento de dirección es realizado por el hardware luego de la recepción de la dirección del esclavo y del bit de acción (R/W) (Ilustración 27). El byte de dirección es el primero en ser recibido luego de que la condición de START es generada por el maestro. El byte de dirección contiene una dirección de 7 bits, el cual es 1101000, seguido por bit de acción (R/W), el cual para escritura tiene un valor de '0'. Luego de recibir y decodificar la dirección el elemento envía la señal de ACK por la línea SDA.

El maestro empezará a transmitir cada byte de datos con el RTC dando a conocer la recepción de cada byte. El maestro podrá generar una condición de stop para terminar la escritura de datos.

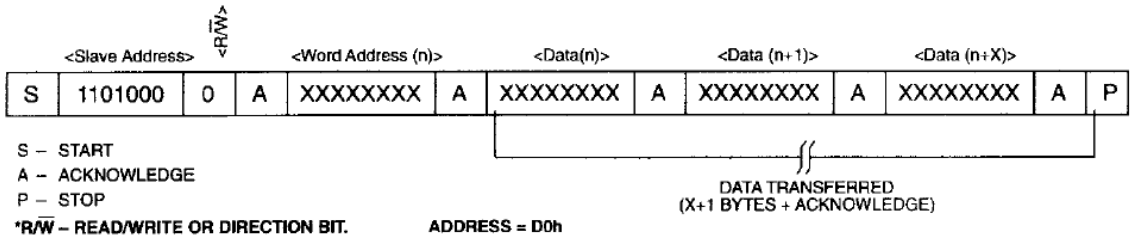

Ilustración 27 Escritura de Datos en el modo de Esclavo receptor

Modo de esclavo transmisor (RTC modo de lectura)._ El primer byte es recibido y mantenido como en el modo de esclavo receptor. Sin embargo, en este modo, el bit de acción indicara que la dirección de transmisión es invertida. Los datos son transmitidos en la línea SDA por el RTC, mientras la señal de reloj ingresa por la línea SCL. Las condiciones de START y STOP son reconocidas como el inicio y el fin de una transferencia serial (Ilustración 28). El byte de dirección es el primer byte recibido luego de la condición de START generada por el maestro. El byte de dirección contiene una dirección de 7 bytes, que es 1101000, seguida del bit de acción (R/W), el cual para lectura tienen un valor de '1'. Luego de recibir y decodificar el byte de dirección el dispositivo ingresa una señal de ACK a la línea SDA. El RTC, entonces, empieza a transmitir datos empezando por la dirección del registro al que señala el puntero de registros.

El RTC debe recibir una señal de NACK para finalizar la lectura.

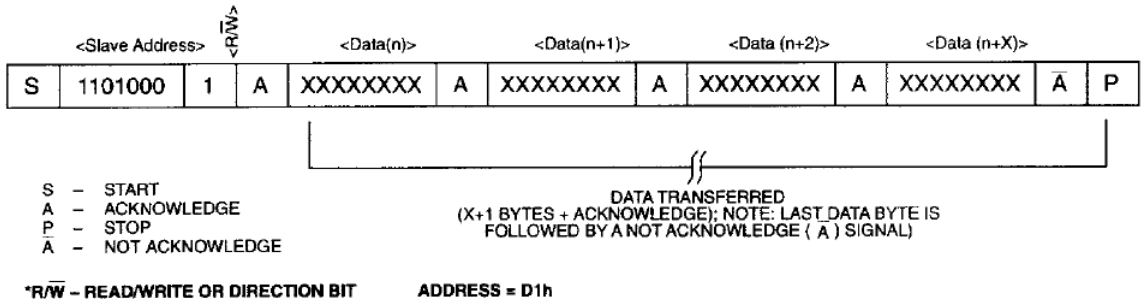

Ilustración 28 Lectura de Datos en modo de Esclavo Transmisor

CAPITULO 3

ESQUEMA ELECTRONICO

3.1 Descripción de los módulos periféricos

Para el correcto desempeño de los módulos periféricos del Sistema SDP877, son necesarias algunas conexiones básicas.

3.1.2 Fuente de alimentación

A fin de energizar a todos los elementos de los módulos periféricos, es necesaria una alimentación de 5 Voltios de corriente continua. Esta fuente externa es la correspondiente a la del Sistema SDP877, cuya conexión se comparte con los módulos periféricos, con ello se evita el uso de otra fuente, dando mayor flexibilidad y facilidad de uso al sistema completo.

Como ya es conocido los pines de alimentación y de tierra del microcontrolador PIC16F877 son los 11 y 32, correspondientes a Vcc, además de los pines 12 y 31 correspondientes a tierra; estas conexiones son activadas mediante las borneras o las salidas de la regleta.

Estos voltajes dentro del módulo periférico se encargarán de:

a) Voltaje de alimentación y de tierra de los microcontroladores PIC16F877.
b) Funcionamiento óptimo de las señales de entrada analógicas.

c) Activación del integrado DS1307, conocido como RTC o Reloj de Tiempo Real.

d) Activación de la matriz de leds.

Configuración de puertos y esquema electrónico de los PIC's en el módulo periférico.

En el módulo periférico que dispone de dos microcontroladores PIC16F877, cuyos puertos están debidamente configurados para ejecutar diferentes acciones.

El primer PIC corresponde al que maneja la red I2C para presentar en el LCD del Sistema SDP877 los valores de los potenciómetros, conectados a las entradas análogas del PIC. Estas entradas análogas corresponden a señales que son simuladas mediante potenciómetros.

3.1.2 Configuración de los Puertos

En este PIC se configuraron los puertos de la siguiente forma:

PUERTO A: Se configura como entradas análogas, que serán conectadas a los potenciómetros (RA0-RA4). El puerto RA5 se configura como salida.

PUERTO B: Configurado como entrada (RB0) y como salidas los puertos RB1-RB7.

PUERTO C: Se establece como salidas, los pines correspondientes a la transmisión y recepción serial, se configuraran como el usuario desee.

PUERTO D y PUERTO E: Se configuran como puertos de salida.

Todos los pines correspondientes a puertos están conectados a la regleta, desde la cual se podrá realizar cualquier conexión necesaria para la implementación de nuevas aplicaciones.

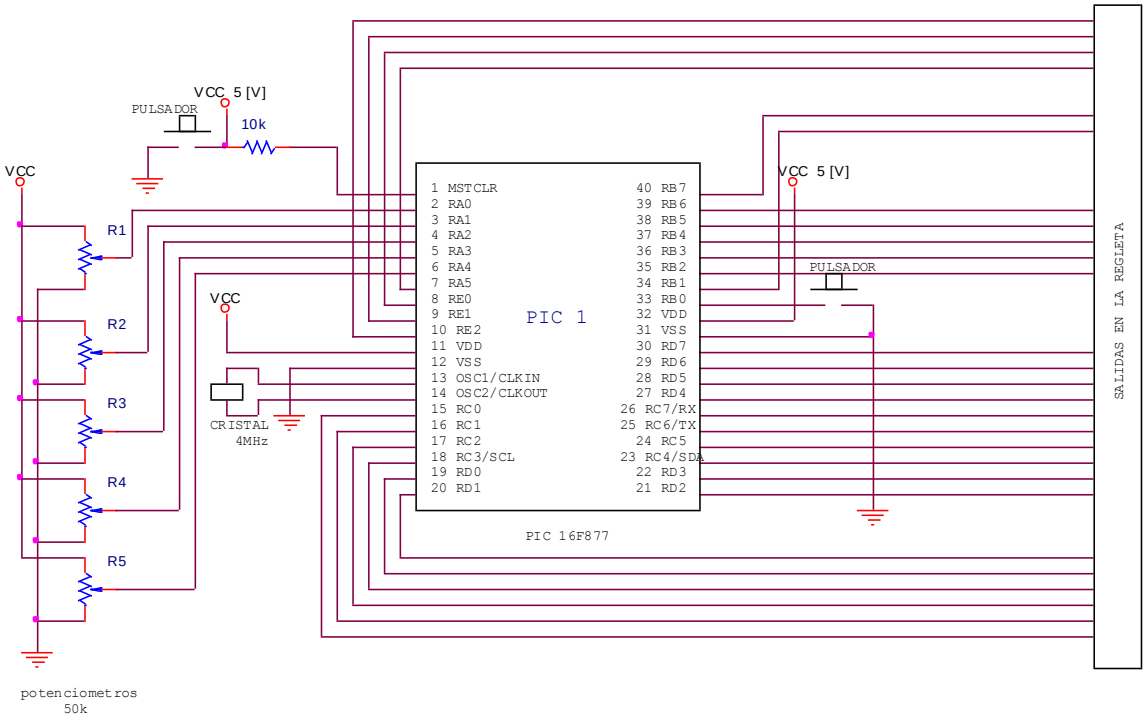

Ilustración 29 Visualización General de la configuración del PIC 1

El segundo PIC es el encargado del manejo de las matrices de leds, para presentar la información que se envía al bus I2C desde el Sistema SDP877. En este elemento los puertos se configuraron de la siguiente manera:

PUERTO A: Esta configurado como salidas, este es el encargado de enviar 0 o 1 lógicos a fin de activar o desactivar las columnas correspondientes a la matriz 2. mediante esta activación o desactivación se genera la señal adecuada para el encendido de las letras, ya sean mayúsculas o minúsculas para la correcta visualización en las matrices de leds. El puerto RA5 no esta conectado.

PUERTO B: Configurado como salidas, es el encargado de activar/desactivar las filas de las matrices de leds, realiza el control de encendido de las mismas. Este proceso consiste de un barrido de las distintas filas mediante software, con ello se activa o no la fila correspondiente para la presentación del texto. El puerto RB0 se establece como entrada, conectado a un potenciómetro.

PUERTO C: Se configuran los pines RC0-RC2 y RC5 como salidas, los pines de transmisión y recepción serial se configuran según el usuario decida. Los pines de señales de datos y reloj (RC4-SDA y RC3-SCL) del bus I2C se configuran como entradas. Estas dos líneas se conectan con sus iguales tanto en el PIC 1 como en el RTC.

PUERTO D: Todos los pines se establecen como salidas, al igual que el puerto A, se encargan del control de las columnas de la matriz 1.

PUERTO E: Sus pines se configuran como salidas, su función es el encendido/apagado de las columnas de las matrices.

3.2 Diagramas eléctricos.

Para observar los diagramas eléctricos del Módulo MP-SDP877 y de las tarjetas de transmisión y recepción del rfPIC, se pide dirigirse a los ANEXO 1.

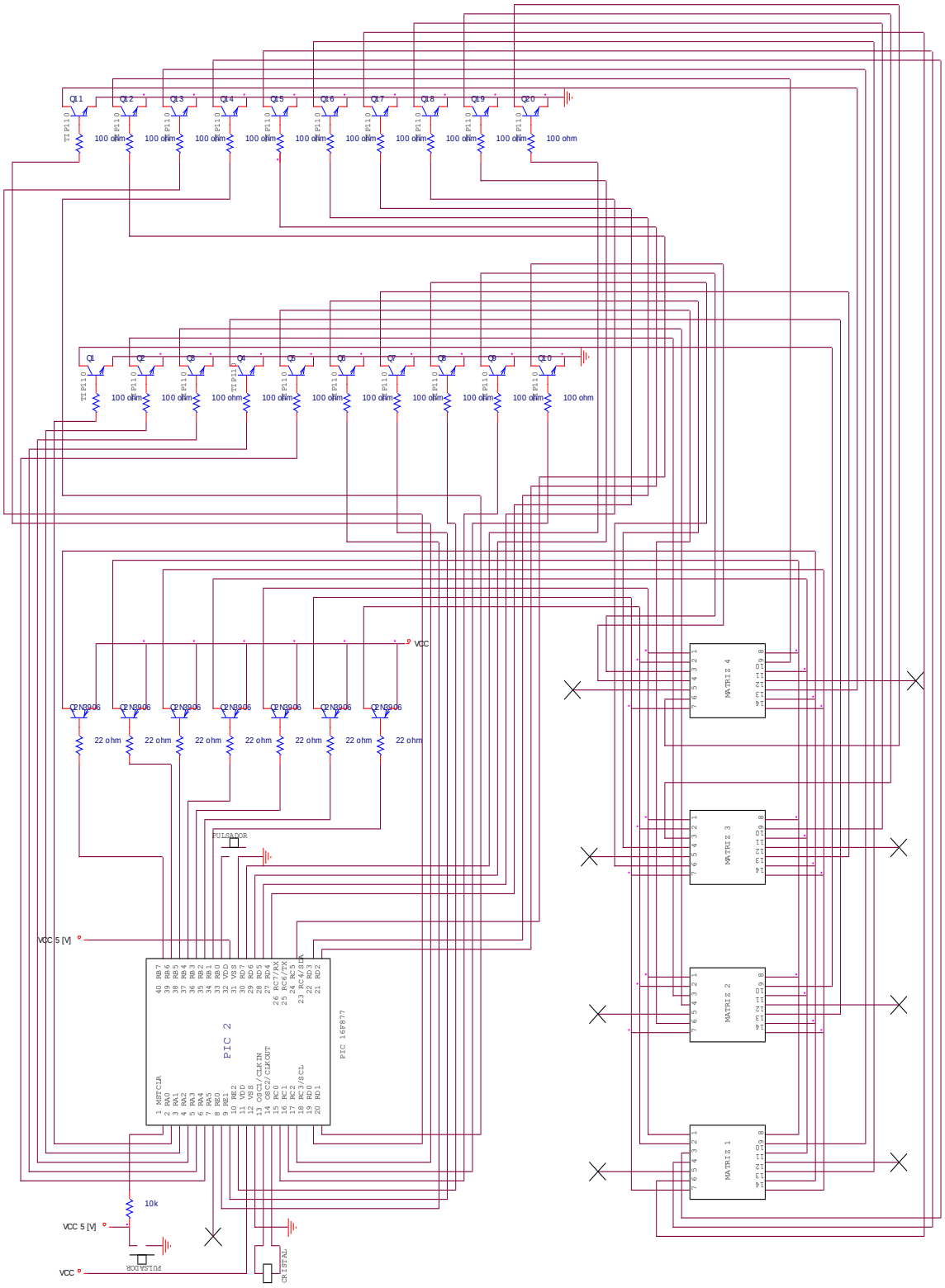

Ilustración 30 Visualización General de la configuración del PIC 2

CAPITULO 4

PRÁCTICAS Y APLICACIONES PARA EL MÓDULO PERIFERICO

4.1 Introducción

En el presente capitulo se brindara información explicita de las prácticas que se realizan con estos módulos periféricos, las explicaciones incluirán hardware y software, de tal forma que el lector pueda tener una idea mas clara de ciertas aplicaciones que son de interés común. Además podrán plantearse futuras prácticas que se recomienda realizarlas para una mejor comprensión de los temas.

Las explicaciones de software se lo realizara de la manera más sencilla y didáctica posible, presentando diagramas de flujos, además se proveerá gráficos de las señales de las distintas aplicaciones, obtenidas gracias al osciloscopio digital.

4.2 Comunicación mediante la red I2C

4.2.1 Objetivos

La finalidad de esta práctica es brindar al estudiante los conocimientos necesarios para el uso e implementación de la red de comunicación I2C. Esto se lo realizara mediante diferentes prácticas que incluirán comunicación entre PIC's y otros elementos tales como RTC.

4.2.2 Marco Teórico

La red de comunicación I2C fue desarrollada por Phillips Semiconductor y consiste básicamente en una forma de comunicación serial sincrónica. Para su operación se dispone únicamente de dos líneas de comunicación: la línea de datos, conocida como SDA y la línea de reloj, llamada también SCL.

La topología generalmente empleada en este tipo de redes de comunicación es la de anillo, en la que se da el permiso de transmitir datos a un terminal a la vez, cuando el dato haya sido transmitido al bus, el elemento esclavo libera la posta de transmisión y esta pasa al siguiente esclavo permitiéndole transmitir. Se dispone de un elemento maestro que registra las direcciones de los elementos esclavos y de acuerdo a esa dirección envía o recibe la información adecuada.

Cada byte que se encuentra en la línea de datos esta compuesto de 8 bits, pero el numero de bytes que se transmite en cada transferencia es indefinido, tanto la línea SDA como la SCL son líneas bidireccionales, se conectan al positivo de la fuente mediante resistencias de pull-up. Se debe anotar que el estado de las líneas cuando el bus esta libre es alto (H).

Los dispositivos pueden clasificarse en maestro (master o principal) o esclavo (slave o secundario). El maestro es el que inicia la transferencia de datos y genera la señal de reloj. Cualquiera de los dispositivos seleccionado por un maestro se lo considera un esclavo.

El I2C es un bus multi-maestro, puede haber mas de un maestro conectado y controlando el bus. Normalmente se trata de elementos microcontroladores o microcomputadores.

Cada elemento posee una única dirección, mediante la cual el elemento maestro los reconoce. Estos elementos pueden ser establecidos como Maestro o Esclavo.

Para que exista una comunicación mediante el bus I2C se deben cumplir ciertas condiciones, una de ellas es la generación de una señal de START para el inicio, una señal de reconocimiento (ACK) luego de cada dato enviado y de una señal de STOP para el final. Estas señales son generadas por el elemento maestro, que reconoce al esclavo así como a la información que viene o dirige a el.

La condición de START se genera como se observa en la Ilustración 31. La línea SDA pasa de alto a bajo, mientras la línea SCL se mantiene en alto.

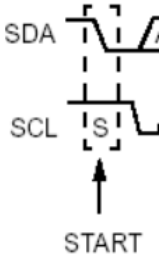

Ilustración 31 Condición de START

La señal de STOP se observa claramente en la Ilustración 32. La línea SDA pasa de bajo a alto, mientras la línea de reloj se mantiene en alto.

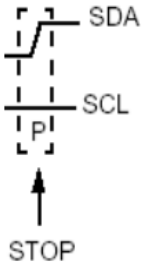

Ilustración 32 Condición de STOP

La señal de ACK, se registra cuando el elemento y las operaciones que se realizan en el esclavo son correctas, esta señal la genera el maestro indicando que el esclavo al cual se dirige es el correcto. En la Ilustración 33 se puede observar la señal, pero hay que notar que esta se general cuando la línea SCL esta en alto, entonces la línea SDA presenta un pulso en bajo.

Ilustración 33 Señal de Acknowledge (ACK)

El bit de datos transferido por la línea SDA debe mantenerse estable durante el periodo en que la señal de reloj esta en nivel '1'. La línea de datos SDA solo puede cambiar de estado durante el periodo en que la señal de reloj esta en '0'. Por lo tanto para que un dato sea valido debe coincidir con la señal de reloj en alto.

4.2.3 Esquema General

Básicamente lo que se pretende implementar es una comunicación entre múltiples dispositivos mediante dos líneas: la línea de datos y la línea de reloj, por medio de las cuales se comunicara información relevante al Reloj de tiempo real, al PIC maestro y esclavos. Además se pueden conectar otros dispositivos periféricos al sistema con fines de expansión, es así que el sistema se convierte en un elemento flexible, pedagógico y de continuo crecimiento en cuanto a tecnología.

En el esquema que a continuación se presenta, se puede observar claramente la aplicación del bus I2C, mediante dos líneas se envía la información a distintos dispositivos, convirtiéndose en una optima opción para la comunicación entre dispositivos de este tipo, reduciendo hardware y con un software adecuado para el estudiante.

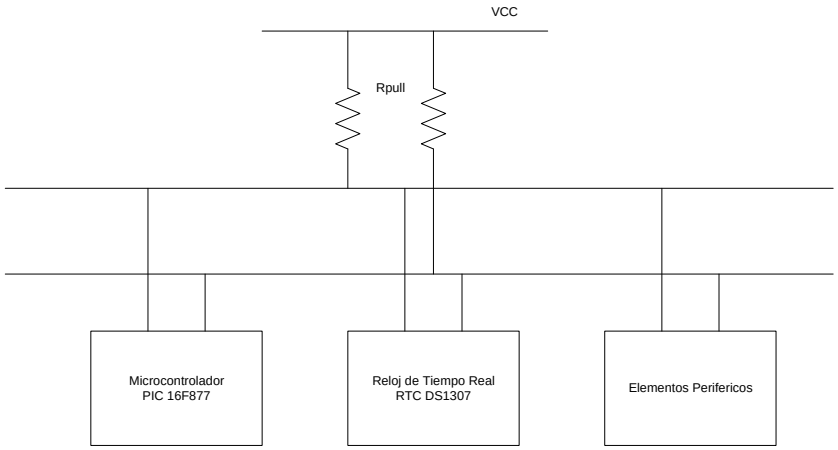

Ilustración 34 Diagrama General del sistema

4.2.4 Registros empleados

Para activar el bus I2C, se debe colocar el bit SSPEN=1, correspondiente al bit 5 del registro SSPCON (SSPCON<5>). A partir de ese momento los pines SDA y SCL quedan configuradas para soportar el protocolo I2C. Previamente dichos pines del PUERTO C, RC3/SCL y RC4/SDA deben ser configuradas como entradas mediante la escritura de los bits correspondientes en el registro TRISC.

Existen seis registros para controlar el bus I2C, los mismos que serán indicados a continuación.

- SSPCON._ Registro de Control.
- SSPCON2._ Registro de Control 2.
- SSPSTAT._ Registro de Estado.
- SSPBUF._ Buffer para los Datos.
- SSPSR._ Registro de desplazamiento no accesible directamente.
- SSPADD._ registro de dirección.

Mediante el bit CKE se establecen los niveles de referencia con respecto al bus I2C o al bus multimaestro. En caso de setearse con ‘0’ hará referencia al bus I2C y con ‘1’ se referirá al bus multimaestro.

Los 4 bits menos significativos del registro SSPCON sirven apara seleccionar el modo de trabajo del bus de la siguiente manera:

- Modo I2C maestro, Reloj=(SSPAD+1)*OSC/4
- Modo I2C esclavo con 7 bits para la dirección
- Modo I2C esclavo con 10 bits para la dirección

El registro de Estado SSPSTAT contiene la información que representa el estado de la transferencia de datos. Detecta las condiciones de inicio y de parada, así como la recepción del byte de dirección.

SSPBUF, es el registro que actúa como buffer y soporta el dato que va a ser transmitido o recibido. Este registro junto al SSPSR realiza la conversión serie/paralelo.

Cuando se ha completado la llegada del byte de información, este es recibido en el registro SSPSR, luego pasa al SSPBUF y se activa el señalizador SSPIF.

En caso de recibirse otro byte antes de leer el que se encuentra en el SSPBUF, se activa el señalizador de desbordamiento SSPOV, correspondiente al bit 6 del registro SSPCON.

El registro SSPAD guarda la dirección del esclavo. Cuando se emplean direcciones de 10 bits, el usuario debe escribir el byte alto de la dirección con el siguiente código, donde A9:A8 son los dos bits de mas peso: 1-1-1-1-0-A9-A8-0.

En cuanto al registro SSPCON2, el bit GCEN solo se usa en modo esclavo. Cuando se pone el bit ACKSTAT=1 significa que se ha recibido el bit de reconocimiento ACK del esclavo. SCKDT es el bit de reconocimiento, pero si vale 1 no lo ha hecho. Cuando se pone ACKEN=1 se inicia la secuencia de generación de la condición de reconocimiento. Este bit se borra automáticamente por hardware.

Para habilitar el modo de recepción del maestro hay que poner el bit RCEN=1. para generar la condición de parada en las líneas SCL y SDA hay que colocar PEN=1. El bit RCEN cuando se pone a 1inicia la repetición de la

condición de inicio. Finalmente, para iniciar la condición de inicio hay que poner SEN=1.

Para mayor información con respecto a los registros del bus I2C, y su funcionalidad, se pide al lector dirigirse a la bibliografía.

4.2.5 Esquema Electrónico

El módulo periférico para el Sistema SDP877 necesita para su funcionamiento únicamente conectar las dos líneas de comunicación por red I2C, además de la alimentación del circuito correspondiente.

Los pines empleados con este objetivo son RC3 y RC4 que respectivamente son las líneas de Reloj (SCL) y de Datos (SDA). Como se menciono anteriormente es necesaria la presencia de resistencias de pull-up entre estas líneas y el voltaje de alimentación.

En el módulo periférico se disponen de 3 aplicaciones que hacen uso del bus I2C, cuyas terminales anteriormente nombradas están conectadas con las del Sistema SDP877. Para estas aplicaciones se ha determinado el uso de un integrado RTC (Reloj de Tiempo Real) y de dos PIC's esclavos.

Las aplicaciones que se les dará a los elementos de comunicación I2C, se detallaran a continuación.

4.3 Elementos conectados a la red I2C

El primer dispositivo conectado a la red I2C, es un Reloj de Tiempo Real, conocido también como RTC.

El RTC es un integrado que como su nombre lo indica, brinda información referente al día, mes, año, hora, minutos, segundos, etc. El PIC maestro esta ubicado en la placa del Sistema SDP877 y este envía y recibe la información

adecuada y necesaria hacia el esclavo. Para su funcionamiento optimo precisa de un cristal de 32768 KHz.

Se realizaran tres prácticas con este integrado, se pretende inicializar el reloj de tiempo real, escribir en el integrado mediante el bus I2C y leer la información proveniente del integrado mediante el bus.

La información del reloj se presentara en el LCD del módulo maestro, que corresponde al Sistema SDP877, de igual forma para igualar el reloj se emplearan los dip switch ubicados en el módulo SDP877.

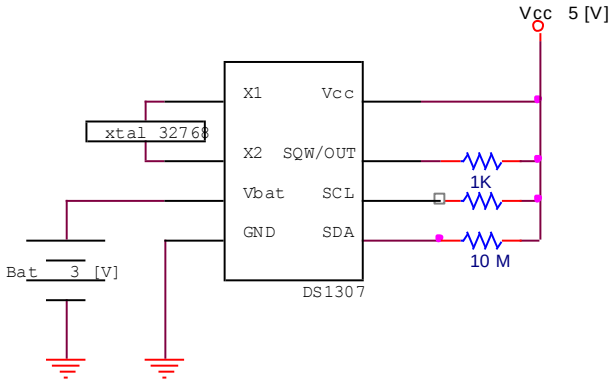

Ilustración 35 Configuración del RTC

Nota: Es necesario el uso de una batería de 3 Voltios, para mantener la configuración establecida por el usuario, ya que al quitar la energía al sistema el RTC pierde todos los cambios realizados, y se debería volver a configurar el RTC antes de operar cualquier otro programa. Con la batería se ahorra tiempo, que puede ser empleado para otras actividades y aplicaciones.

El segundo elemento es un microcontrolador PIC16F877, este elemento trabajara como esclavo del sistema, recibirá información proveniente de los potenciómetros conectados al Puerto A, luego enviara esos datos al bus I2C y el maestro los leerá, para luego presentarlos en el LCD.

Además se dispondrá de otro microcontrolador de iguales características que el anterior, este elemento tendrá por objeto el presentar la información proveniente del bus en las 4 matrices de leds ubicadas en la parte inferior del módulo. La información ingresada al bus será la que se envía desde el PC, mediante comunicación serial.

PRÁCTICAS REALIZADAS

4.4 Configuración del RTC

Realizar un programa que emplee el bus I2C para inicializar el Reloj de Tiempo Real (RTC), una vez realizado esto, se deberá comprobar el estado de activación del integrado mediante la señal emitida en el pin 7 (SQW/OUT).

4.4.1 Información requerida

Se debe conocer que el RTC posee 8 registros importantes, que se localizan desde la dirección 00H hasta la 07H, los seis primeros contienen la información relacionada con fecha, hora, minutos, segundos, etc. El ultimo registro es el 07H, este es el registro de control y es el que va a ser modificado para los requerimientos del problema.

El registro de control posee 8 bits, de los cuales los bits 0 y 1 son los encargados del control de la frecuencia de la onda cuadrada de salida. De acuerdo a la combinación que se establezca en estos bits será definida la frecuencia.

El bit 4, SQWE, es el que deberá setearse para que el oscilador se encienda y de esta forma se active el RTC.

Entonces la palabra de configuración que se deberá escribir en el registro 07H será 0b00010000, correspondiente a 10H.

4.4.2 Descripción de Funcionamiento

Se puede observar en las líneas subrayadas, la forma en la que se debe configurar el RTC, mediante el bus I2C. Los estados que existen en el Bus se indican en la variable enum i2c_bus_states, y con esa configuración se debe configurar cualquier otro dispositivo.

Primero se genera una señal de START (1), luego se indica la dirección del esclavo con el que se desea trabajar, WRITE_ADDRESS0 (10), que en este caso es el RTC, cuya dirección es D0. Luego se envía una indicación de que se desea realizar escritura de datos, en este caso a la dirección indicada se sumara 1, WRITE_ADDRESS1(8), se escribe el valor cero para inicializar, luego se vuelve a dar una señal de RESTART, seguida siempre de la dirección del esclavo para proceder a indicar la dirección del registro y el valor que se desea cargar en el mismo.

En el arreglo WriteStatBufSlaveIni, se observan los valores de los registros a modificar y el valor con el que serán grabados.

Con respecto a la subrutina I2CbusState, es una función muy útil y de importancia vital en este análisis, pues la función se emplea en todos los programas sin ningún tipo de cambio, es flexible a cualquier programa de comunicación I2C.

Permite la configuración de los bits de los registros encargados de la comunicación I2C en una sola subrutina y dependiendo de la función que van a cumplir, tal es el caso de la función START, en la que se configura al bit RCEN como 1. esto permite que la comunicación I2C se inicie, se genera la señal correspondiente y las líneas SDA y SCL son activadas con este propósito.

De igual manera con STOP, en la que configura PEN=1. para el caso de Escritura guarda el valor de la dirección del esclavo en el SSPBUF. Si por el contrario la operación a realizar es la de lectura se guarda en SSPBUF la dirección del esclavo + 1 porque el LSB indica la operación a ejecutar. Cuando se desee escribir un dato, se guardara en el

SSPBUF el valor del registro al que señala el puntero respectivo. Si se desea leer, se activa el bit RCEN=1.

Para las señales de ACK se encera la bandera correspondiente ACKDT y para la señal de NACK se lo configura como '1'.

Para el correcto desempeño del programa se crearon banderas que tienen la finalidad de enviar las señales necesarias para el funcionamiento de la comunicación I2C.

Las banderas empleadas a lo largo de todos los programas se indican en la siguiente tabla, pero el usuario puede ir incrementándolas a medida que vaya desarrollando su propia aplicación.

Nombre de la bandera	Función
sflag.event.i2c	Bandera que con un valor de '1' indica la existencia de un evento I2C.
eflag.i2c.ack_error	Indica un error en la comunicación al generarse una colisión del bus.
sflag.event.write_state	Verifica que el ultimo estado de la comunicación haya sido escritura.
sflag.event.writes_done	Toma el valor de '1' cuando ya se realizo la escritura de un dato.

Tabla 7 Banderas empleadas en todas las aplicaciones

Además de las banderas indicadas se crearon otras dependiendo de la aplicación a realizar. En este caso se creo ***sflag.event.rtc_start***, es la bandera encargada de registrar la inicialización del reloj, una vez inicializado se pone en '1'.

4.4.4 Diagramas de Flujo

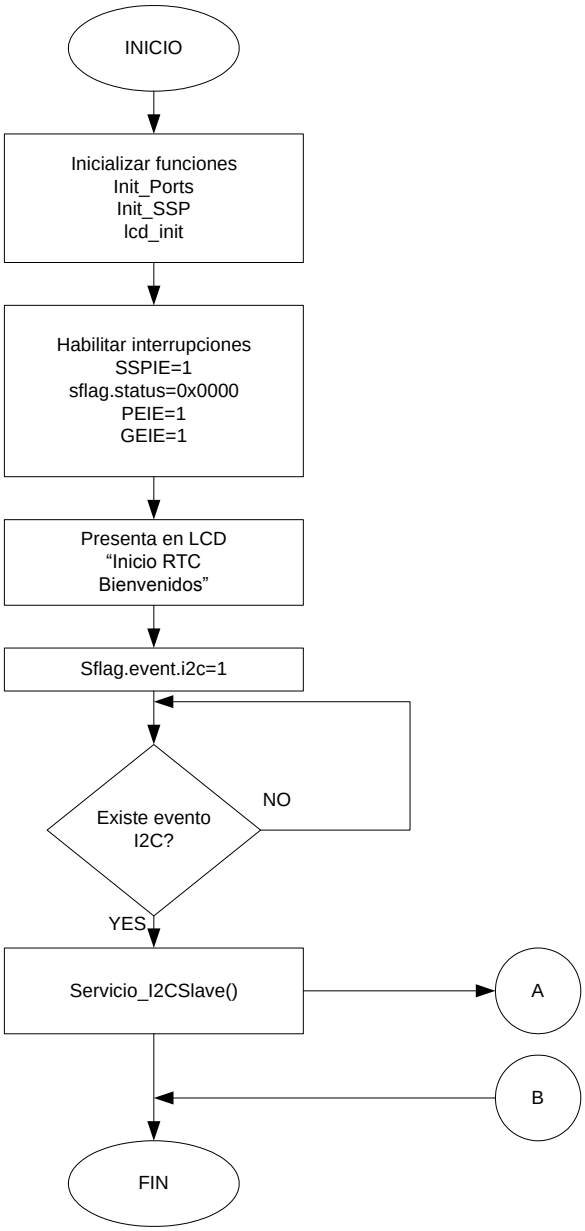

Ilustración 36 Diagrama de Flujo del programa Maestro1.C

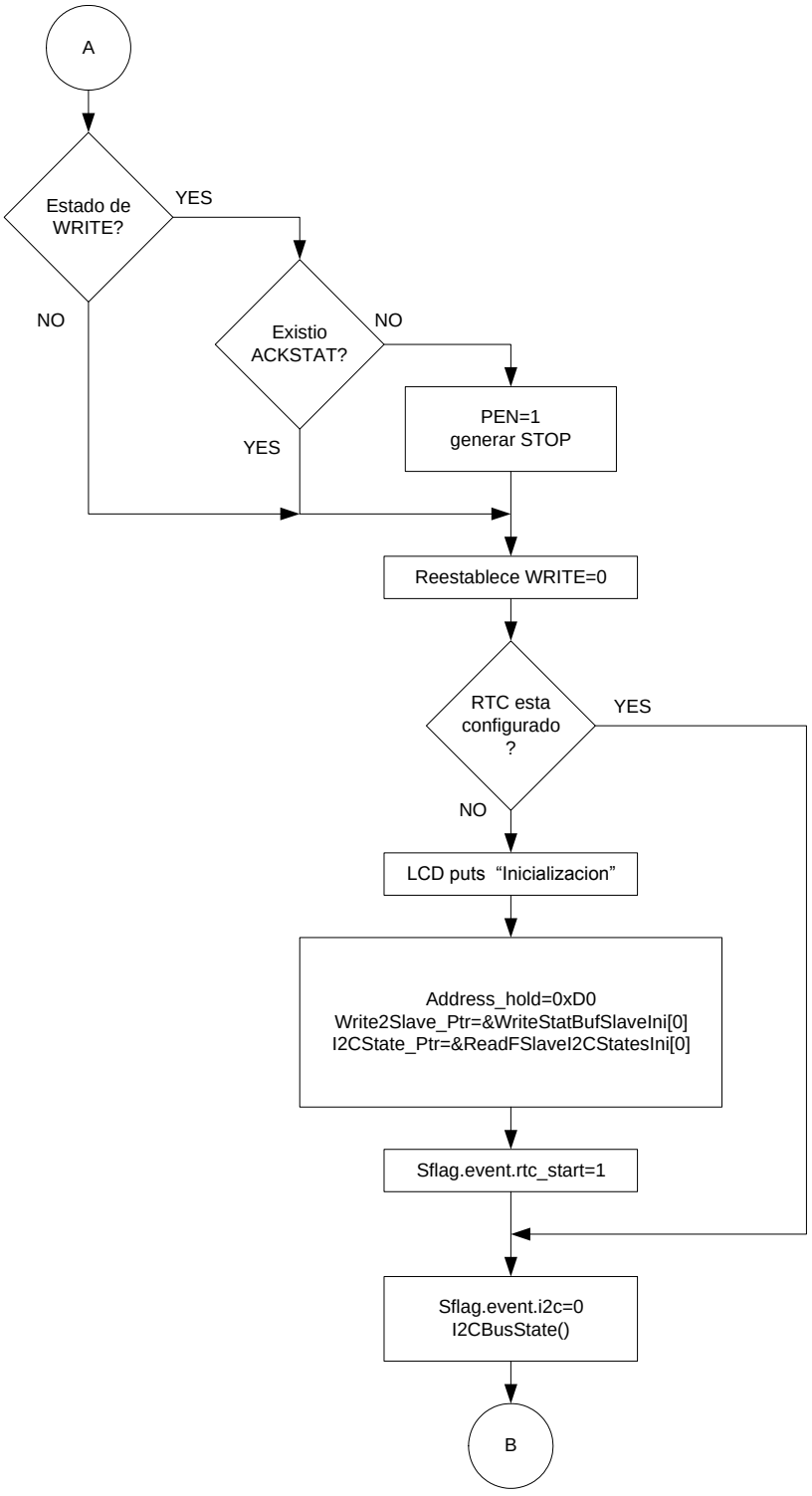

Ilustración 37 Diagrama de flujo del programa COM1_I2C.C

4.4.5 Funcionamiento del programa

La finalidad del programa es la comunicación mediante el bus I2C, para obtener y enviar información desde un dispositivo maestro, que en este caso es un microcontrolador PIC 16F877, a un elemento esclavo que es el Reloj de tiempo Real, conocido como RTC.

Se presentaran a continuación ilustraciones obtenidas de la respuesta del sistema a este proceso, mediante un osciloscopio digital.

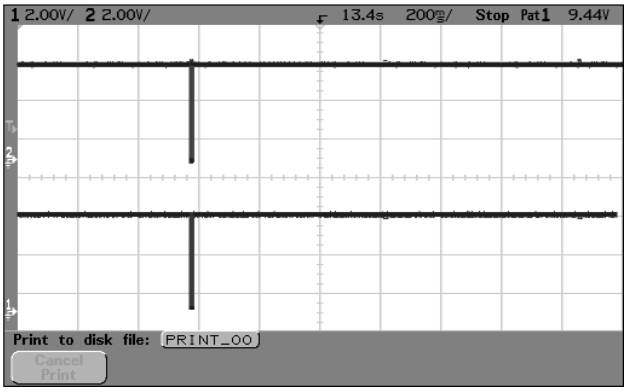

Ilustración 38 Evento I2C

La señal superior corresponde a la línea de Datos y la inferior a la línea de reloj.

Cada una de estas señales ocurren cuando se envía o se recibe información a través del bus I2C.

Si se hace un análisis de este evento se podrá observar características mas detalladas, que permitirán verificar el proceso que ocurre en la comunicación, así como las señales que se están enviando con el propósito de inicializar el reloj.

Recordemos que el propósito de la presente práctica es inicializar el reloj de tiempo real o RTC. Esto se lo realiza grabando en el registro de control un valor determinado, el registro de control esta en la dirección 0x07H del RTC, mientras que la palabra de control a ser grabada es la 0x10H.

Una vez que esto se realice, se podrá comprobar que el RTC comienza a funcionar mediante la salida que se registre en el pin 7, correspondiente a SQW/OUT.

Este pin emitirá una señal oscilante, una vez que el RTC ha sido inicializado.

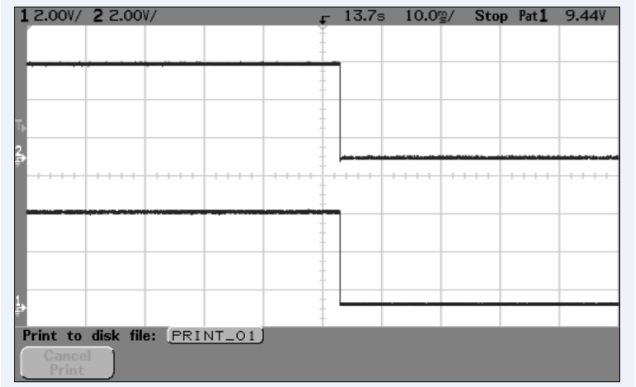

Ilustración 39 Señal de START

Para una adecuada comunicación mediante el bus I2C es necesaria la generación de una señal de inicio o de START, como ya se indico anteriormente se la reconoce porque la línea SDA pasa a bajo cuando la línea SCL se encuentra en alto.

Este es el primer paso para que el bus de comunicación inicie su funcionamiento.

Ilustración 40 Señal de ADDRESS_0 (0xD0H)

Posteriormente se envía una señal de direccionamiento (Case WRITE_ADDRESS0), esta es la que indica la dirección del esclavo con el cual se desea trabajar. La dirección se registra en los 7 bits mas significativos, mientras que el bit menos significativo indica el proceso a realizarse, ya sea éste lectura (1) o escritura(0).

Al finalizar esta segunda señal el noveno pulso de reloj debe corresponder a una señal en bajo de la línea SDA. Esto significara que el esclavo reconoce la dirección que esta siendo enviada, en caso de no reconocer este ultimo bit, significara que algo malo esta sucediendo en la comunicación, por lo que se deberá verificar conexiones y luego verificar las direcciones que envía el maestro junto a la dirección que el esclavo tiene.

En este caso se observa que la señal enviada por el maestro es 0b11010000, que corresponde a 0xD0H, y al final la señal de ACK es correcta. Por lo tanto la comunicación hasta este punto es correcta, pues la dirección del esclavo que se envía es la correcta y este esta reconociendo esta dirección mediante la emisión de la señal ACK.

Ilustración 41 Registro 0x00H

Siguiendo la configuración establecida en el programa, primero en el registro 0x00H se grabara el valor 0x00H, luego en el registro 0x07H se grabara el valor 0x10H. En esta ilustración se observa el primer registro del RTC al que se accede desde el maestro, correspondiente al 0x00H.

Ilustración 42 Valor 0x00H a ser grabado en la dirección 0x00H

La siguiente señal en enviarse es el valor que debe ser grabado en esa dirección, este corresponde a 0x00H. El registro 0x00H es el que maneja los segundos del RTC por lo tanto van a ser encerados.

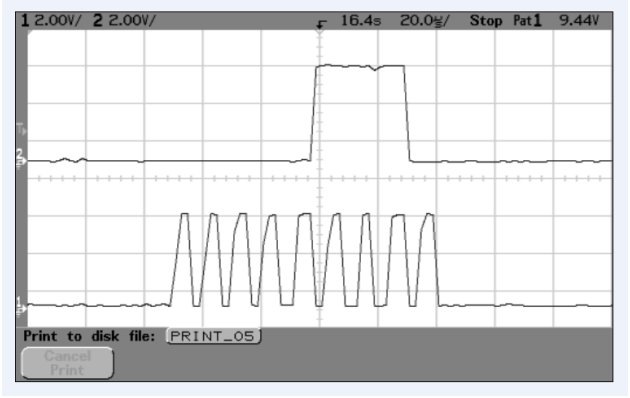

Ilustración 43 Dirección del registro de control del RTC, 0x07H

La próxima señal a ser enviada corresponde a la dirección 0x07H, que efectivamente se verifica mediante la señal mostrada en la ilustración 43, siendo esta la 0b00000111. Este registro es el encargado del control del RTC, mediante el cual vamos a inicializarlo.

Ilustración 44 Palabra de configuración del RTC, 0x10H

Posteriormente, una vez dentro del registro, se procede a grabar la palabra de control, que como ya se indico anteriormente es la 0x10H. Esta palabra de configuración se observa en la ilustración superior, correspondiendo al código binario 0b00010000.

Una vez que se envió la información necesaria a través del bus, se debe enviar una ultima señal, que es la de STOP o parada de comunicación. Ya se indico que esto se observa cuando la señal SDA pasa a alto, mientras la señal de SCL se mantiene en alto.

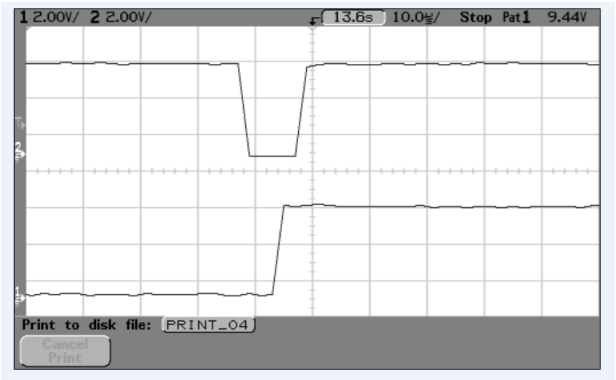

Ilustración 45 Señal de STOP

Cuando el RTC fue configurado e inicializado, la única manera de comprobarlo es observando la señal que se genera en el pin 7 del mismo. La salida SQW/OUT debe oscilar cuando el RTC inicia su funcionamiento. Esto se puede observar en la ilustración inferior.

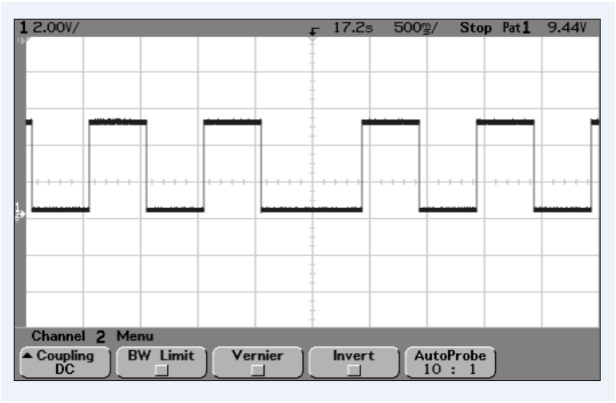

Ilustración 46 Salida del pin SQW/OUT del RTC

Nota: Es necesario resaltar que el RTC es un elemento volátil, es decir cuando la alimentación de la fuente se quita, entonces toda la configuración del mismo se pierde, de ahí la necesidad de emplear la batería de litio de 3V en el pin 3. de no hacerlo, cada vez que se encienda el sistema se deberá primero configurar el RTC antes de realizar cualquier otra práctica.

4.4.6 Programa No. 1

Maestro1.C

```
#include <pic.h> // processor if/def file
#include "c:\ht-pic\samples\mas_i2c.h"
#include "c:\ht-pic\samples\init.c"
#include "c:\ht-pic\samples\com1_i2c.c"

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>

void main(void)
{

//INICIALIZACIONES
Init_Ports(); // inicializa Puertos
lcd_init(); // inicializa el LCD
```

```
Init_Ssp(); // inicializa el módulo SSP

//HABILITA INTERRUPCIONES
SSPIE = 1; // habilita interrupciones por I2C
sflag.status = 0x0000; // encera las banderas
PEIE = 1; // habilita interrupciones periféricas
GIE=1;

//PRESENTACION EN LCD
lcd_clear();
lcd_goto(3);
lcd_puts("Inicio RTC");
lcd_goto(0x40);
lcd_puts("***Bienvenidos***");
DelayMs(250);
DelayMs(250);
DelayMs(250);
lcd_clear();
sflag.event.i2c=1;
for(;;) // LAZO INFINITO
{

        if(!sflag.event.i2c)//BANDERA DE EVENTO I2C
        {
                lcd_puts("wait");
        }
        else{
                DelayUs(400);
                Service_I2CSlave();
        }
}//FIN DEL CICLO INFINITO
}//FIN DEL MAIN

void interrupt piv( void )
{
        if ( SSPIE && SSPIF ) // test for I2C event completion
        {
                SSPIF = 0; // reset I2C based interrupt flag
                sflag.event.i2c = 1; // set I2C event service flag
        }
```

```
}
```

COM1_I2C.C

```
#include <pic.h>
#include "c:\ht-pic\samples\comm_i2c.h"
#include "c:\ht-pic\samples\delay.h"
#include "c:\ht-pic\samples\lcd.c"


//FUNCION SERVICIO I2C
void Service_I2CSlave( void )
{               DelayUs(400);
                if(!sflag.event.rtc_start)//AUN NO SE CONFIGURA RTC
                {
                        lcd_puts("INICIALIZACION");
                        DelayMs(250);
                        //DelayMs(250);
                        address_hold=0xD0;
                        Write2Slave_Ptr = &WriteStatBufSlaveIni[0];
                        I2CState_Ptr = &ReadFSlaveI2CStatesIni[0];
                        DelayMs(11);
                        sflag.event.rtc_start=1;
                }//FIN AUN NO CONFIGURA LOS PUNTEROS

if (sflag.event.write_state)//PRUEBA SI EL ULTIMO ESTADO I2C FUE ESCRITURA
{
        if ( ACKSTAT ) // was NOT ACK received?
        {
        PEN = 1; // generate bus stop condition
        eflag.i2c.ack_error = 1; // set acknowledge error flag
        }//FIN PRUEBA DE ACK
        sflag.event.write_state = 0; // reset write state flag
}//FIN ULTIMO ESTADO ESCRITURA

        sflag.event.i2c=0;
        I2CBusState(); // execute next I2C state

}//FIN DE FUNCION SERVICIO I2C
```

```
//FUNCION ESTADOS DE BUS

void I2CBusState ( void )
{
i2cstate = *I2CState_Ptr++; //ACTULIZA PUNTERO CON SIGUIENTE ESTADO
switch ( i2cstate ) //EVALUA EL ESTADO
{
        case ( READ ): //LECTURA
                RCEN = 1; //
                break;
        case ( WRITE_DATA ): // ESCRITURA
                SSPBUF = *Write2Slave_Ptr++; //ALMACENA EN BUFFER
                sflag.event.write_state = 1;
                break;
        case ( WRITE_ADDRESS1 ): //(R/W=1)
                SSPBUF = address_hold + 1;
                sflag.event.write_state = 1;
                break;
        case ( START ): // START
                SEN = 1;
                break;
        case ( WRITE_ADDRESS0 ): //(R/W=0)
                SSPBUF = address_hold;
                sflag.event.write_state = 1;
                break;
        case ( SEND_ACK ): // ACK
                *ReadFSlave_Ptr++ = SSPBUF;
                if ( read_count > 0)
                {
                        read_count -= 1;
                        I2CState_Ptr -= 2;
                }
                ACKDT = 0;
                ACKEN = 1;
                break;
        case ( SEND_NACK ): //NACK
                *ReadFSlave_Ptr = SSPBUF;
                ACKDT = 1;
                ACKEN = 1;
                break;
```

```
        case ( STOP )://STOP
                PEN = 1;
                sflag.event.next_i2cslave = 1;
                sflag.event.writes_done = 1;
                break;
        case ( RESTART ):
                RSEN = 1;
                break;
        default:
                break;

}//FIN EVALUAR ESTADO
}//FIN FUNCION DE ESTADOS
```

COM_I2C.H

```
void I2CBusState( void );

//VARIABLES ( DEFINED HERE )
unsigned char address_hold, read_count;
unsigned char i2cstate;
unsigned char *Write2Slave_Ptr;
bank1 unsigned char ReadStatBufFromSlave[3];
bank1 unsigned char *ReadFSlave_Ptr;
unsigned char WriteStatBufSlaveIni[4]={0x00,0x00,0x07,0x10};  //Inicio
unsigned char WriteStatBufSlaveIg[4]={0x01,0x00,0x00,0x01};//Escritura
unsigned char WriteStatBuf2Slave[1]={0x00} ;  //Lectura

// define estados I2C
enum i2c_bus_states{ START =1, RESTART =2, STOP =3, SEND_ACK =4, SEND_NACK =5,
GEN_CALL =6, READ =7, WRITE_DATA =8, WRITE_ADDRESS1 =9, WRITE_ADDRESS0 =10
};

const unsigned char ReadFSlaveI2CStatesIni[]={1,10,8,8,2,10,8,8,3,0}; //Inicio
const unsigned char ReadFSlaveI2CStatesIg[] = {1,10,8,8,8,2,10,8,2,9,7,4,7,5,3,0}; //Escritura
const unsigned char ReadFSlaveI2CStates[] = {1,10,8,2,9,7,4,7,5,3,0}; //Lectura

const unsigned char *I2CState_Ptr; // puntero para acceder a los estados I2C
```

4.5 Lectura del RTC

Elaborar un programa que permita mediante la comunicación I2C leer los registros de horas, minutos y segundos del RTC. El microcontrolador PIC 16F877 actuara como maestro. La información será presentada en el LCD del sistema maestro SDP877.

4.5.1 Diagrama General de Funcionamiento

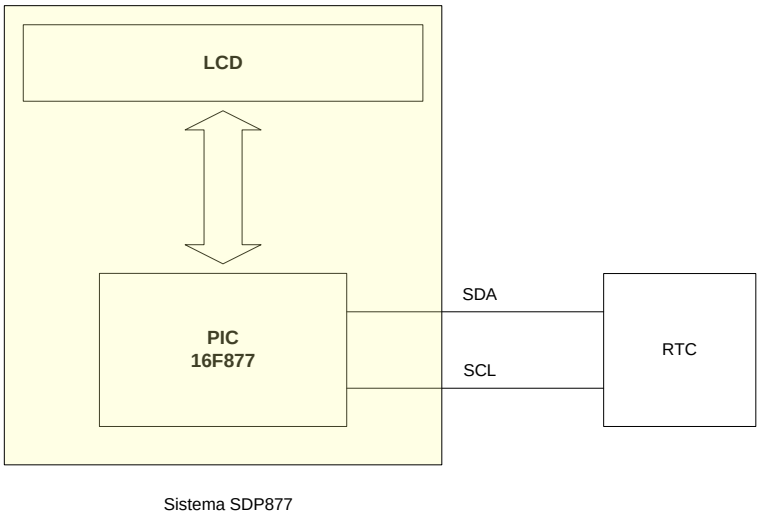

Ilustración 47 Diagrama general de funcionamiento

La información que se obtenga del RTC y se envíe mediante el bus I2C, se presentara en el LCD del sistema maestro, en el siguiente formato:

HH:MM:SS

En cuanto a las banderas creadas para esta aplicación, a mas de las explicadas en la práctica anterior y que son generales para el resto de aplicaciones, se uso la bandera ***sflag.event.rtc_start***, que registra el momento

en que la información del RTC es leída, entonces que se coloca un ‘1’ en la bandera.

4.5.2 Diagramas de Flujo

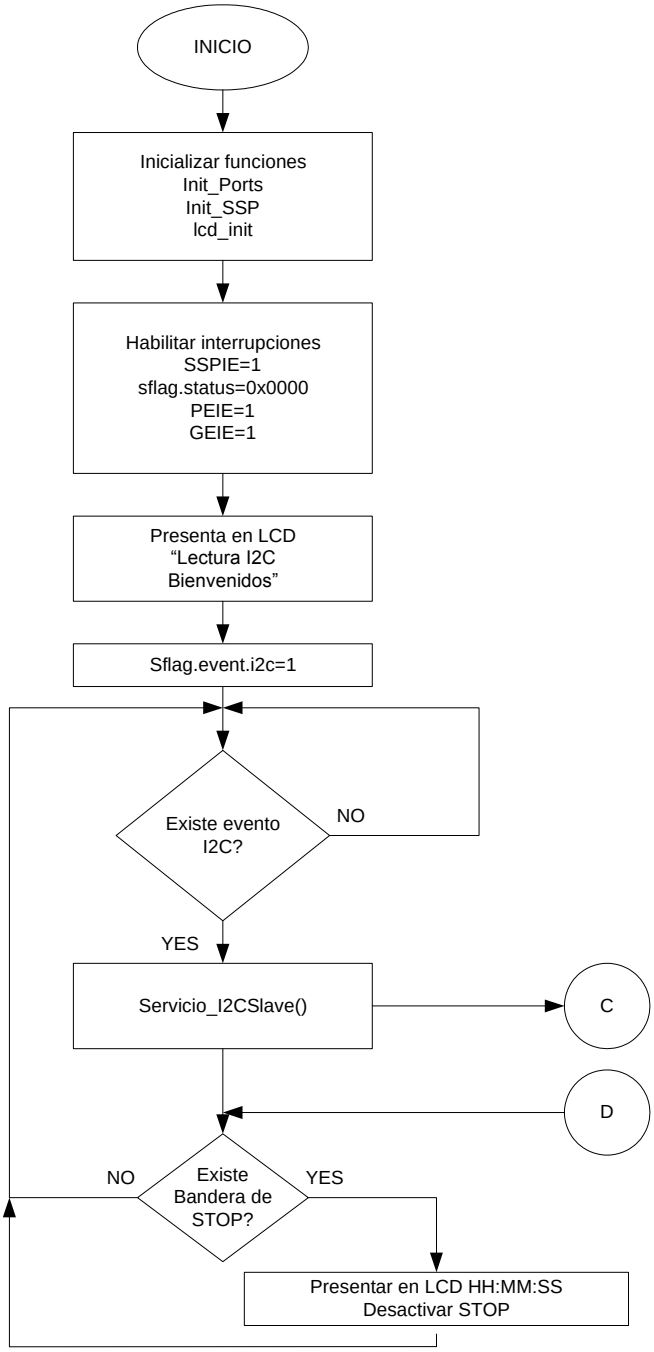

Ilustración 48 Diagrama de Flujo del programa Maestro3.C

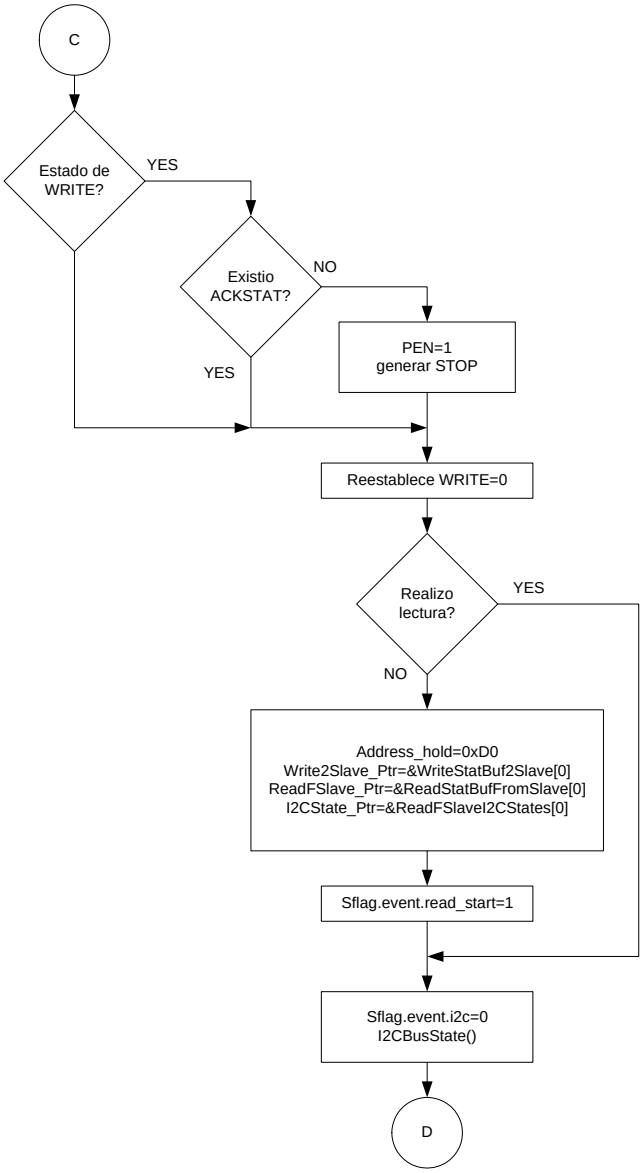

Ilustración 49 Diagrama de flujo del programa Comm3_i2c.C

4.5.3 Funcionamiento del Programa

Se debe poner atención en el registro WriteStatBuf2Slave, se coloca el valor 0x00H, debido a que desde esa dirección se ubican los registros que contienen la información necesaria para esta práctica, siendo los registros los correspondientes a segundos (0x00H), minutos (0x01H) y horas (0x02H).

Como ya se indico anteriormente el RTC debe estar inicializado para la ejecución de cualquier otra práctica.

En este primer grafico se observara el evento I2C ocurrido por cada lectura que se realice del RTC. La línea superior corresponde a los pulsos que contienen los datos y la inferior corresponde a los pulsos de reloj.

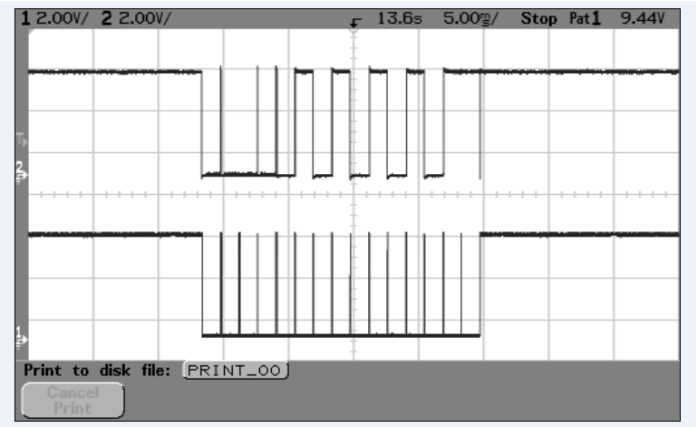

Ilustración 50 Evento I2C de Lectura

Como sucede en todo ciclo de comunicación mediante el bus I2C, es necesario generar una señal de START, la misma que tiene las características iguales a las nombradas en la práctica anterior.

Luego, se envía una señal de ADDRESS_0, que indica la dirección del esclavo y del cual el ultimo bit indica la operación a realizarse, en este caso se realizara primero una operación de escritura (0), con la finalidad de ubicar al puntero en la posición adecuada (0x00H), para iniciar la lectura de los registros de horas, minutos y segundos (Ilustración 51).

La dirección del esclavo continua siendo 0xD0H, pues el elemento empleado es el RTC, y el bit de reconocimiento indica que no existe problemas en la comunicación con el esclavo, significa que la dirección esta siendo reconocida.

Ilustración 51 Señal de ADDRESS_0 (0xD0H)

Si se observa en el programa la variable ReadFSlaveI2Cstates, la siguiente operación a ejecutarse es un RESTART, que es una señal similar a la de START y que siempre debe ir seguida de la dirección del esclavo junto a la operación a realizarse (lectura o escritura).

Ilustración 52 Señal de RESTART

Esta va inmediatamente seguida de ADDRESS_1, señal que indica la dirección del esclavo, pero la operación a realizar es de lectura, motivo por el cual el ultimo bit es un 1 lógico.

Ilustración 53 Señal de ADDRESS_1 (0xD1H)

Ahora comienzan a leerse los datos desde el registro 0x00H, en la ilustración 54 se observa el dato de los segundos, que en binario es 0b00010010, correspondiente a 0x12H.

Ilustración 54 Dato del registro 0x00H, 12 segundos

Al final de cada dato, se observa la señal de ACK, que indica que la comunicación I2C es correcta, que no presenta problemas y que el esclavo es reconocido adecuadamente.

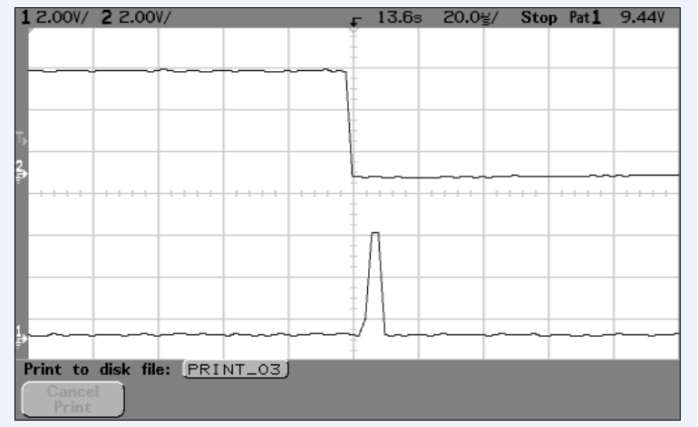

Ilustración 55 Señal de ACK enviada al final de cada dato

La señal del registro 0x01H, de los minutos es el mostrado en la ilustración 56, con un valor binario de 0b01000000, equivalente a 0x40H. En la ilustración 57 se observa la señal del registro 0x02, de las horas, con un valor binario de 0b00000101, equivalente a 0x05H. Todas seguidas del bit de reconocimiento o ACK.

Ilustración 56 Dato del registro 0x01H, 40 minutos

Ilustración 57 Dato del registro 0x02H, 5 horas

La comunicación I2C finaliza con una señal de STOP o parada, cuya ilustración es igual a la mostrada en la práctica anterior.

4.5.4 Programa No. 2

Maestro3.C

```
#include <pic.h>
#include "c:\ht-pic\samples\mas_i2c.h"
#include "c:\ht-pic\samples\init.c"
#include "c:\ht-pic\samples\com3_i2c.c"

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>

void main(void)
{

//VARIABLES
unsigned char min[10];
unsigned char hora[10];
unsigned char seg[10];
unsigned char dato[10];
```

```
//INICIALIZACIONES
Init_Ports(); // inicializa Puertos
lcd_init(); // inicializa el LCD
Init_Ssp(); // initializa el módulo SSP
Init_Timer1(); // initializa el TMR1

//HABILITA INTERRUPCIONES
TMR1IE = 0; // interrupcion del  TMR1
SSPIE = 1; // interrupciones I2C
sflag.status = 0x0000; // todas las banderas de eventos son reseteadas
PEIE = 1; // habilita interrupciones periféricas
GIE=1;

//PRESENTACION EN LCD

lcd_clear();
lcd_goto(3);
lcd_puts("red I2C - Lectura");
lcd_goto(0x40);
lcd_puts("***Bienvenidos***");
DelayMs(250);
DelayMs(250);
DelayMs(250);
lcd_clear();
sflag.event.i2c=1;

for(;;) // LAZO INFINITO
{
        if(!sflag.event.i2c)//BANDERA DE EVENTO I2C
        {lcd_puts("wait");
        }
        else{
                DelayUs(400);
                Service_I2CSlave();
        }
                if(sflag.event.writes_done)//IF PARA PRESENTACION
        {
                if(ReadStatBufFromSlave[0]>=60)
                {
                        ReadStatBufFromSlave[0]=0;
```

```
                    ReadStatBufFromSlave[1]++;
                }
                if(ReadStatBufFromSlave[1]>=60)
                {
                    ReadStatBufFromSlave[1]=0;
                    ReadStatBufFromSlave[2]++;
                }

                lcd_clear();
                lcd_goto(0x00);
sprintf(dato,"%02d:%02d:%02d",ReadStatBufFromSlave[2],ReadStatBufFromSlave[1],
ReadStatBufFromSlave[0]);
                lcd_puts(dato);
                DelayMs(250);
                DelayMs(250);

                sflag.status=0x0000;
                sflag.event.i2c=1;

        }//FIN DE IF DE PRESENTACION
}//FIN DEL CICLO INFINITO
}//FIN DEL MAIN

void interrupt piv( void )
{
        if ( SSPIE && SSPIF ) // prueba de realizacion de evento I2C
        {
                SSPIF = 0;
                sflag.event.i2c = 1; // setea la bandera de evento
        }
}
```

COMM3_I2C.C

```
#include <pic.h>
#include <stdio.h>
#include "c:\ht-pic\samples\comm_i2c.h"
#include "c:\ht-pic\samples\delay.h"
#include "c:\ht-pic\samples\lcd.c"
unsigned char seg[10];
```

```
//FUNCION SERVICIO I2C
void Service_I2CSlave( void )
{                   DelayUs(400);

                    if(!sflag.event.read_start)//SI AUN NO EXISTE PETICION DE LECTURA
                    {
                            address_hold=0xD0;
                            read_count=3;
                            if(!sflag.event.writes_done){
                            Write2Slave_Ptr = &WriteStatBuf2Slave[0];
                            ReadFSlave_Ptr = &ReadStatBufFromSlave[0];
                            I2CState_Ptr = &ReadFSlaveI2CStates[0];
                            DelayMs(11);
                            }
                            sflag.event.read_start=1;
                    }// FIN INICIALIZO PUNTEROS

if (sflag.event.write_state)//PRUEBA SI EL ULTIMO ESTADO I2C FUE ESCRITURA
{
        if ( ACKSTAT ) // was NOT ACK received?
        {
        PEN = 1; // generate bus stop condition
        eflag.i2c.ack_error = 1; // set acknowledge error flag
        }//FIN PRUEBA DE ACK
        sflag.event.write_state = 0; // reset write state flag

}//FIN ULTIMO ESTADO ESCRITURA

        sflag.event.i2c=0;
        I2CBusState(); // execute next I2C state

}//FIN DE FUNCION SERVICIO I2C

//FUNCION ESTADOS DE BUS
void I2CBusState ( void )
{
i2cstate = *I2CState_Ptr++; //ACTULIZA PUNTERO CON SIGUIENTE ESTADO
switch ( i2cstate ) //EVALUA EL ESTADO
{
```

```
case ( READ ): //LECTURA
        RCEN = 1; //
        break;
case ( WRITE_DATA ): // ESCRITURA
        SSPBUF = *Write2Slave_Ptr++; //ALMACENA EN BUFFER
        sflag.event.write_state = 1;
        break;
case ( WRITE_ADDRESS1 ): //(R/W=1)
        SSPBUF = address_hold + 1;
        sflag.event.write_state = 1;
        break;
case ( START ): // START
        SEN = 1;
        break;
case ( WRITE_ADDRESS0 ): //(R/W=0)
        SSPBUF = address_hold;
        sflag.event.write_state = 1;
        break;
case ( SEND_ACK ): // ACK
        *ReadFSlave_Ptr++ = SSPBUF;
        if ( read_count > 0)
        {
                read_count -= 1;
                I2CState_Ptr -= 2;
        }
        ACKDT = 0;
        ACKEN = 1;
        break;
case ( SEND_NACK ): //NACK
        *ReadFSlave_Ptr = SSPBUF;
        ACKDT = 1;
        ACKEN = 1;
        break;
case ( STOP )://STOP
        PEN = 1;
        sflag.event.next_i2cslave = 1;
        sflag.event.writes_done = 1;
        break;
case ( RESTART ):
        RSEN = 1;
```

```
                break;
        default:
                break;

}//FIN EVALUAR ESTADO
}//FIN FUNCION DE ESTADOS
```

COMM_I2C.H

```
//DEFINICION DE FUNCIONES
void I2CBusState( void );

//DEFINICION DE VARIABLES
unsigned char address_hold, read_count;
unsigned char i2cstate;
unsigned char *Write2Slave_Ptr;
bank1 unsigned char ReadStatBufFromSlave[3];
bank1 unsigned char *ReadFSlave_Ptr;

unsigned char WriteStatBufSlaveIni[4]={0x00,0x00,0x07,0x10};  // Inicio
unsigned char WriteStatBufSlaveIg[4]={0x01,0x00,0x00,0x01};  // Escritura
unsigned char WriteStatBuf2Slave[1]={0x00} ;  // Lectura

// define estados I2C

enum i2c_bus_states{ START =1, RESTART =2, STOP =3, SEND_ACK =4, SEND_NACK =5,
GEN_CALL =6, READ =7, WRITE_DATA =8, WRITE_ADDRESS1 =9,
WRITE_ADDRESS0 =10 };

const unsigned char ReadFSlaveI2CStatesIni[]={1,10,8,8,2,10,8,8,3,0};  // Inicio
const unsigned char ReadFSlaveI2CStatesIg[]={1,10,8,8,8,2,10,8,2,9,7,4,7,5,3,0};  // Escritura
const unsigned char ReadFSlaveI2CStates[] = {1,10,8,2,9,7,4,7,5,3,0};  // Lectura

const unsigned char *I2CState_Ptr; // puntero para acceder a los estados I2C
```

4.6 Escritura en el RTC

Elaborar un programa que permita mediante la comunicación por el bus I2C, igualar las horas y minutos del Reloj de Tiempo Real. El elemento maestro será el microcontrolador PIC 16F877 del sistema SDP877, se emplearan los DIP SWITCH del sistema Maestro y la presentación de la información se lo hará mediante el LCD del mismo sistema.

4.6.1 Diagrama General de Funcionamiento

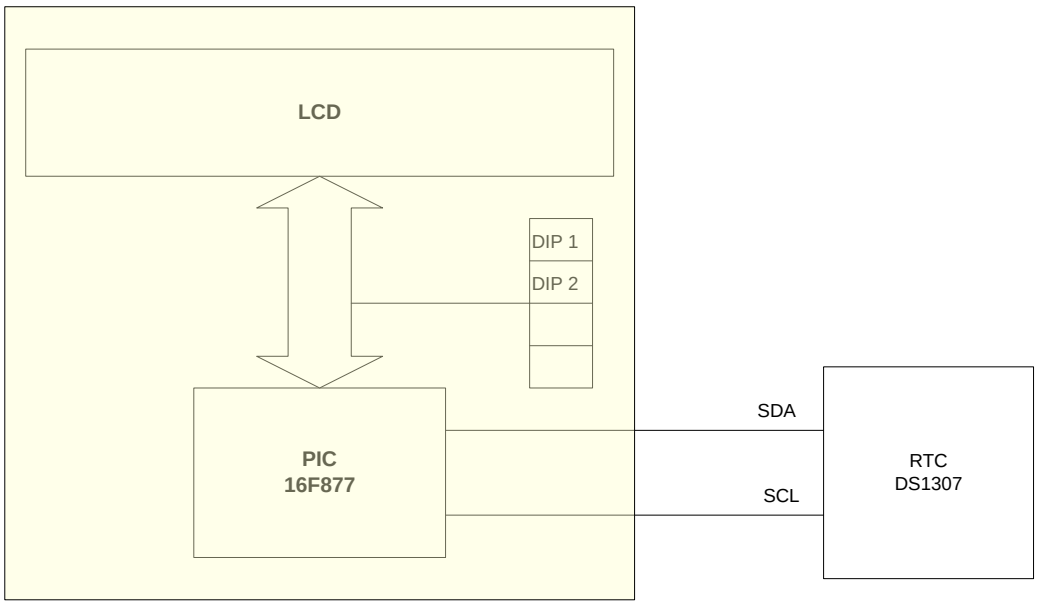

Ilustración 58 Diagrama general de funcionamiento

Básicamente se quiere obtener la información de RTC para presentarla en el LCD del sistema maestro, en el formato HH:MM:SS. Mediante el uso de los DIP SWITCH del mismo sistema se modificarán los minutos y las horas, una vez desactivados la cuenta del reloj continua desde el valor establecido por el usuario.

La bandera empleada en esta práctica además de las que son comunes para el resto de programas es ***sflag.event.igualar***, encargada de verificar la operación a realizar, si detecta cambio en los dip switch se tomara como '1' y la función que realizara será escribir en el bus, pero si no detecta ningún cambio se tomara como '0' y procederá solamente a realizar la lectura del RTC.

4.6.2 Diagramas de Flujo

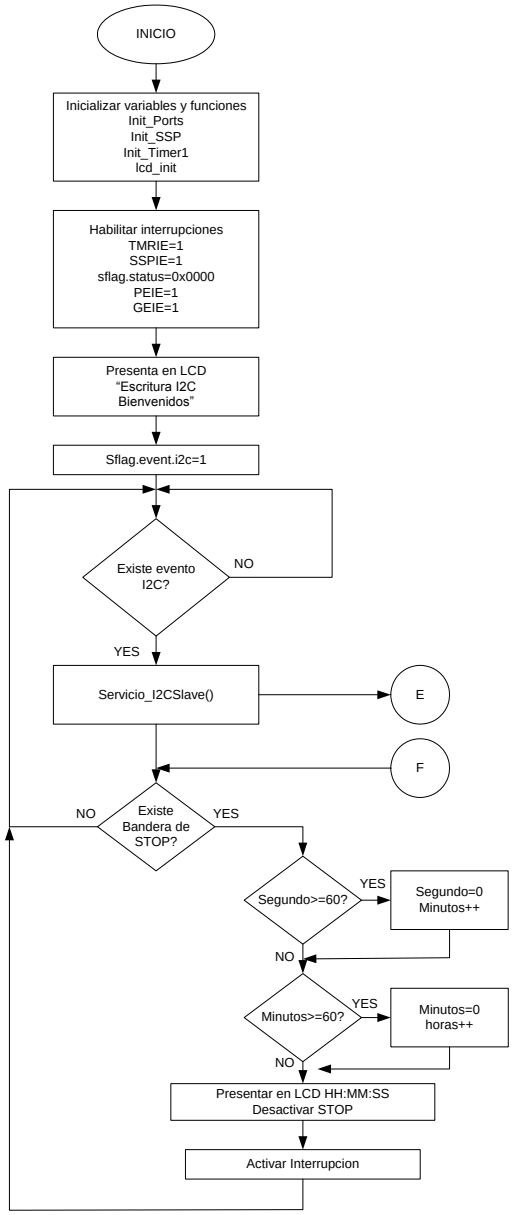

Ilustración 59 Diagrama de Flujo del programa Maestro4.C

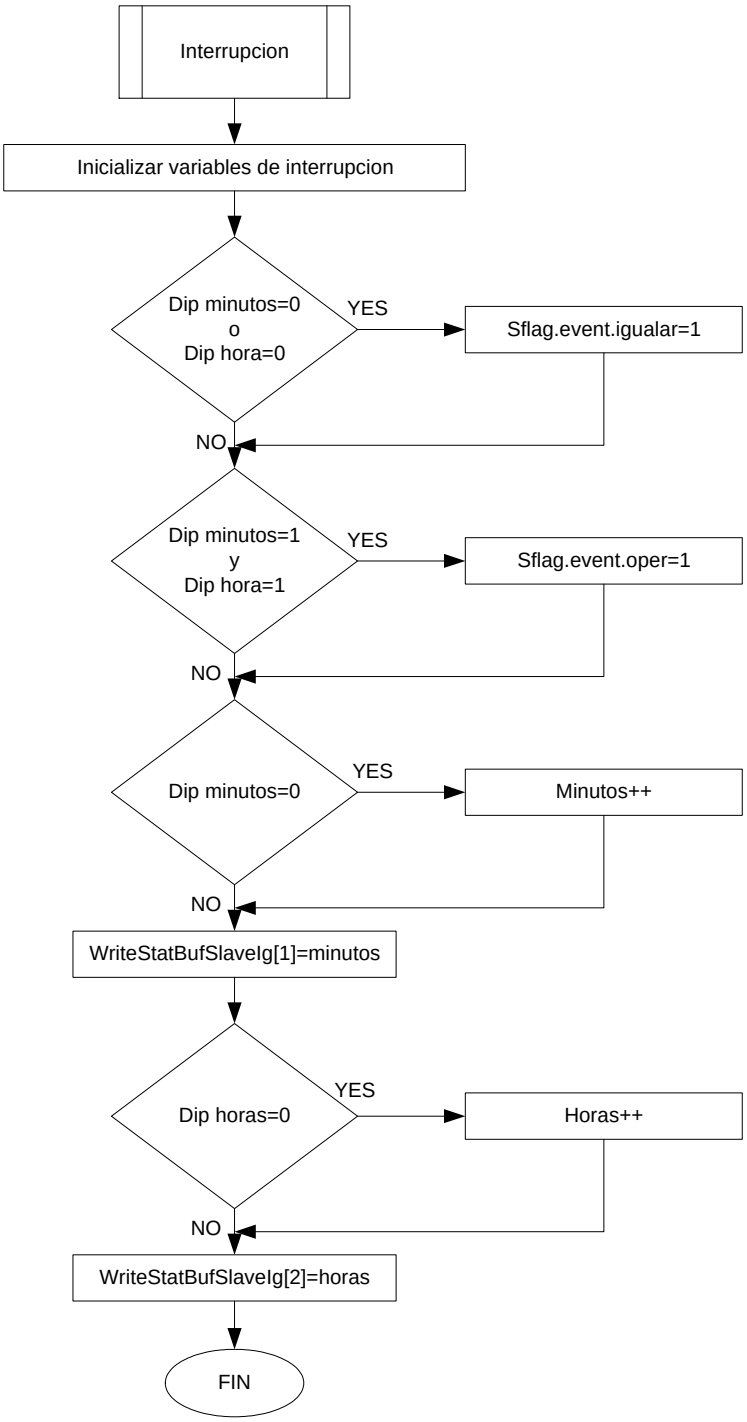

Ilustración 60 Diagrama de flujo de la interrupción del programa principal

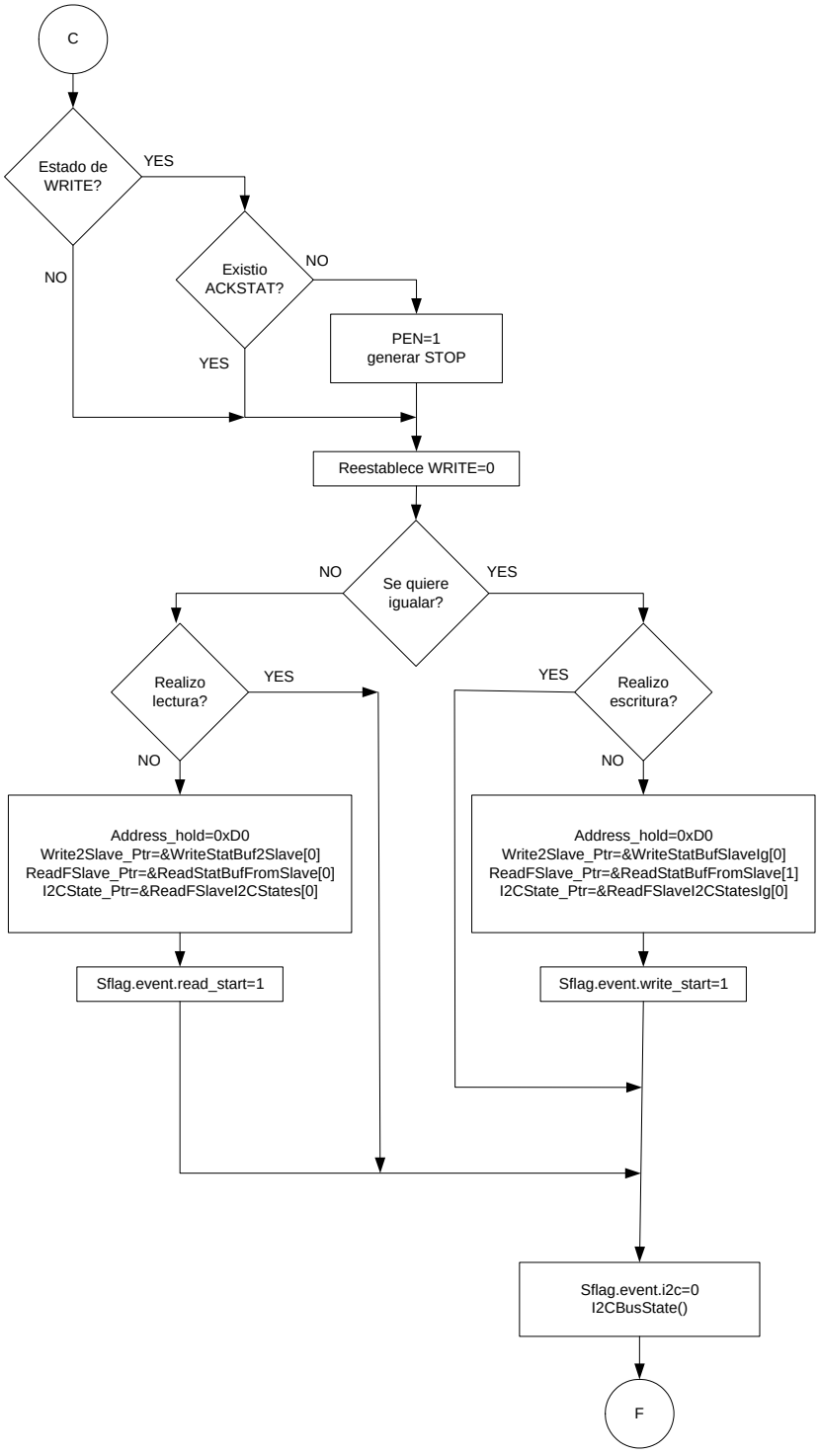

Ilustración 61 Diagrama de flujo del programa Comm_i2c.C

4.6.3 Funcionamiento del Programa.

Obsérvese bien la variable WriteStatBufSlaveIg que contiene la dirección del registro desde el cual se va a empezar a modificar, en este caso se modificaran los minutos y las horas, por lo tanto la dirección inicial debe ser la 0x01H, una vez que se accede al registro, el puntero modifica ese registro y pasa automáticamente al siguiente (0x02H), correspondiente al de horas.

La variable ReadFSlaveI2CstatesIg realiza la escritura de dos datos en el registro 0x01H para luego proceder a la lectura de los mismos con propósito de comprobación.

Todo este proceso se lo realiza mediante comunicación con el bus I2C, propiedad que posee el PIC 16F877. La información que se envía al bus produce eventos I2C mediante interrupciones, por lo tanto en el grafico inferior se puede observar ese conjunto de eventos.

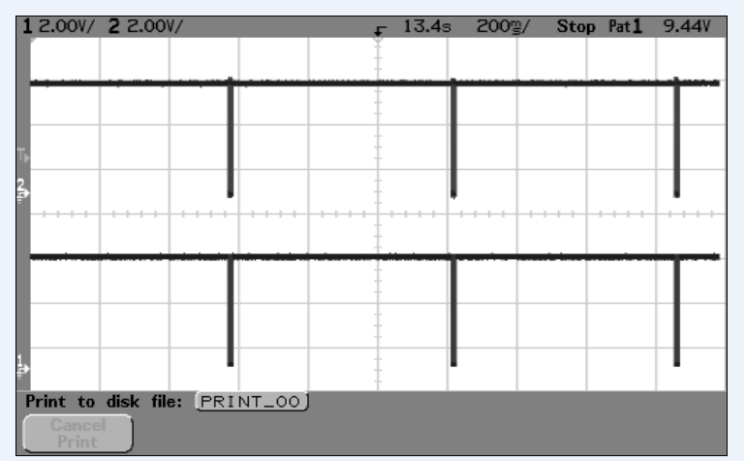

Ilustración 62 Conjunto de eventos I2C de Escritura

Se procede a hacer un análisis de un solo evento de los que se mostraron anteriormente, en donde la línea superior corresponde a la señal de datos que se transportan en el bus y la línea inferior es la señal de reloj o SCL.

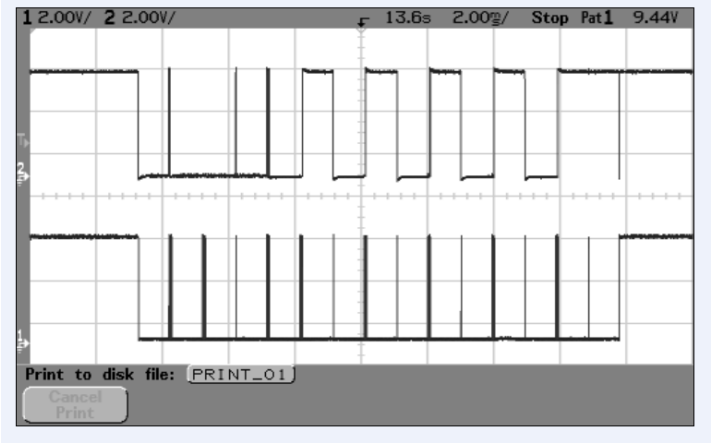

Ilustración 63 Evento I2C de Escritura

Para iniciar una comunicación I2C, es necesaria una señal de START o inicio, esto se produce cuando la línea SDA pasa a bajo mientras la líneas SCL se encuentra en alto.

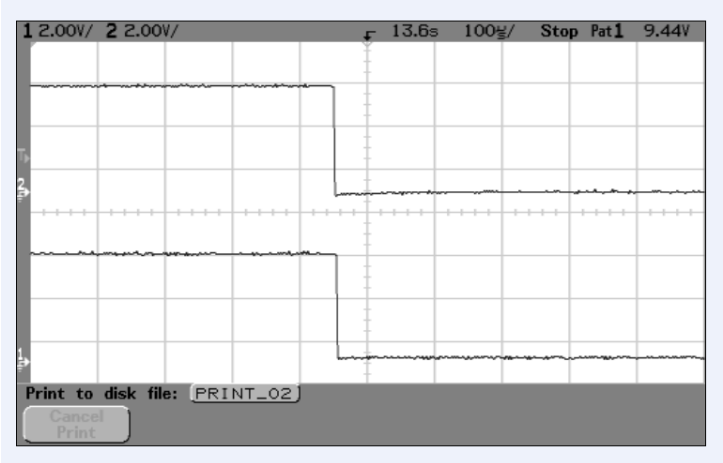

Ilustración 64 Señal de START

La siguiente señal en enviarse es la dirección del esclavo, que para el RTC es 0xD0H, seguida de la operación a realizarse, que para este caso es escritura (0). Se puede observar en la ilustración 65 que la dirección enviada es 0b11010000, equivalente a 0xD0.

Ilustración 65 Dirección del esclavo, 0xD0H

La señal de reconocimiento o ACK esta presente en la comunicación, por lo tanto el esclavo es reconocido por el maestro y la comunicación hasta el momento es correcta.

En la primera fase escritura solo se enceran los registros necesarios, como son los de minutos y horas, para luego iniciar la grabación de los nuevos valores ingresados mediante los DIP SWITCH, gracias a la interrupción generada.

Con el propósito de comprobación y presentación en el LCD, se leen los registros del RTC, ya modificados. En la Ilustración 66 se observa que luego de un RESTART, señal que tiene un propósito igual al de START, debe estar seguido de la dirección del esclavo, junto al bit que indica la operación a realizarse, lectura (1). Para este caso la señal debe ser 0xD1H.

La operación de lectura se realiza de los registros de segundos, minutos y horas. En la Ilustración 67 se observa el valor del registro de segundos, correspondiente a 0.

Ilustración 66 Dirección del esclavo con operación de lectura, 0xD1H

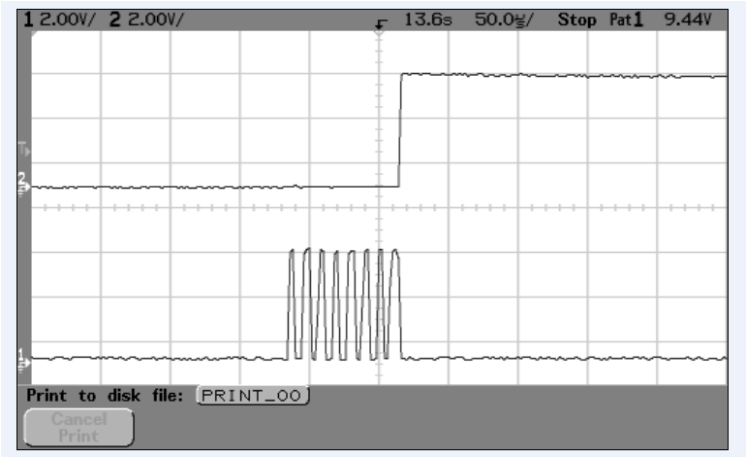

Ilustración 67 Valor del registro de segundos (0x00H)

Luego de cada dato enviado al bus se puede apreciar la señal de reconocimiento o ACK, que es un pulso en bajo de la línea de datos mientras la línea SCL o de reloj genera un pulso en alto.

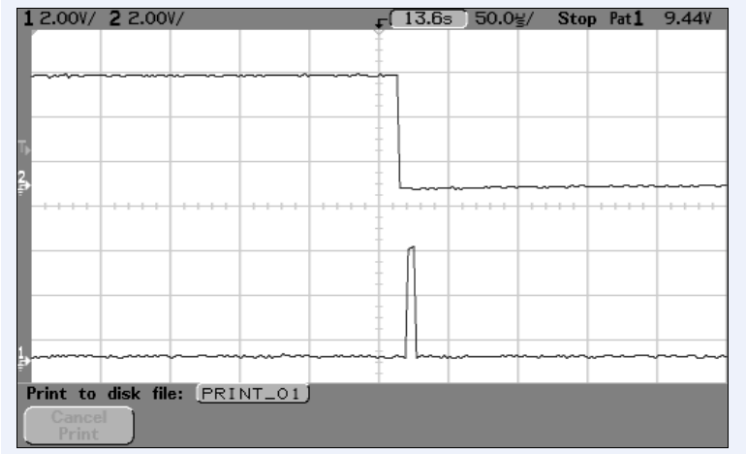

Ilustración 68 Señal de reconocimiento o ACK

El puntero se dirige a la siguiente posición de registro, que corresponde a la 0x01H, correspondiente al registro que maneja los minutos del reloj. En la ilustración 69 se observa el valor que contiene dicho registro: 7 minutos.

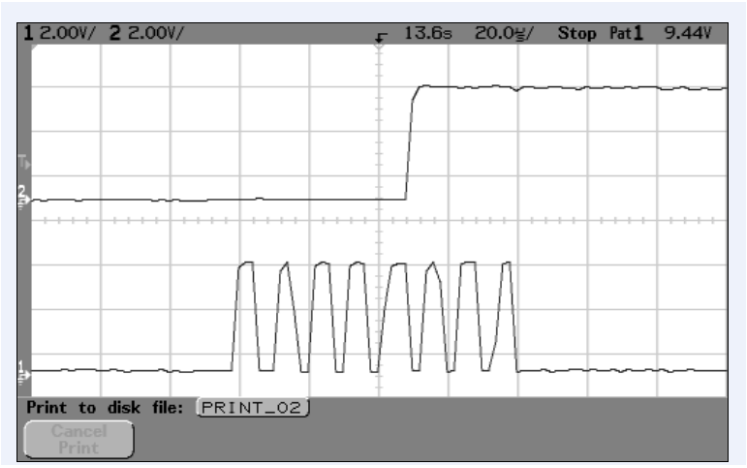

Ilustración 69 Valor del registro de minutos (0x01H)

El ultimo registro en ser leído es el correspondiente al de las horas, y este presenta un valor de 5 horas, 0b00000101.

Ilustración 70 Valor del registro de horas (0x02H)

Para finalizar la comunicación I2C, se genera la señal de STOP o parada, cuando la línea SDA pasa a alto mientras la línea SCL permanece en alto.

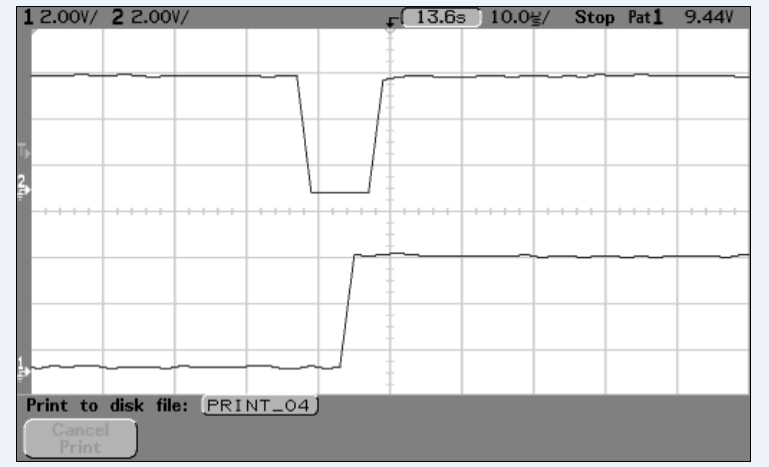

Ilustración 71 Señal de STOP

4.6.4 Programa No. 3

MAESTRO4.C

```
#include <pic.h>
#include "c:\ht-pic\samples\mas_i2c.h"
#include "c:\ht-pic\samples\init.c"
#include "c:\ht-pic\samples\com4_i2c.c"
```

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>

# define IG_MINUTOS   RB4      // PARA IGUALAR SE USA DIP1 Y DIP2
# define IG_HORAS     RB5
int minutos, horas, n1_dms;

void main(void)
{

//VARIABLES
unsigned char min[10];
unsigned char hora[10];
unsigned char seg[10];
unsigned char dato[10];

//INICIALIZACIONES
Init_Ports(); // inicializa Puertos
lcd_init(); // inicializa el LCD
Init_Ssp(); // initializa el módulo SSP
Init_Timer1(); // initializa el TMR1

//HABILITA INTERRUPCIONES
TMR1IE = 1; // interrupcion del  TMR1
SSPIE = 1; // interrupciones I2C
sflag.status = 0x0000; // todas las banderas de eventos son reseteadas
PEIE = 1; // enable peripheral interrupts
GIE=1;

//PRESENTACION EN LCD

lcd_clear();
lcd_goto(3);
lcd_puts("red I2C-WRITE");
lcd_goto(0x40);
lcd_puts("***Bienvenidos***");
DelayMs(250);
DelayMs(250);
```

```
DelayMs(250);
lcd_clear();
//sflag.event.i2c=1;

for(;;) // LAZO INFINITO
{
	if(sflag.event.i2c)//BANDERA DE EVENTO I2C
	{
		DelayUs(400);
		Service_I2CSlave();
	}

	if(sflag.event.writes_done)//IF PARA PRESENTACION
	{
		if(ReadStatBufFromSlave[0]>=60)
		{
			ReadStatBufFromSlave[0]=0;
			ReadStatBufFromSlave[1]++;
		}
		if(ReadStatBufFromSlave[1]>=60)
		{
			ReadStatBufFromSlave[1]=0;
			ReadStatBufFromSlave[2]++;
		}

		lcd_clear();
		lcd_goto(0x00);

	sprintf(dato,"%02d:%02d:%02d",ReadStatBufFromSlave[2],ReadStatBufFromSlave[1],Re
adStatBufFromSlave[0]);
		lcd_puts(dato);
		DelayMs(250);
		DelayMs(250);
	sflag.status=0x0000;
	TMR1ON=1;
	TMR1IE=1;

	}//FIN DE IF DE PRESENTACION
}//FIN DEL CICLO INFINITO
}//FIN DEL MAIN
```

```
void interrupt piv( void )
{
	if ( SSPIE && SSPIF ) // prueba de realizacion de evento I2C
	{
		SSPIF = 0;
		sflag.event.i2c = 1; // setea la bandera de evento
	}

	else if ( TMR1IE && TMR1IF ) //PRUEBA DE INTERRUPCION DEL TMR1
	{
	sflag.event.i2c = 1;

//REINICIALIZA VALORES DEL TMR1 PARA 100 MS

	TMR1IF = 0;
	TMR1L += 0x2C;
	TMR1H = 0xCF;
	TMR1ON=0;
	TMR1IE=0;

	if(IG_MINUTOS==0||IG_HORAS==0)//ESTADOS DEL DIP
	{
		n1_dms++;
		sflag.event.igualar=1;
	}//FIN DE ANALISIS DE DIP

	if(IG_MINUTOS==1&&IG_HORAS==1)//ESTADOS DEL DIP
	{
		sflag.event.oper=1;
	}

		if(IG_MINUTOS==0)
		{
			if(n1_dms==5)
		{
				n1_dms=0;
				minutos++;
			}//FIN DE MINUTOS++
			if(minutos>=60)
```

```
                        {
                                minutos=0;
                                horas++;
                        }
                }//FIN DE MINUTOS
                        WriteStatBufSlaveIg[1]=minutos;
        if(IG_HORAS==0)
                {
                        if(n1_dms==5)
                        {
                                n1_dms=0;
                        horas++;
                        }  //FIN DE HORAS
                        if(horas>=24)
                        {
                                horas=0;
                        }
        }//FIN DE HORAS
                WriteStatBufSlaveIg[2]=horas;
}//FIN DE PRUEBA DE INTERRUPCION DEL TMR1
}//fin de interrupciones
```

COMM4_I2C.C

```
#include <pic.h>
#include <stdio.h>
#include "c:\ht-pic\samples\comm_i2c.h"
#include "c:\ht-pic\samples\delay.h"
#include "c:\ht-pic\samples\lcd.c"

//FUNCION SERVICIO I2C
void Service_I2CSlave( void )
{               DelayUs(400);

if(sflag.event.igualar) //SI  SE QUIERE IGUALAR RELOJ
{
        if(!sflag.event.write_start)
        {
        address_hold=0xD0;
```

```
        read_count=2;
        if(!sflag.event.writes_done){
        Write2Slave_Ptr = &WriteStatBufSlaveIg[0];
        ReadFSlave_Ptr=&ReadStatBufFromSlave[1];
        I2CState_Ptr = &ReadFSlaveI2CStatesIg[0];
        DelayMs(11);
        }
        sflag.event.write_start=1;
        }

}//FIN DE SI IGUALAR RELOJ

if(sflag.event.oper){
                if(!sflag.event.read_start)//SI AUN NO EXISTE PETICION DE LECTURA
                {
                        address_hold=0xD0;
                        read_count=3;
                        if(!sflag.event.writes_done){
                        Write2Slave_Ptr = &WriteStatBuf2Slave[0];
                        ReadFSlave_Ptr = &ReadStatBufFromSlave[0];
                        I2CState_Ptr = &ReadFSlaveI2CStates[0];
                        DelayMs(11);
                        }
                        sflag.event.read_start=1;

                }// FIN INICIALIZO PUNTEROS
}//FIN DE OPERACION
if (sflag.event.write_state)//PRUEBA SI EL ULTIMO ESTADO I2C FUE ESCRITURA
{
        if ( ACKSTAT ) // was NOT ACK received?
        {
        PEN = 1; // generate bus stop condition
        eflag.i2c.ack_error = 1; // set acknowledge error flag
        }//FIN PRUEBA DE ACK
        sflag.event.write_state = 0; // reset write state flag

}//FIN ULTIMO ESTADO ESCRITURA

        sflag.event.i2c=0;
        I2CBusState(); // execute next I2C state
```

```
}//FIN DE FUNCION SERVICIO I2C

//FUNCION ESTADOS DE BUS
void I2CBusState ( void )
{

i2cstate = *I2CState_Ptr++; //ACTULIZA PUNTERO CON SIGUIENTE ESTADO

switch ( i2cstate ) //EVALUA EL ESTADO
{

        case ( READ ): //LECTURA
                RCEN = 1; //
                break;
        case ( WRITE_DATA ): // ESCRITURA
                SSPBUF = *Write2Slave_Ptr++; //ALMACENA EN BUFFER
                sflag.event.write_state = 1;
                break;
        case ( WRITE_ADDRESS1 ): //(R/W=1)
                SSPBUF = address_hold + 1;
                sflag.event.write_state = 1;
                break;
        case ( START ): // START
                SEN = 1;
                break;
        case ( WRITE_ADDRESS0 ): //(R/W=0)
                SSPBUF = address_hold;
                sflag.event.write_state = 1;
                break;
        case ( SEND_ACK ): // ACK
                *ReadFSlave_Ptr++ = SSPBUF;
                if ( read_count > 0)
                {
                        read_count -= 1;
                        I2CState_Ptr -= 2;
                }
                ACKDT = 0;
                ACKEN = 1;
                break;
```

```
        case ( SEND_NACK ): //NACK
                *ReadFSlave_Ptr = SSPBUF;
                ACKDT = 1;
                ACKEN = 1;
                break;
        case ( STOP ):STOP
                PEN = 1;
                sflag.event.next_i2cslave = 1;
                sflag.event.writes_done = 1;
                break;
        case ( RESTART ):
                RSEN = 1;
                break;
        default:
                break;
}//FIN EVALUAR ESTADO
}//FIN FUNCION DE ESTADOS
```

COMM_I2C.H

```
// DEFINICION DE FUNCIONES
void I2CBusState( void );

//DEFINICION DE VARIABLES
unsigned char address_hold, read_count;
unsigned char i2cstate;
unsigned char *Write2Slave_Ptr;
bank1 unsigned char ReadStatBufFromSlave[3];
bank1 unsigned char *ReadFSlave_Ptr;

unsigned char WriteStatBufSlaveIni[4]={0x00,0x00,0x07,0x10};  // Inicio
unsigned char WriteStatBufSlaveIg[4]={0x01,0x00,0x00,0x01};  // Escritura
unsigned char WriteStatBuf2Slave[1]={0x00} ;  // Lectura

// define estados I2C
enum i2c_bus_states{ START =1, RESTART =2, STOP =3, SEND_ACK =4, SEND_NACK =5,
GEN_CALL =6, READ =7, WRITE_DATA =8, WRITE_ADDRESS1 =9,
WRITE_ADDRESS0 =10 };
```

```
const unsigned char ReadFSlaveI2CStatesIni[]={1,10,8,8,2,10,8,8,3,0};  // Inicio
const unsigned char ReadFSlaveI2CStatesIg[]={1,10,8,8,8,2,10,8,2,9,7,4,7,5,3,0};  //Escritura
const unsigned char ReadFSlaveI2CStates[] = {1,10,8,2,9,7,4,7,5,3,0};  // Lectura

const unsigned char *I2CState_Ptr; // puntero para acceder a los estados I2C
```

4.7 Comunicación PIC-PIC con presentación en matriz de leds

Elaborar un programa que permita la comunicación entre dos PIC's. La información a enviarse serán caracteres ASCII. El maestro estará conectado a la PC, desde donde recibirá texto. El esclavo dispone de 4 matrices de leds para presentar la información transmitida.

4.7.1 Información Requerida

Se realizara una comunicación serial entre el PIC maestro y el PC. Para luego mediante el bus de comunicación I2C transferir dicha información al PIC esclavo que será el encargado de la presentación de dicha información en las matrices de leds que tendrá bajo su cargo.

Las matrices de leds cumplirán la función de cartel electrónico, en el cual se podrá visualizar la información que se envía a través del bus I2C. Se deberá tomar en cuenta el tamaño que deben tener, así como el color, el brillo, etc.

Para la implementación del modelo se emplearan transistores, que permitirán el aumento de corriente en las matrices, pues su intensidad disminuye a medida que se aumenta la cantidad.

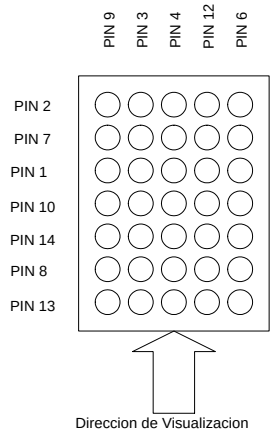

Ilustración 72 Configuración de los pines de la matriz de leds

Las matrices a emplear son de 5x7 cuya configuración se determina en las Ilustraciones 72 y 73.

Ilustración 73 Orden de los pines de la matriz de leds

4.7.2 Diagrama General de Funcionamiento

Se realizara comunicación USART entre el PC y el PIC Maestro, para luego por medio de las líneas SDA y SCL, pertenecientes al Bus de Comunicación I2C, se envíe esa información al PIC esclavo.

El PIC esclavo deberá presentar la información (4 letras mayúsculas o minúsculas) en 4 matrices de leds.

Las 4 letras se ingresaran por teclado y serán reconocidas únicamente cuando se presione la tecla ENTER. En ese momento se producirá una interrupción por evento I2C.

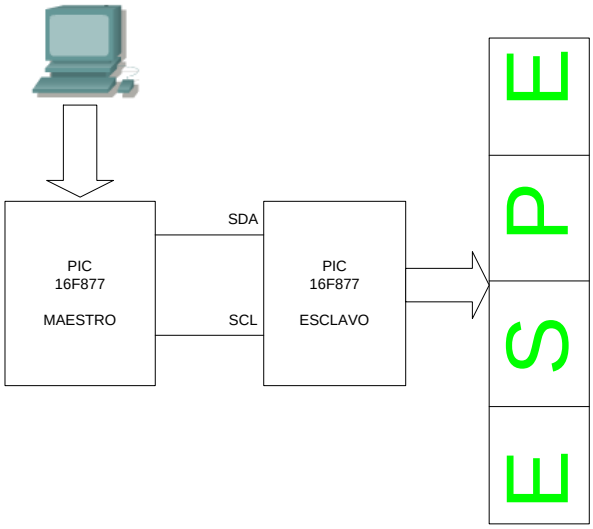

Ilustración 74 Diagrama General de funcionamiento

4.7.3 Diagramas de Flujo

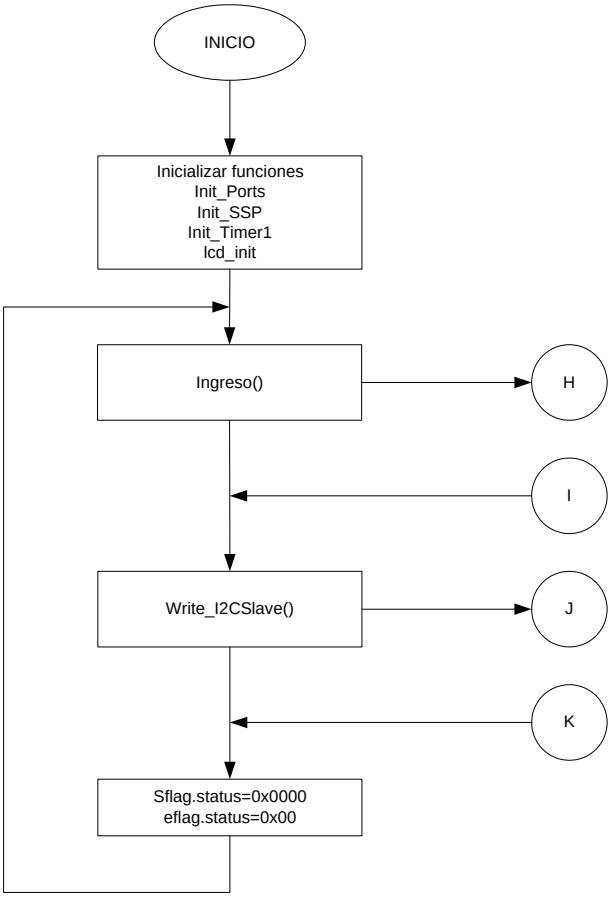

Ilustración 75 Diagrama de Flujo del programa Matriz.C

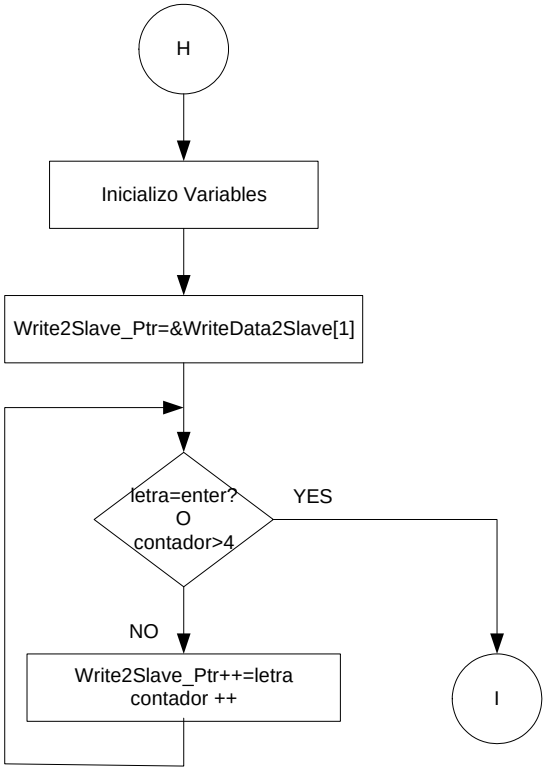

Ilustración 76 Diagrama de flujo de la función Ingreso del programa principal

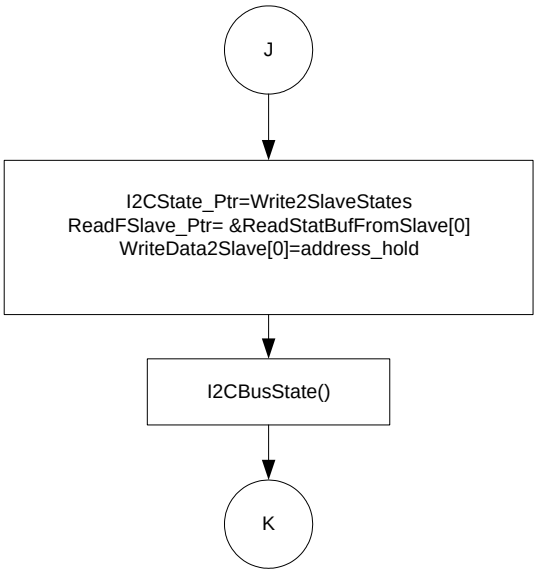

Ilustración 77 Diagrama de Flujo-función Write_I2CSlave()-programa principal

4.7.4 Funcionamiento del Programa

Por default el programa presentara el mensaje “ESPE” antes de cualquier otra información.

Usando el programa Hyperterminal se procederá a la comunicación entre el PC y el PIC maestro, determinando ciertas características especiales en la sesión de comunicación. Al observar la siguiente ilustración se conocerán los parámetros que deben ser establecidos.

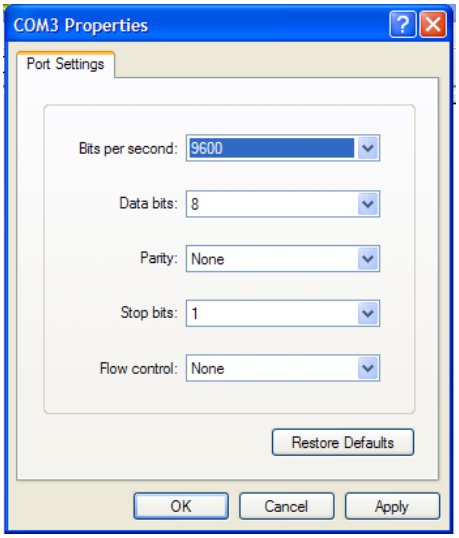

Ilustración 78 Configuración de la sesión de hyperterminal comunicación PC - PIC

La velocidad de transmisión se determina como 9600 baudios y no existe control de flujo.

Una vez configurada la sesión de comunicación se establece la conexión y se ingresan los 4 datos, como se muestra en la Ilustración 79.

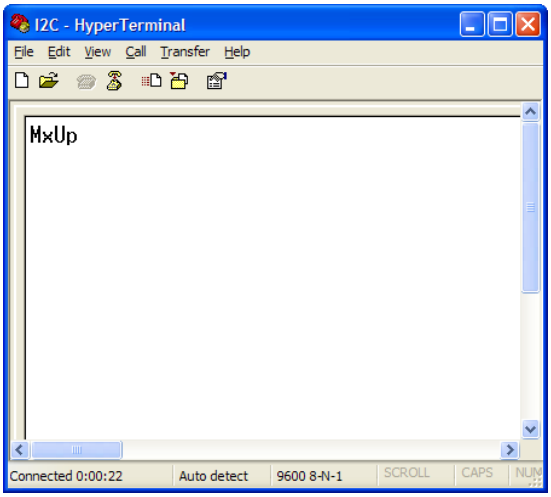

Ilustración 79 Conexión establecida con el PIC

Como ya se conoce es necesaria una señal de START para toda comunicación, que se generara cuando se presione la tecla ENTER desde el teclado. Entonces la comunicación I2C iniciara.

Se envía la dirección del elemento esclavo junto a la operación a realizarse y la señal de reconocimiento o ACK, la que indica que el esclavo es reconocido.

En código ASCII se conoce que las letras M, x, U y p tienen un valor de 0x4DH, 0x78H, 0x55H y 0x70H respectivamente.

Usando el osciloscopio digital se puede observar estas señales que son transmitidas a través del bus I2C hacia el PIC esclavo.

El primer dato enviado es la letra mayúscula “M”, cuyo código ASCII hexadecimal es 0x4DH, este se lo puede observar en la Ilustración 80.

Ilustración 80 Código ASCII de la letra "M"

La segunda letra es la “x” minúscula, cuyo código ASCII es 0x78H.

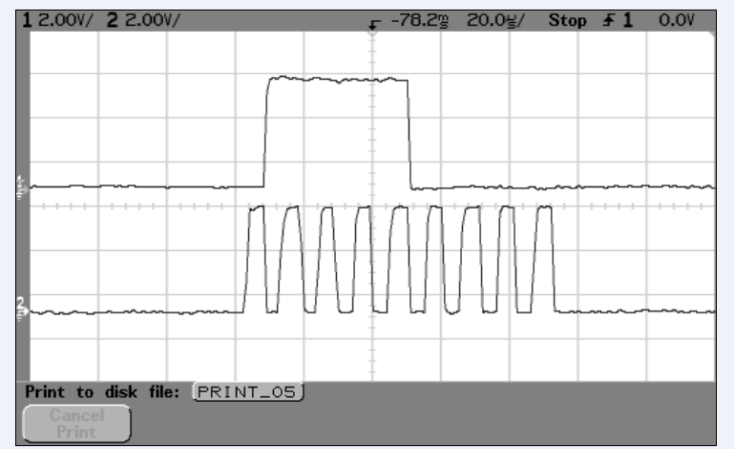

Ilustración 81 Código ASCII de la letra "x"

La tercer letra es la mayúscula “U”, el código ASCII es 0x55H, su valor en binario es 0b01010101. se puede ver en la Ilustración 82.

La ultima letra enviada al bus es la “p” minúscula, el código binario de la letra es 0b01110000, en hexadecimal es la 0x70H, Ilustración 83.

Ilustración 82 Código ASCII de la letra "U"

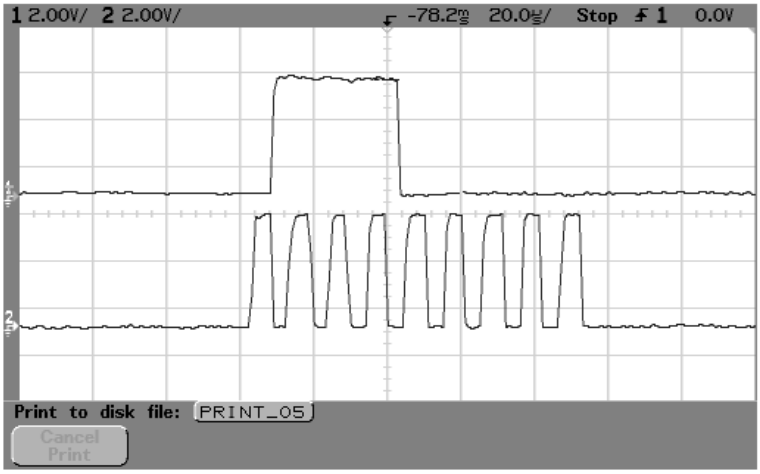

Ilustración 83 Código ASCII de la letra "p"

Cada uno de estos datos están seguidos de la señal de reconocimiento o acknowledge, que demuestra la correcta comunicación I2C entre los elementos del sistema.

Para finalizar la comunicación se genera una señal de parada o STOP.

Con respecto al programa del PIC esclavo, se debe tener en cuenta que los valores correspondientes a los cuatro datos que se envían desde el PC a través del Bus I2C, se guardan en direcciones de memoria del PIC, siendo estas direcciones la 0x23, 0x24,0x25 y 0x26. por lo tanto cuando se desea decodificar

estas señales para presentarlas en las matrices, se evalúan cada una de ellas mediante un barrido de filas.

4.7.5 Programa No. 4

MATRIZ.C

```
#include <pic.h> // processor if/def file
#include "c:\ht-pic\samples\matriz.h"
#include "c:\ht-pic\samples\init.c"
#include "c:\ht-pic\samples\i2c_mat.h"
#include "c:\ht-pic\samples\delay.h"
#include "c:\ht-pic\samples\lcd.h"
#include "c:\ht-pic\samples\lcd.c"
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

void ingreso();
void Write_I2CSlave( void );
unsigned int Calc_Checksum( unsigned char *ptr, unsigned char length );
void I2CBusState ( void );

void main(void)
{/* Inicialización */
Init_Usart(); // inicializa USART
Init_Ports(); // inicializa Puertos
Init_Ssp(); // inicializa SSP
lcd_init();//inicializa lcd
lcd_puts("HOLA"); // para comprobación
DelayMs(250);
DelayMs(250);
while(1){
ingreso();
Write_I2CSlave();
sflag.status = 0x0000;
eflag.status = 0x00;
```

```
}
}//fin del main

void ingreso(){
unsigned char dat[8];
unsigned char tecla='n';
int contador=0;
Write2Slave_Ptr=&WriteData2Slave[1];

                while(tecla!='\r'){
                        while(!RCIF){}
                        tecla=getch();
                        contador++;
                        *Write2Slave_Ptr++=tecla;
                        if(tecla=='\r'||contador>4)
                        break;
                        putch(tecla);
                        lcd_goto(0x40);
                        sprintf(dat,"%d",tecla);
                        lcd_puts(dat);
                }
                tecla='n';
}//fin de ingreso

void Write_I2CSlave( void )
{
unsigned char *temp_ptr; // define auto variable
sflag.event.writes_done = 0; // asegura que la bandera sea reseteada.
I2CState_Ptr = Write2SlaveStates; // inicializa puntero de estados I2C
ReadFSlave_Ptr = &ReadStatBufFromSlave[0];
Write2Slave_Ptr = &WriteData2Slave[1]; // inicializa puntero
        do
        {DelayUs( 400 );
        SSPIF = 0;
                I2CBusState();
                while ( !SSPIF);
                if ( sflag.event.write_state )
                {
                        if ( ACKSTAT )
                        {
```

```
                    PEN = 1;
                    sflag.event.writes_done = 1;
                }
                sflag.event.write_state = 0;
            }
        } while( !sflag.event.writes_done );
}

void I2CBusState ( void )
{
i2cstate = *I2CState_Ptr++;
        switch ( i2cstate )
        {
        case ( READ ):
            RCEN = 1;
            break;
        case ( WRITE_DATA ):
            SSPBUF = *Write2Slave_Ptr++;
            sflag.event.write_state = 1;
            break;
        case ( WRITE_ADDRESS1 ):
            SSPBUF = address_hold + 1;
            sflag.event.write_state = 1;
            break;
        case ( START ):
            SEN = 1;
            break;
        case ( WRITE_ADDRESS0 ):
            SSPBUF = address_hold;
            sflag.event.write_state = 1;
            break;
        case ( SEND_ACK ):
            *ReadFSlave_Ptr++ = SSPBUF;
            if ( read_count > 0)
            {
                read_count -= 1;
                I2CState_Ptr -= 2;
            }
            ACKDT = 0;
            ACKEN = 1;
```

```
                break;
        case ( SEND_NACK ):
                *ReadFSlave_Ptr = SSPBUF;
                ACKDT = 1;
                ACKEN = 1;
                break;
        case ( STOP ):
                PEN = 1;
                sflag.event.next_i2cslave = 1;
                sflag.event.writes_done = 1;
                break;
        case ( RESTART ):
                RSEN = 1;
                break;
        default:
                break;
        }
}
```

Programa Esclavo.

Slavemat.C

```
static volatile unsigned char      DIREC @ 0x20;
static volatile unsigned char      LONGI @ 0x21;
static volatile unsigned char      OFFSE @ 0x22;
static volatile unsigned char      DATO1 @ 0x23;
static volatile unsigned char      DATO2 @ 0x24;
static volatile unsigned char      DATO3 @ 0x25;
static volatile unsigned char      DATO4 @ 0x26;
#define FOSC (4000000L) // define external clock frequency
#define CCP_HBYTE 0x03
#define CCP_LBYTE 0xe8
#define COUNT_10MSEC 10
#define COUNT_100MSEC 10
#define COUNT_1000MSEC 10
#define TEMP_OFFSET 58
#define NODE_ADDR 0x18
#define ADRES ADRESH
```

```
#define ON 1
#define TRUE 1
#define OFF 0
#define FALSE 0

/*---------------------------------------------------------------------
 Definiciones de longitud de buffer
---------------------------------------------------------------------*/
#define RX_BUF_LEN 8
#define SENS_BUF_LEN 12
#define CMD_BUF_LEN 4

/*---------------------------------------------------------------------
 Valores de indices del buffer Receptor
---------------------------------------------------------------------*/
#define SLAVE_ADDR 0
#define DATA_OFFS 2
#define DATA_LEN 1
#define RX_DATA 3

/*---------------------------------------------------------------------
 Valores de indices del buffer del sensor
---------------------------------------------------------------------*/
#define COMM_STAT 0
#define SENSOR_DATA 3
#define STATUS0 1
#define STATUS1 2
#define TEMP0 3
#define TACH0 4
#define ADRES0 5
#define ADRES1 6
#define ADRES2 7
#define ADRES3 8
#define TACH1 9
#define TACH2 10
#define TACH3 11

/*---------------------------------------------------------------------
 Valores de indices del buffer de comandos
---------------------------------------------------------------------*/
```

```
#define CMD_BYTE0 0
#define CMD_BYTE1 1
#define CMD_BYTE2 2
#define CMD_BYTE3 3

/*-------------------------------------------------------------------
 Definiciones de pines
//-------------------------------------------------------------------*/

#define TACH_IN0 0x10
#define TACH_IN1 0x20
#define TACH_IN2 0x40
#define TACH_IN3 0x80
#define LED_0 RB0
#define LED_1 RB1
#define FAN_CNTRL RC2

/*-------------------------------------------------------------------
// Archivos Include
//-------------------------------------------------------------------*/
#include <pic.h>
#include <delay.h>
#include <delay.c>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/*-------------------------------------------------------------------
 Prototipos de funciones
//-------------------------------------------------------------------*/
void Setup(void);
interrupt void ISR_Handler(void);
void WriteI2C(char data);
char ReadI2C(void);
void SSP_Handler(void);
```

```
/*-------------------------------------------------------------------
//Declaracion de variables
//-------------------------------------------------------------------*/

unsigned char Count_10m,
Count_100m,
Count_1000m,
Count_Tach0,
Count_Tach1,
Count_Tach2,
Count_Tach3;
extern bank1 char RXBuffer[RX_BUF_LEN];
char CmdBuf[CMD_BUF_LEN];
char RXChecksum;
unsigned char
RXBufferIndex,
RXByteCount,
SensBufIndex,
CmdBufIndex,
PORTBold,
temp;
union INTVAL
{
char b[2];
int i;
}
union INTVAL TXChecksum;
union SENSORBUF
{
struct{
unsigned chkfail:1;
unsigned rxerror:1;
unsigned ovflw:1;
unsigned sspov:1;
unsigned bit4:1;
unsigned bit5:1;
unsigned bit6:1;
unsigned r_w:1;
} comm_stat ;
unsigned char b [SENS_BUF_LEN];
```

```
} SensorBuf ;
struct{
unsigned msec10:1;
unsigned msec100:1;
unsigned msec1000:1;
unsigned bit3:1;
unsigned wdtdis:1;
unsigned bit5:1;
unsigned bit6:1;
unsigned bit7:1;
} stat ;

/*-------------------------------------------------------------------
// codigo de interrupcion
//-------------------------------------------------------------------*/
interrupt void ISR_Handler(void)
{
if(SSPIF)
{
//LED_0 = ON;
SSPIF = 0;
SSP_Handler();
}
}

/*-------------------------------------------------------------------
// void SSP_Handler(void)
//-------------------------------------------------------------------*/

void SSP_Handler(void)
{
        char RXBuffer[RX_BUF_LEN];
        unsigned char i,j;

/*-------------------------------------------------------------------
// ESTADO 1: Operación de escritura y el ultimo byte es una direccion
//-------------------------------------------------------------------*/

if(!STAT_DA && !STAT_RW && STAT_BF && STAT_S)
{
```

```
CLRWDT();
stat.wdtdis = 1;
RXBufferIndex = SLAVE_ADDR;
RXByteCount = 0;
RXBuffer[RXBufferIndex] = ReadI2C();

DIREC=RXBuffer[0];

RXChecksum = RXBuffer[RXBufferIndex];
RXBufferIndex++;
SensorBuf.b[COMM_STAT] = 0x02;

if(SSPOV)
        {
        SensorBuf.comm_stat.sspov = 1;
        SSPOV = 0;
        }
}

/*-------------------------------------------------------------------------
// ESTADO 2: Operación de escritura y el ultimo byte es un dato.
//-------------------------------------------------------------------------*/

else if(STAT_DA && !STAT_RW && STAT_BF)
{
        if(RXBufferIndex == RX_BUF_LEN)
        {
                SensorBuf.comm_stat.ovflw = 1;
                RXBufferIndex = RX_BUF_LEN - 1;
        }
        if(SSPOV)
        {
                SensorBuf.comm_stat.sspov = 1;
                SSPOV = 0;
        }

        RXBuffer[RXBufferIndex]= ReadI2C();
        LONGI=RXBuffer[1];
        OFFSE=RXBuffer[2];
        DATO1=RXBuffer[3];
```

```
DATO2=RXBuffer[4];
DATO3=RXBuffer[5];
DATO4=RXBuffer[6];

RXChecksum += RXBuffer[RXBufferIndex];

if(RXBufferIndex == DATA_LEN)
{
	if(RXBuffer[DATA_LEN] & 0x80)
	{
		RXBuffer[DATA_LEN] &= 0x7f;
		SensorBuf.comm_stat.r_w = 1;
		RXByteCount = 3;
	}
else
{
	SensorBuf.comm_stat.r_w = 0;
	RXByteCount = RXBuffer[DATA_LEN] + 3;
	if(RXByteCount > RX_BUF_LEN)
	{
		SensorBuf.comm_stat.rxerror = 1;
		SensorBuf.comm_stat.ovflw = 1;
	}
}
}

else if(RXBufferIndex == DATA_OFFS)
{
	if(SensorBuf.comm_stat.r_w)
	{
		if(RXBuffer[DATA_LEN] + RXBuffer[DATA_OFFS] > SENS_BUF_LEN -
1)
		{
			SensorBuf.comm_stat.rxerror = 1;
			SensorBuf.comm_stat.ovflw = 1;
		}
		else
		{
		SensorBuf.comm_stat.rxerror = 0;
		SensorBuf.comm_stat.ovflw = 0;
```

```
                }
            }
            else
            {
                if(RXBuffer[DATA_LEN] + RXBuffer[DATA_OFFS] > CMD_BUF_LEN - 1)
                {
                    SensorBuf.comm_stat.rxerror = 1;
                    SensorBuf.comm_stat.ovflw = 1;
                }
                else
                {
                    SensorBuf.comm_stat.rxerror = 0;
                    SensorBuf.comm_stat.ovflw = 0;
                }
            }
        }

        else if(RXBufferIndex == RXByteCount)
        {
            if(SensorBuf.comm_stat.r_w)
            {
                if(RXChecksum)
                SensorBuf.comm_stat.chkfail = 1;
            else
                SensorBuf.comm_stat.chkfail = 0;
            }
        else
        {
            if(RXChecksum)
                SensorBuf.comm_stat.chkfail = 1;
            else
            {
            for(i=RXBuffer[DATA_OFFS]+3,                j                =                0;
i<(RXBuffer[DATA_LEN]+RXBuffer[DATA_OFFS]+3); i++,j++)
            {
                if(j == CMD_BUF_LEN) j--;
                    CmdBuf[j] = RXBuffer[i];
            }
            SensorBuf.comm_stat.chkfail = 0;
            }
```

```
        }
        }
        else;
        RXBufferIndex++;
        }

/*-------------------------------------------------------------------
// ESTADO 3: Operación de lectura y el ultimo byte fue una direccion
//-------------------------------------------------------------------*/

else if(!STAT_DA && STAT_RW && !STAT_BF && STAT_S)
{
        SensBufIndex = 0;
        TXChecksum.i = (int)SensorBuf.b[COMM_STAT];

        WriteI2C(SensorBuf.b[COMM_STAT]);
}

/*-------------------------------------------------------------------
// ESTADO 4: operación de lectura y el ultimo byte recibido fue un dato
//-------------------------------------------------------------------*/

else if(STAT_DA && STAT_RW && !STAT_BF)
{

        if(SensBufIndex < RXBuffer[DATA_LEN])
        {
                WriteI2C(SensorBuf.b[SensBufIndex + RXBuffer[DATA_OFFS]]);
                TXChecksum.i += (int)ReadI2C();
                SensBufIndex++;
        }
        else
                if(SensBufIndex == RXBuffer[DATA_LEN])
                {
                        TXChecksum.i = ~TXChecksum.i;
                        TXChecksum.i++;
                        WriteI2C(TXChecksum.b[0]);
                        SensBufIndex++;
                }
                else
```

```
                    if(SensBufIndex == (RXBuffer[DATA_LEN] + 1))
                    {
                        WriteI2C(TXChecksum.b[1]);
                        SensBufIndex++;
                    }
                    else
                    {
                        WriteI2C(0x55);
                    }
                }

/*-----------------------------------------------------------------------
// ESTADO 5: una señal de NACK del maestro es usado para indicar que una
// transmisión completa ha sido efectuada. El borrar elWDT es rehabilitado en el lazo de
// codigo principal.
//-----------------------------------------------------------------------*/

else if(STAT_DA && !STAT_RW && !STAT_BF)
{

    stat.wdtdis = 0;
    CLRWDT();
}
else;
}

/*-----------------------------------------------------------------------
// void WriteI2C(char data)
//-----------------------------------------------------------------------*/

void WriteI2C(char data)
{
    do
    {

        WCOL = 0;
        SSPBUF = data;
    } while(WCOL);
    CKP = 1;
}
```

```
/*-------------------------------------------------------------------
// char ReadI2C(void)
//-------------------------------------------------------------------*/

char ReadI2C(void)
{

return(SSPBUF);
}

//FUNCION DE DECODIFICACION DE LETRAS
unsigned char leds5 (unsigned char letra,unsigned char row)
{
        unsigned char fila_pos;

        switch (letra)
    {
        case 'A':     switch (row){
                case   1: fila_pos=0b00011111;break;
                case   2: fila_pos=0b00010001;break;
                case   3: fila_pos=0b00010001;break;
                case   4: fila_pos=0b00011111;break;
                case   5: fila_pos=0b00010001;break;
                case   6: fila_pos=0b00010001;break;
                case   7: fila_pos=0b00010001;break;
                default : break;
          }
          break;

        case 'B':     switch (row){
                case   1: fila_pos=0b00011110;break;
                case   2: fila_pos=0b00010001;break;
                case   3: fila_pos=0b00010001;break;
                case   4: fila_pos=0b00011110;break;
                case   5: fila_pos=0b00010001;break;
                case   6: fila_pos=0b00010001;break;
                case   7: fila_pos=0b00011110;break;
                default : break;
          }
          break;
```

```
case 'C':      switch (row){
           case   1: fila_pos=0b00011111;break;
           case   2: fila_pos=0b00010000;break;
           case   3: fila_pos=0b00010000;break;
           case   4: fila_pos=0b00010000;break;
           case   5: fila_pos=0b00010000;break;
           case   6: fila_pos=0b00010000;break;
           case   7: fila_pos=0b00011111;break;
           default : break;
  }
  break;

case 'D':      switch (row){
      case   1: fila_pos=0b00011110;break;
      case   2: fila_pos=0b00010001;break;
       case   3: fila_pos=0b00010001;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00010001;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00011110;break;
  default : break;
  }
  break;

case 'E':      switch (row){
      case   1: fila_pos=0b00011111;break;
      case   2: fila_pos=0b00010000;break;
       case   3: fila_pos=0b00010000;break;
       case   4: fila_pos=0b00011111;break;
       case   5: fila_pos=0b00010000;break;
       case   6: fila_pos=0b00010000;break;
       case   7: fila_pos=0b00011111;break;
  default : break;
  }
  break;

case 'F':      switch (row){
      case   1: fila_pos=0b00011111;break;
      case   2: fila_pos=0b00010000;break;
```

```
        case   3: fila_pos=0b00010000;break;
        case   4: fila_pos=0b00011110;break;
        case   5: fila_pos=0b00010000;break;
        case   6: fila_pos=0b00010000;break;
        case   7: fila_pos=0b00010000;break;
    default : break;
    }
    break;

case 'G':      switch (row){
       case   1: fila_pos=0b00011111;break;
       case   2: fila_pos=0b00010000;break;
        case   3: fila_pos=0b00010000;break;
        case   4: fila_pos=0b00011110;break;
        case   5: fila_pos=0b00010001;break;
        case   6: fila_pos=0b00010001;break;
        case   7: fila_pos=0b00011111;break;
    default : break;
    }
    break;

case 'H':      switch (row){
       case   1: fila_pos=0b00010001;break;
       case   2: fila_pos=0b00010001;break;
        case   3: fila_pos=0b00010001;break;
        case   4: fila_pos=0b00011111;break;
        case   5: fila_pos=0b00010001;break;
        case   6: fila_pos=0b00010001;break;
        case   7: fila_pos=0b00010001;break;
    default : break;
    }
    break;

case 'I':      switch (row){
       case   1: fila_pos=0b00011111;break;
       case   2: fila_pos=0b00000100;break;
        case   3: fila_pos=0b00000100;break;
        case   4: fila_pos=0b00000100;break;
        case   5: fila_pos=0b00000100;break;
        case   6: fila_pos=0b00000100;break;
```

```
      case   7: fila_pos=0b00011111;break;
  default : break;
  }
  break;

case 'J':     switch (row){
     case   1: fila_pos=0b00011111;break;
     case   2: fila_pos=0b00000100;break;
      case   3: fila_pos=0b00000100;break;
      case   4: fila_pos=0b00000100;break;
      case   5: fila_pos=0b00000100;break;
      case   6: fila_pos=0b00000100;break;
      case   7: fila_pos=0b00011100;break;
  default : break;
  }
  break;

case 'K':     switch (row){
     case   1: fila_pos=0b00010001;break;
     case   2: fila_pos=0b00010001;break;
      case   3: fila_pos=0b00010010;break;
      case   4: fila_pos=0b00011100;break;
      case   5: fila_pos=0b00010010;break;
      case   6: fila_pos=0b00010001;break;
      case   7: fila_pos=0b00010001;break;
  default : break;
  }
  break;

case 'L':     switch (row){
     case   1: fila_pos=0b00010000;break;
     case   2: fila_pos=0b00010000;break;
      case   3: fila_pos=0b00010000;break;
      case   4: fila_pos=0b00010000;break;
      case   5: fila_pos=0b00010000;break;
      case   6: fila_pos=0b00010000;break;
      case   7: fila_pos=0b00011111;break;
  default : break;
  }
  break;
```

```
case 'M':      switch (row){
      case   1: fila_pos=0b00011011;break;
      case   2: fila_pos=0b00010101;break;
       case   3: fila_pos=0b00010101;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00010001;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00010001;break;
  default : break;
  }
  break;

case 'N':      switch (row){
      case   1: fila_pos=0b00011001;break;
      case   2: fila_pos=0b00010101;break;
       case   3: fila_pos=0b00010011;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00010001;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00010001;break;
  default : break;
  }
  break;

case 'O':      switch (row){
      case   1: fila_pos=0b00001110;break;
      case   2: fila_pos=0b00010001;break;
       case   3: fila_pos=0b00010001;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00010001;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00001110;break;
  default : break;
  }
  break;

case 'P':      switch (row){
      case   1: fila_pos=0b00011110;break;
      case   2: fila_pos=0b00010001;break;
```

```
        case   3: fila_pos=0b00010001;break;
        case   4: fila_pos=0b00011110;break;
        case   5: fila_pos=0b00010000;break;
        case   6: fila_pos=0b00010000;break;
        case   7: fila_pos=0b00010000;break;
    default : break;
    }
    break;

case 'Q':      switch (row){
       case   1: fila_pos=0b00011110;break;
       case   2: fila_pos=0b00010010;break;
        case   3: fila_pos=0b00010010;break;
        case   4: fila_pos=0b00010010;break;
        case   5: fila_pos=0b00010110;break;
        case   6: fila_pos=0b00011110;break;
        case   7: fila_pos=0b00000001;break;
    default : break;
    }
    break;


case 'R':      switch (row){
       case   1: fila_pos=0b00011111;break;
       case   2: fila_pos=0b00010001;break;
        case   3: fila_pos=0b00010010;break;
        case   4: fila_pos=0b00011100;break;
        case   5: fila_pos=0b00010010;break;
        case   6: fila_pos=0b00010001;break;
        case   7: fila_pos=0b00010001;break;
    default : break;
    }
    break;

case 'S':      switch (row){
       case   1: fila_pos=0b00011111;break;
       case   2: fila_pos=0b00010000;break;
        case   3: fila_pos=0b00010000;break;
        case   4: fila_pos=0b00011111;break;
        case   5: fila_pos=0b00000001;break;
```

```
      case   6: fila_pos=0b00000001;break;
      case   7: fila_pos=0b00011111;break;
  default : break;
  }
  break;

case 'T':     switch (row){
     case   1: fila_pos=0b00011111;break;
     case   2: fila_pos=0b00000100;break;
      case   3: fila_pos=0b00000100;break;
      case   4: fila_pos=0b00000100;break;
      case   5: fila_pos=0b00000100;break;
      case   6: fila_pos=0b00000100;break;
      case   7: fila_pos=0b00000100;break;
  default : break;
  }
  break;


case 'U':     switch (row){
     case   1: fila_pos=0b00010001;break;
     case   2: fila_pos=0b00010001;break;
      case   3: fila_pos=0b00010001;break;
      case   4: fila_pos=0b00010001;break;
      case   5: fila_pos=0b00010001;break;
      case   6: fila_pos=0b00010001;break;
      case   7: fila_pos=0b00011111;break;
  default : break;
  }
  break;

case 'V':     switch (row){
     case   1: fila_pos=0b00010001;break;
     case   2: fila_pos=0b00010001;break;
      case   3: fila_pos=0b00010001;break;
      case   4: fila_pos=0b00010001;break;
      case   5: fila_pos=0b00010001;break;
      case   6: fila_pos=0b00001010;break;
      case   7: fila_pos=0b00000100;break;
  default : break;
```

```
  }
  break;


case 'W':      switch (row){
      case   1: fila_pos=0b00010001;break;
      case   2: fila_pos=0b00010001;break;
       case   3: fila_pos=0b00010001;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00010101;break;
       case   6: fila_pos=0b00010101;break;
       case   7: fila_pos=0b00011011;break;
  default : break;
  }
  break;

case 'X':      switch (row){
      case   1: fila_pos=0b00010001;break;
      case   2: fila_pos=0b00010001;break;
       case   3: fila_pos=0b00010001;break;
       case   4: fila_pos=0b00001110;break;
       case   5: fila_pos=0b00001010;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00010001;break;
  default : break;
  }
  break;

case 'Y':      switch (row){
      case   1: fila_pos=0b00010001;break;
      case   2: fila_pos=0b00010001;break;
       case   3: fila_pos=0b00011111;break;
       case   4: fila_pos=0b00000100;break;
       case   5: fila_pos=0b00000100;break;
       case   6: fila_pos=0b00000100;break;
       case   7: fila_pos=0b00000100;break;
  default : break;
  }
  break;
```

```
case 'Z':      switch (row){
      case   1: fila_pos=0b00011111;break;
      case   2: fila_pos=0b00000001;break;
       case   3: fila_pos=0b00000010;break;
       case   4: fila_pos=0b00000100;break;
       case   5: fila_pos=0b00001000;break;
       case   6: fila_pos=0b00010000;break;
       case   7: fila_pos=0b00011111;break;
  default : break;
  }
  break;
```

//MINUSCULAS

```
case 'a':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00011111;break;
       case   4: fila_pos=0b00000001;break;
       case   5: fila_pos=0b00001111;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00001111;break;
  default : break;
  }
  break;


case 'b':      switch (row){
      case   1: fila_pos=0b00010000;break;
      case   2: fila_pos=0b00010000;break;
       case   3: fila_pos=0b00010000;break;
       case   4: fila_pos=0b00011110;break;
       case   5: fila_pos=0b00010001;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00011110;break;
  default : break;
  }
  break;
```

```
case 'c':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00001111;break;
       case   4: fila_pos=0b00010000;break;
       case   5: fila_pos=0b00010000;break;
       case   6: fila_pos=0b00010000;break;
       case   7: fila_pos=0b00001111;break;
  default : break;
  }
  break;

case 'd':      switch (row){
      case   1: fila_pos=0b00000001;break;
      case   2: fila_pos=0b00000001;break;
       case   3: fila_pos=0b00000001;break;
       case   4: fila_pos=0b00001111;break;
       case   5: fila_pos=0b00010001;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00001111;break;
  default : break;
  }
  break;

case 'e':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00001110;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00011110;break;
       case   6: fila_pos=0b00010000;break;
       case   7: fila_pos=0b00001110;break;
  default : break;
  }
  break;

case 'f':      switch (row){
      case   1: fila_pos=0b00001111;break;
      case   2: fila_pos=0b00010000;break;
       case   3: fila_pos=0b00010000;break;
```

```
        case   4: fila_pos=0b00011110;break;
        case   5: fila_pos=0b00010000;break;
        case   6: fila_pos=0b00010000;break;
        case   7: fila_pos=0b00010000;break;
    default : break;
    }
    break;
  case 'g':      switch (row){
        case   1: fila_pos=0b00000000;break;
        case   2: fila_pos=0b00000000;break;
        case   3: fila_pos=0b00001110;break;
        case   4: fila_pos=0b00010001;break;
        case   5: fila_pos=0b00001111;break;
        case   6: fila_pos=0b00000001;break;
        case   7: fila_pos=0b00011110;break;
    default : break;
    }
    break;

  case 'h':      switch (row){
        case   1: fila_pos=0b00010000;break;
        case   2: fila_pos=0b00010000;break;
        case   3: fila_pos=0b00010000;break;
        case   4: fila_pos=0b00011111;break;
        case   5: fila_pos=0b00010001;break;
        case   6: fila_pos=0b00010001;break;
        case   7: fila_pos=0b00010001;break;
    default : break;
    }
    break;

  case 'i':      switch (row){
        case   1: fila_pos=0b00000100;break;
        case   2: fila_pos=0b00000000;break;
        case   3: fila_pos=0b00000000;break;
        case   4: fila_pos=0b00000100;break;
        case   5: fila_pos=0b00000100;break;
        case   6: fila_pos=0b00000100;break;
        case   7: fila_pos=0b00001110;break;
    default : break;
```

```
  }
  break;


case 'j':      switch (row){
      case   1: fila_pos=0b00000100;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00000100;break;
       case   4: fila_pos=0b00000100;break;
       case   5: fila_pos=0b00010100;break;
       case   6: fila_pos=0b00010100;break;
       case   7: fila_pos=0b00011100;break;
  default : break;
  }
  break;

case 'k':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00010010;break;
       case   4: fila_pos=0b00010100;break;
       case   5: fila_pos=0b00011000;break;
       case   6: fila_pos=0b00010100;break;
       case   7: fila_pos=0b00010001;break;
  default : break;
  }
  break;

case 'l':      switch (row){
      case   1: fila_pos=0b00000100;break;
      case   2: fila_pos=0b00000100;break;
       case   3: fila_pos=0b00000100;break;
       case   4: fila_pos=0b00000100;break;
       case   5: fila_pos=0b00000100;break;
       case   6: fila_pos=0b00000100;break;
       case   7: fila_pos=0b00001110;break;
  default : break;
  }
  break;
```

```
case 'm':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00011011;break;
       case   4: fila_pos=0b00010101;break;
       case   5: fila_pos=0b00010001;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00010001;break;
  default : break;
  }
  break;

case 'n':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00010000;break;
       case   4: fila_pos=0b00011111;break;
       case   5: fila_pos=0b00010001;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00010001;break;
  default : break;
  }
  break;

case 'o':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00001110;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00010001;break;
       case   6: fila_pos=0b00010001;break;
       case   7: fila_pos=0b00001110;break;
  default : break;
  }
  break;


case 'p':      switch (row){
      case   1: fila_pos=0b00000000;break;
```

```
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00001110;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00011110;break;
       case   6: fila_pos=0b00010000;break;
       case   7: fila_pos=0b00010000;break;
  default : break;
  }
  break;

case 'q':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00001111;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00001111;break;
       case   6: fila_pos=0b00000001;break;
       case   7: fila_pos=0b00000001;break;
  default : break;
  }
  break;

case 'r':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00011110;break;
       case   4: fila_pos=0b00010000;break;
       case   5: fila_pos=0b00010000;break;
       case   6: fila_pos=0b00010000;break;
       case   7: fila_pos=0b00010000;break;
  default : break;
  }
  break;


case 's':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00011111;break;
       case   4: fila_pos=0b00010000;break;
```

```
        case   5: fila_pos=0b00011111;break;
        case   6: fila_pos=0b00000001;break;
        case   7: fila_pos=0b00011111;break;
    default : break;
    }
    break;

case 't':      switch (row){
       case   1: fila_pos=0b00000100;break;
       case   2: fila_pos=0b00000100;break;
        case   3: fila_pos=0b00001111;break;
        case   4: fila_pos=0b00000100;break;
        case   5: fila_pos=0b00000100;break;
        case   6: fila_pos=0b00000100;break;
        case   7: fila_pos=0b00000111;break;
    default : break;
    }
    break;

case 'u':      switch (row){
       case   1: fila_pos=0b00000000;break;
       case   2: fila_pos=0b00000000;break;
        case   3: fila_pos=0b00010001;break;
        case   4: fila_pos=0b00010001;break;
        case   5: fila_pos=0b00010001;break;
        case   6: fila_pos=0b00010001;break;
        case   7: fila_pos=0b00001110;break;
    default : break;
    }
    break;


case 'v':      switch (row){
       case   1: fila_pos=0b00000000;break;
       case   2: fila_pos=0b00000000;break;
        case   3: fila_pos=0b00010001;break;
        case   4: fila_pos=0b00010001;break;
        case   5: fila_pos=0b00010001;break;
        case   6: fila_pos=0b00001010;break;
        case   7: fila_pos=0b00000100;break;
```

```
  default : break;
  }
  break;

case 'w':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00010001;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00010001;break;
       case   6: fila_pos=0b00010101;break;
       case   7: fila_pos=0b00011011;break;
  default : break;
  }
  break;

case 'x':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00010001;break;
       case   4: fila_pos=0b00001010;break;
       case   5: fila_pos=0b00000100;break;
       case   6: fila_pos=0b00001010;break;
       case   7: fila_pos=0b00010001;break;
  default : break;
  }
  break;


case 'y':      switch (row){
      case   1: fila_pos=0b00000000;break;
      case   2: fila_pos=0b00000000;break;
       case   3: fila_pos=0b00010001;break;
       case   4: fila_pos=0b00010001;break;
       case   5: fila_pos=0b00001111;break;
       case   6: fila_pos=0b00000001;break;
       case   7: fila_pos=0b00011110;break;
  default : break;
  }
  break;
```

```
        case 'z':      switch (row){
             case   1: fila_pos=0b00000000;break;
             case   2: fila_pos=0b00000000;break;
              case   3: fila_pos=0b00011111;break;
              case   4: fila_pos=0b00000010;break;
              case   5: fila_pos=0b00000100;break;
              case   6: fila_pos=0b00001000;break;
              case   7: fila_pos=0b00011111;break;
          default : break;
          }
          break;
                default:
                break;
     }
  return fila_pos;

}
//FIN DECODIFICACION

/*---------------------------------------------------------------------
// void main(void)
//---------------------------------------------------------------------*/
void main(void)
{
        unsigned char fila;
        unsigned char pos;
        unsigned char leds_on[4];
        unsigned char aux;
        unsigned char letra[4];
        unsigned char aux1;
        unsigned char aux2;
        Setup();
        aux1=0x02;

//INICIALIZACIONES
        while(1)
        {
                if(!stat.wdtdis)
                CLRWDT();
        letra[0]=DATO1;
```

```
        letra[1]=DATO2;
        letra[2]=DATO3;
        letra[3]=DATO4;

    for(fila = 1;fila<=7;fila++)     // fila indica la fila actual
    {
        PORTB=~aux1;
        aux1=aux1<<1;

                        for(pos= 0;pos<=4;pos++) //4 pos indica la posicion de letra
        {
                aux=letra[pos];
                leds_on[pos]=leds5(aux,fila);
        }

        aux2=leds_on[0];
                PORTD=((aux2|0b11100000)&(PORTD|0b00011111));//MATRIZ 1
                aux2=leds_on[1];
                PORTA=aux2;//MATRIZ 2
                aux2=leds_on[2];
                PORTE=((aux2>>2|0b11111000)&(PORTE|0b00000111));//MATRIZ 3
                aux2=leds_on[2];
                PORTD=((aux2<<5|0b10011111)&(PORTD|0b01100000));
                aux2=leds_on[3];
                PORTD=(aux2<<3|0b01111111)&(PORTD|0b10000000);//MATRIZ 4
                aux2=leds_on[3];
                PORTC=(aux2>>1|0b11111000)&(PORTC|0b00000111);
                aux2=leds_on[3];
                PORTC=(aux2<<5|0b11011111)&(PORTC|0b00100000);
                DelayMs(2);
                PORTD=0x00;
                PORTC=0x00;
                PORTA=0x00;
                PORTE=0x00;
        }
     aux1=0x02;
 } //FIN WHILE
}//FIN MAIN

/*-------------------------------------------------------------------
```

```
 void Setup(void)
 Inicializa las variables de programa y registros periféricos.
---------------------------------------------------------------------------*/

void Setup(void)
{
stat.msec10 = 0;
stat.msec100 = 0;
stat.msec1000 = 0;
stat.wdtdis = 0;
RXBufferIndex = 0;
SensBufIndex = 0;
CmdBufIndex = 0;
TXChecksum.i = 0;
RXChecksum = 0;
Count_10m = 0;
Count_100m = 0;
Count_Tach0 = 0;
Count_Tach1 = 0;
Count_Tach2 = 0;
Count_Tach3 = 0;
CmdBuf[0] = 0;
//PUERTOS MATRICES
        ADCON1= 0x07;
        TRISA = 0b00000000;
        TRISB = 0b00000000;
        TRISD = 0b00000000;
        TRISC = 0x98;
        TRISE = 0b00000000;
        PORTA=0x00;
        PORTB=0x00;
        PORTD=0x00;
        PORTE=0x00;
        DATO1='E';
        DATO2='S';
        DATO3='P';
        DATO4='E';

OPTION = 0x78;
SSPADD = NODE_ADDR;
```

```
SSPSTAT = 0;
SSPCON = 0;
SSPCON = 0x36;
CCPR2L = CCP_LBYTE;
CCPR2H = CCP_HBYTE;
CCP2CON = 0x0a;
TMR1L = 0;
TMR1H = 0;
T1CON = 0x01;
CLRWDT();
CCP2IF = 0;
ADIF = 0;
SSPIF = 0;
SSPIE = 1;
PEIE = 1;
GIE = 1;
}
```

4.8 Monitoreo de señales - Aplicación AN736

Realizar una análisis de la aplicación AN736 y efectuar las modificaciones necesarias para que pueda entrar en funcionamiento, de tal forma que se visualice en el LCD del sistema Maestro SDP877 los valores de las señales analógicas.

4.8.1 Información Requerida

Para esta práctica es necesario otro microcontrolador PIC 16F877 para comunicarlo con el maestro y obtener la información necesaria mediante el uso del bus I2C.

El primer paso y el más importante es conocer la dirección del esclavo, en este caso la dirección del PIC 16F877 esclavo es la 0x18H. Esta dirección puede ser modificada en el programa del esclavo, en la variable NODE ADDRESS, es allí donde se determinara la dirección que el usuario decida. Claro esta, se deberá trabajar en el programa maestro haciendo referencia a esa misma

dirección. En caso de no hacerlo de forma correcta, el maestro no reconocerá al esclavo y la comunicación fallara.

Esta comunicación será establecida entre dos PIC's, usando el programa base de la Microchip. Donde se informa que se pueden implementar entradas analógicas y digitales, además sensores de temperatura y de velocidad. Para la práctica actual se hará referencia a las señales analógicas, dejando como prácticas propuestas las restantes.

Las señales analógicas serán representadas con bancos de potenciómetros y los valores se presentaran en el LCD del sistema maestro.

4.8.1.1 Definición del protocolo de red.

Ahora que el funcionamiento básico del bus I2C ha sido explicado en los capítulos anteriores, se procederá a examinar las necesidades de la red. En este sistema, un elemento maestro esta en el bus e iniciará comunicaciones periódicas con los elementos esclavos.

El protocolo debe permitir al elemento maestro leer o escribir datos desde un elemento esclavo particular. El tipo y longitud del dato leído desde, o escrito hacia el esclavo dependerá, por supuesto, de las funciones específicas del esclavo.

Por esta razón, sería eficiente que el protocolo de red soporte un dato de longitud variable que depende del sensor. El protocolo puede además permitir que la dirección del dato sea especificada. Usando la dirección del dato y la longitud del mismo, el nodo maestro puede pedir uno o todos los datos de información disponibles del nodo esclavo.

Debe existir un método en el protocolo de red para asegurar que el dato fue transmitido o recibido satisfactoriamente, conocido como checksum. Usando

un control total (checksum), los elementos maestros y esclavos en el sistema verifican que el dato recibido fue válido. Si el dato no es válido, el dato debe ser retransmitido.

Las causas de error de transmisión incluyen: múltiples elementos respondiendo a la misma dirección (colisiones del bus) o condiciones de no repuesta desde los elementos del bus.

4.8.1.2 Formato de mensajes del elemento maestro

Ya que toda la comunicación en el bus I2C es iniciada por el elemento maestro, es necesaria una descripción del protocolo implementado por este elemento. En esta aplicación el elemento maestro puede iniciar uno de dos tipos de mensajes; un mensaje de escritura de datos o un mensaje de petición de datos.

4.8.1.3 Formato del mensaje de escritura de datos

El formato de un mensaje de escritura de datos es mostrado en la Ilustración 84. El mensaje de escritura de datos empieza con la condición de START. Cuando la condición de START se completa, el elemento maestro envía la dirección I2C del nodo esclavo con el bit R/W encerado para indicar que el dato será escrito al elemento esclavo. El siguiente byte provee la información de longitud del byte.

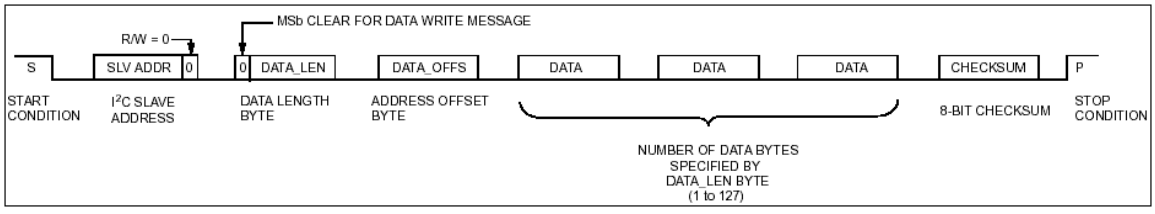

Ilustración 84 Formato de Escritura de Datos por el bus I2C

Para esta discusión, este byte será referido como el byte DATA_LEN. El byte DATA_LEN tiene dos propósitos. Primero, los siete bits menos significativos más bajos indican el numero de bytes de datos a ser escritos al elemento esclavo. Segundo, los bits más significativos (MSB's) indican si el dato va a ser escrito o leído del esclavo. En este caso el bit más significativo es encerado para indicar que una escritura de datos va a ser realizada.

El MSB del byte DATA_LEN efectúa una operación similar para el protocolo de red, así como el bit R/W en el byte de dirección I2C, pero aun así, ninguno de los dos debe ser confundido.

El siguiente byte enviado por el maestro indica la dirección de inicio en el buffer de datos del nodo esclavo, en donde va a ser escrita o leída la información. Este byte será referido como el byte DATA_OFFS. Cada elemento esclavo en la red mantiene un rango de memoria de datos para los datos recibidos y los datos a ser transmitidos.

En un mensaje de escritura de datos, el numero de bytes de datos especificados por el byte DATA_LEN estará seguido del byte de DATA_OFFS. Cuando el último byte de datos ha sido enviado, el maestro envía 8 bits, complementando el checksum de todos los datos previamente enviados, incluyendo el byte de dirección I2C del nodo esclavo. Finalmente, el elemento maestro termina el mensaje de escritura de datos generando una condición de STOP.

4.8.1.4 Formato de Mensaje de Petición de Datos

El formato para un mensaje de petición de datos es mostrado en la Ilustración 85. Siguiendo la condición de START, el elemento maestro envía la dirección del nodo esclavo con el bit R/W encerado a fin de indicar una escritura de datos al elemento I2C esclavo. Luego, el byte DATA_LEN es enviado. Los siete bits menos significativos de este byte indica el numero de bytes de datos a

ser leídos del esclavo. Se procede a establecer el valor del MSB porque una lectura de datos del esclavo debe ser realizada. El byte DATA_OFFS sigue al byte de longitud del dato e indica la dirección de inicio en la memoria de datos del nodo esclavo desde el cual el dato va a ser leído.

Es necesario realizar una escritura de datos primero, ya que el puntero que leerá los datos debe ser ubicado en la dirección adecuada y eso se lo hace mediante un proceso de escritura, luego de ello se puede proceder a la lectura de la información.

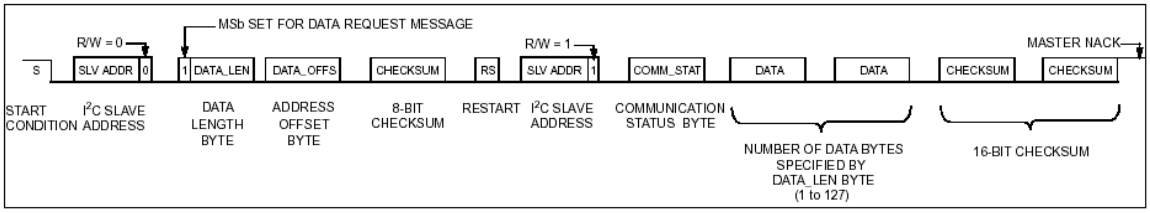

Ilustración 85 Formato de Petición de datos en la comunicación I2C

4.8.1.5 Procesamiento del mensaje del nodo esclavo

En general, el elemento maestro puede leer datos del esclavo luego de un mensaje de escritura de datos o un mensaje de requerimiento de datos, inicializando una condición de encendido en el bus I2C y enviando la dirección del esclavo con el bit R/W establecido. El tipo de mensaje que fue previamente enviado por el maestro y su validez, determina el dato que será retornado por el esclavo.

Cada nodo esclavo mantiene varios bits de status para indicar la validez del mensaje enviado por el elemento maestro. Estos bits de status son almacenados en el byte de status de comunicación (COMM_STAT) y la Tabla 8 indica el significado de cada bit.

Bit No.	Nombre del Bit	Descripción
0	comm_stat.chkfail	Indica una falla en el control de la transmisión ocurrida en el último

		mensaje enviado
1	comm_stat.rxerror	Indica que el nodo esclavo no interpreta correctamente el mensaje del maestro.
2	comm_stat.ovflw	Indica que el elemento maestro ha hecho una petición de lectura/escritura de uno o varios bytes de datos del nodo esclavo, fuera del rango válido de direcciones para un esclavo en particular.
3	comm_stat.sspov	Indica un desborde ocurrido en el módulo SSP para una dirección de esclavo dado, porque el elemento esclavo no estuvo en capacidad de procesar los datos I2C ingresados lo suficientemente rápido.
4		No es usado
5		No es usado
6		No es usado
7	comm_stat.r_w	Indica tanto el último mensaje del maestro fue un mensaje de petición de datos (R/W=1) o un mensaje de escritura de datos (R/W=0)
Nota: la estructura del bit 'comm_stat' es usada en el código fuente C, para acceder los bits del byte COMM_STAT		

Tabla 8 Definiciones de los bits de COMM_STAT

El byte COMM_STAT es siempre el primer byte de datos a ser retornado en cualquier transferencia de datos desde el nodo esclavo al nodo maestro. Esto permite al maestro verificar que el mensaje previamente enviado fue procesado correctamente por el nodo esclavo. Si, por ejemplo, el maestro envía un mensaje de escritura de datos, el valor de byte COMM_STAT será 00h, si el dato fue recibido correctamente por el esclavo. Si el mensaje de requerimiento de datos fue previamente enviado por el maestro, el valor del byte COMM_STAT sería 80h. Si el maestro recibe otro valor en el byte COMM_STAT, algún tipo de error ha ocurrido y el maestro puede enviar el mensaje de escritura de datos o de petición de datos nuevamente.

Si el mensaje de escritura de datos fue previamente enviado al nodo esclavo, el maestro no necesita recibir mas bytes del nodo esclavo, luego de que el byte COMM_STAT es leído. Para un mensaje de petición de datos, el maestro puede leer el numero de bytes de datos especificado en DATA_LEN.

Un complemento de 16 bits para checksum es calculado para el dato retornado al maestro. El valor de checksum incluye el byte COMM_STAT,

además todos los bytes de datos que fueron retornados. El elemento maestro puede recibir dos bytes de checksum para errores que ocurren mientras se reciben los bytes de datos, éste puede tratar de leer el dato del esclavo nuevamente.

Ahora que la implementación básica del protocolo de red ha sido definida, se procederá a presentar la operación funcional de los nodos maestro y esclavo.

4.8.2 Nodo Maestro

4.8.2.1 Descripción general.

Para la presente aplicación, un PICmicro 16F877 ha sido implementado como el controlador maestro en un bus I2C. Este dispositivo provee módulos MSSP y USART para las comunicaciones I2C y USART respectivamente.

El código de esta aplicación esta escrito en C, usando el compilador Hi-Tech PIC . Para ello la Tabla 9 provee una breve descripción de los archivos de sistema.

Nombre del archivo	Descripción
mstri2c.c	Lazo de código principal y funciones de control de interrupciones
mstri2c.h	Declaración de variables y definiciones
I2c_comm.c	Rutinas para comunicación con los elementos esclavos I2C
I2c_comm.h	Declaración de variables y definiciones
init .c	Rutinas para inicializar los periféricos y puertos del PIC
pic.h	Requerido por el compilador para declaraciones SFR (archivo Hi-Tech)
delay.h	Función prototipo de retardo (archivo Hi-Tech)

Tabla 9 Archivos de código fuente 'C' del Maestro I2C

En esta aplicación, el maestro ejecuta tres tareas básicas:

1. Lecturas del esclavo
2. Escritura del esclavo
3. Transmisión de los datos recibidos del esclavo I2C y estado del bus al PC.

Para la mayor parte, estas funciones ocurren en base a una interrupción.

Existen cuatro tipos de interrupciones que pueden ser implementadas.

1. Interrupción de fin de evento I2C._ Esta interrupción de evento I2C indica que el evento I2C ha sido completado.

   Los eventos I2C incluyen señales de START (inicio), STOP (parada), Restart (reinicio), Acknowledge (conocimiento), READ (Lectura) y WRITE (Escritura). El bit SSPIF (PIR1<6>) de hardware periférico es aquel al cual se hace referencia para este tipo de interrupción.

2. Interrupción de colisión de bus._ Esta interrupción es utilizada para manejar la detección de colisión de bus. Típicamente, en un sistema de maestro único, una colisión de bus no es probable.

3. Interrupción de desbordamiento del Timer1._ Esta interrupción es utilizada para generar un pulso de tiempo de 100 ms. para iniciar la comunicación I2C. Cuando el maestro completa un ciclo de comunicación I2C, el Timer1 es reiniciado. Cuando el Timer1 se desborda (100 ms. mas tarde), el siguiente ciclo de comunicaciones I2C inicia.

4. Interrupción de Transmisión USART._ Esta interrupción es usada para enviar 10 bytes de datos al PC. Luego de que el maestro se comunica con cada esclavo, un paquete de datos es formado. El paquete consiste de los datos a ser leídos del esclavo y el estado del bus I2C. Cada byte es transmitido al PC a una velocidad de 9600 baudios.

4.8.2.2 Implementación del Maestro

El elemento maestro, ejecuta una inicialización de variables y de periféricos básicos. Las funciones empleadas para la inicialización de los periféricos están nombradas a continuación.

• Init_Usart()
• Init_Ports()
• Init_Timer1()
• Init_Ssp()

Estas funciones están localizadas en el archivo init.c.

Con la función INIT_Ssp(), el módulo MSSP es inicializado para el modo I2C maestro, 400 KHz para tasa de transmisión. Una vez que se ha inicializado los periféricos, las interrupciones globales y periféricas son habilitadas y se ingresa al lazo principal de ejecución.

En el lazo principal se prueba el estado de dos banderas:

• sflag.event.read_i2c
• sflag.event.i2c_event

Estas banderas son inicialmente colocadas en alto en la Rutina de Interrupción del Timer1 (ISR). La interrupción del Timer1 inicia el proceso de comunicación I2C y se repite cada 100 ms. En el ISR del Timer1, el timer es apagado, la respectiva interrupción se deshabilita y hace referencia a las banderas de eventos que se colocaron.

Cuando se ejecuta el lazo principal del programa, el F/W prueba el estado de esas dos banderas. Si las dos son un ‘1’ lógico, la función Service_I2Cslave() es llamada. Si una o dos de las banderas son negadas (‘0’ lógico), un lazo que incluye una instrucción de CLRWDT y el proceso que prueba la bandera son repetidos.

Cuando la función Service_I2Cslave() es llamada, algunos estados de código son probados y ejecutados, en caso de cumplirse:

- Se prueba si una nueva ronda de comunicaciones esclavo se inician. Si esto ocurre, se inicializan las variables y las banderas. Esta prueba es verdadera cada que ocurre un evento con el Timer1.

- Prueba si el estado del bus I2C fue un estado de escritura I2C. Si esto se cumple, se hará una prueba de conocimiento de error. Si este error existe entonces el problema es la condición de STOP.

- Probar si existe un bus I2C o un conocimiento de error. Si es verdad, establecer el estado de error para la transmisión al PC. Si es falso, encerar el estado de error.

- Probar si el maestro I2C puede comunicarse con el siguiente elemento esclavo. Si es verdad elaborar lo siguiente:

    - Inicializar las variables y banderas.

    - Llamar a la función Compose_Buffer(). En esta función una prueba es realizada para determinar si el paquete de datos leídos desde el esclavo es válido. Si es válido, se inicia la transmisión del paquete de datos al PC. Si no es válido, ejecuta una nueva transmisión con el mismo esclavo.

    - Probar si solo un valor recibido del esclavo esta fuera de rango. El rango limite de valores para pruebas es establecido por #define limit 0x80.

    - Probar si el maestro se ha comunicado con todos los elementos esclavos. Si es verdadero, retornar al lazo de código principal y esperar los siguientes 100 ms para que expire. Si es falso, iniciar el

siguiente estado del bus I2C, que puede ser START, STOP, Restart, Read, Write, envío de ACK o NACK.

Como se menciona, cada nuevo ciclo de comunicaciones I2C empieza 100ms antes de completar el ciclo anterior. Este ciclo es un poco arbitrario, desde que el dato del esclavo no es usado mas que para presentación en el PC y una recolección de datos con tasa de 100 ms es adecuado para esta aplicación.

El ciclo de comunicación I2C toma aproximadamente 5ms con cada esclavo. Siguiendo esto, 10 bytes son transmitidos al PC a 9600 baudios, que equivalen a aproximadamente 5.3ms. El dato es transmitido al PC en base a una interrupción con la función interrup_piv().

En esta aplicación, el elemento maestro I2C se comunica con algunos esclavos, siendo posible incrementar el numero de esclavos, sin embargo se deberá tomar en cuenta la limitación de recursos que tenga el PIC a emplear.

Por ejemplo un elemento esclavo luego de una petición puede transmitir 127 bytes de datos al maestro. El dato leído del esclavo debe calzar en la memoria en base a un arreglo de variables usado para mantener el dato. Esto puede o no ser posible, basado en los requerimientos de recursos del elemento maestro I2C. Además se define un arreglo de variables RAM y se inicializa con un byte de longitud de dato y 8 bits para control por cada esclavo.

Se conoce que el tamaño del arreglo depende del numero de esclavos. Sin embargo como este arreglo está ubicado en la RAM, puede ser ubicado en la memoria de programa y entonces una actualización dinámica al arreglo no sería posible. En conclusión esta aplicación puede ser modificada para aumentar el numero de esclavos I2C con pocos cambios al código.

Durante cada ciclo de comunicación del esclavo, el maestro lee el dato y estado del esclavo mientras monitorea y graba errores, tal como una colisión de bus y errores de Acknowledge (NACK). Mientras las colisiones de bus son más comunes en un ambiente multi-maestro, una colisión de bus puede aun ocurrir en

un sistema con un solo maestro. Por ejemplo, un elemento esclavo puede experimentar un mal funcionamiento y como resultado los niveles del bus SDA y SCL se manejan en bajo durante la transmisión. La última condición de error puede resultar en una falla permanente del bus hasta que una acción correctiva sea tomada. En todo caso, el elemento maestro I2C puede monitorear para esta condición y tomar una acción adecuada. Cuando una colisión de bus es detectada, el bit de estatus debe ser colocado en '1' lógico para el esclavo en particular. Cuando la colisión de bus es corregida, el mismo bit de status será colocado en cero lógico. Esta información de estado es parte del paquete de datos enviado al PC.

Además, el maestro puede atender al menos una petición repetida de comunicación I2C. Adicionalmente los reintentos son atendidos cambiando el texto de sustitución en la macro definida en el archivo i2c_comm.h. Por ejemplo una petición de reenvío es implementada por: #define MaxSlaveryRetry 1, dos reintentos de comunicación son hechas por: #define MaxSlaveryRetry 2.

Otra condición de error que el elemento maestro I2C puede detectar es la condición de Desconocimiento (NACK). Si por alguna razón, el esclavo enfrenta un problema de NACK (no envía la línea SDA a bajo durante el noveno pulso de reloj en escritura), el maestro puede detectarlo y tomar la acción adecuada. Así como el error de colisión de bus, un bit de status será analizado de acuerdo al estado del error. Para esta condición, el maestro enfrenta el problema de una condición de STOP luego de detectar el NACK. Esta acción difiere de la colisión de bus, en ese caso como resultado de una colisión de bus el módulo MSSP pasa a un estado de reposo (IDDLE). El siguiente estado I2C válido sería una condición de START. En un bus I2C la combinación STOP/START se ejecuta dependiendo de la acción deseada.

Para esta aplicación (Ilustración 85), el maestro lee cinco bytes de información de cada esclavo, tres bytes de datos y dos bytes de control. El dato junto con la identificación del esclavo, el estado del bus y de error de comunicación son transmitidos al PC para presentación. Mientras la transmisión USART esta en progreso, el maestro también puede ejecutar una secuencia de

escritura al esclavo. La secuencia de escritura es automática para cada esclavo, pero el dato a escrito depende del valor del segundo byte leído desde el esclavo.

La función Write_I2Cslave() efectúa una secuencia de escritura y es llamada desde la función Compose_Buffer(). Esta secuencia de escritura es concurrente con la comunicación USART. Para esta aplicación la función Write_I2Cslave() provee al elemento esclavo I2C una respuesta del maestro. Esta función ejecuta un lazo de control usando la interrupción de realización de evento I2C.

Finalmente, para cada estado de comunicaron I2C con el esclavo, excluyendo la función Write_I2Cslave(), el maestro genera el estado del bus I2C con la función I2CBusState() . Esta función esta basada en las líneas de control switch/case. Desde que se ingresa esta función, el F/W efectúa una chequeo a la tabla para conocer el siguiente estado I2C. Los estados para cada secuencia están predefinidos en el arreglo const unsigned char ReadFSlaveI2Cstates, declarados en el archivo i2c_comm.h. esta implementación permite añadir o eliminar estados del bus I2C. Cuando el siguiente estado del bus I2C ha sido obtenido, una sentencia de switch evalúa el estado de las variables i2cstate y el caso correcto inicia el siguiente estado del bus. Entonces el F/W retorna al lazo de código principal y espera la siguiente interrupción de evento I2C.

4.8.3 Nodo Esclavo

El código del nodo esclavo esta escrito para el PIC 16F877 usando el compilador Hi-Tech. Se eligió este PIC porque posee el módulo SSP que es necesario para la comunicación I2C.

El código esclavo contiene las siguientes funciones:

- Setup()
- ISR_Handler()
- SSP_Handler()
- AD_Handler()

• CCP2_Handler()

La función Setup() inicializa todos los registros especiales de las funciones (SFR) y todas las variables de programa.

4.8.3.1 Interrupciones

El código del nodo esclavo es principalmente manejado por interrupciones.

Los módulos SSP, CCP2 y A/D son fuentes de interrupción. La función ISR_Handler() lleva los bits de las banderas de interrupción y llama a la función del módulo adecuado.

4.8.3.2 Evento de Timing

El módulo CCP2 es usado en el modo comparador como un evento de tiempo y provee una interrupción cada 1 mseg. La función CCP2_Handler() es llamada cuando una interrupción CCP2 ocurre. A mas de la interrupción de 1 mseg, la función CCP2_Handler() también mantiene banderas de tiempo de 10 mseg, 100 mseg y 1000 mseg para otros eventos programados.

4.8.3.3 Buffers de Datos del nodo Esclavo

Tres buffers de datos son empleados en la aplicación del nodo esclavo. El primero de estos buffers de datos es SensorBuf, que tiene una longitud de 12 bytes y mantiene los datos del sensor a ser enviados al nodo maestro. El primer byte de SensorBuf mantiene el byte de estado de comunicación (COMM_STAT), el cual tiene bits de estado que indican el éxito o falla de una operación realizada por el maestro. Los siguientes dos bytes en SensorBuf contienen los bits de estado reservados para indicar las condiciones de fuera de rango para cada

elemento en el nodo esclavo. Estos bits pueden ser leídos por el elemento maestro para obtener una rápida respuesta de 'go/no-go' para todos los parámetros que el nodo esclavo esta registrando. Los nueve bytes restantes en SensorBuf contienen 8 bits de valores de datos para cada nodo esclavo. Las constantes son definidas al inicio del código fuente

El siguiente buffer el RXBuffer, el cual contiene los bytes enviados por el maestro durante los procesos de requerimiento de datos y escritura de datos. La longitud de este buffer se ha definido que debe ser de 8 bytes.

Este buffer debe ser lo suficientemente largo para almacenar el byte de dirección de esclavo (SLAVE_ADDR), el byte de longitud de dato (DATA_LEN), el byte de offset del dato (DATA_OFF) y el numero total de bytes de datos que el maestro puede escribir en el esclavo.

El tercer buffer es CmdBuf, el cual tiene los bytes de datos escritos al elemento esclavo. Los bytes de datos son copiados desde RXBuffer a CmdBuf, cuando un mensaje de escritura de datos ha sido recibido desde el maestro. Si el mensaje de escritura de datos es inválido, los bytes en RXBuffer son descartados.

4.8.3.4 Evento SSP

Los eventos del bus I2C son procesados en la función SSP_Handler(), la cual es el corazón del protocolo de red I2C.

El módulo SSP esta configurado para el modo esclavo I2C, direccionamiento de 7 bits. Cuando una interrupción SSP ocurre, la función SSP_Handler debe identificar el evento I2C que acaba de ocurrir en el bus y tomar la decisión adecuada. Para propósitos de explicación, es útil identificar los posibles estados del módulo SSP luego de un evento I2C y discutir cada uno de ellos.

Los siguientes cinco estados son reconocidos por la función SSP_Handler() mediante los bits en el registro SSPSTAT:

Estado 1: Operación de escritura I2C, el último byte recibido fue una dirección, buffer esta lleno.

Estado 2: operación de escritura, el último byte recibido fue un dato, el buffer esta lleno.
Estado 3: Operación de lectura I2C, el último byte recibido fue una dirección, el buffer esta vacío.

Estado 4: Operación de lectura I2C, el último byte recibido fue un dato, el buffer esta vacío.
Estado 5: Reseteo lógico de I2C por NACK enviado desde el elemento maestro.

Estado 1.

El estado 1 ocurre luego de que una condición de START válida ocurre en el bus y una dirección fue transmitida, causando una compatibilidad de dirección en el módulo SSP del elemento esclavo. El LSB del byte de direcciones es ‘0’, lo cual indica una operación de escritura. Esta condición indica que el elemento maestro esta apunto de enviar los bytes para una nueva escritura de datos o petición de datos.

Este es el inicio de una nueva transacción en el bus, una bandera de estatus es establecida en el software para evitar una deshabilitación del WatchDog Timer en el programa principal. Si la transacción toma mas tiempo de lo esperado, se generaran dos errores con el elemento esclavo o un error en el bus, luego el WatchDog Timer resetea el elemento esclavo y el módulo SSP. En adición, el byte COMM_STAT es inicializado mediante el seteo del bit comm_stat.rxerror. Este bit no será reseteado hasta que todos los bytes del mensaje de escritura de datos y petición de datos hayan sido recibidos y una transmisión válida haya sido verificada. La variable RXBufferIndex es seteada en ‘0’ y RXBuffer es reseteado.

El byte de dirección que esta actualmente en SSPBUF esta almacenado en RXBuffer y es usado también para inicializar el valor de RXChecksum.

Estado 2.

En el segundo estado reconocido por la función SSP_Handler(), los bytes para un mensaje de escritura de datos y petición de datos son almacenados en RXBuffer y RXBufferIndex es incrementado luego de cada byte recibido, para apuntar a la siguiente ubicación vacía del buffer. El valor de RXBufferIndex es revisado comparando con la longitud de RXBuffer para asegurarse que no existe desborde del buffer. Si un desborde del RXBuffer ocurre, el valor de RXBufferIndex es colocado en la última ubicación del buffer y el bit comm_stat.ovflw es ubicado en el byte COMM_STAT para indicar que el desborde ocurrió. Si un módulo SSP se desborda, el bit comm_stat.sspov es colocado en COMM_STAT. Luego de que cada byte sea es recibido, su valor se añade a RXChecksum y el RXBufferIndex es comparado con el índice constante para determinar la importancia del presente byte en SSPBUF.

Si el byte recién recibido es el byte de DATA_LEN (byte #1), el MSB se revisa para ver si una escritura de dato o petición de dato va a ser ejecutada y el bit comm_stat.r_w en el byte COMM_STAT es determinado para indicar el estado del mensaje. Si el MSB del byte DATA_LEN está seteado, indicando un mensaje de petición de datos, este bit es enmascarado a ‘0’, de tal forma que no afecte futuros cálculos usando la longitud del dato almacenado en los 7 LSB. El valor de DATA_LEN es usado para determinar el valor de RXByteCount, el cual tiene el numero esperado de bytes a ser recibidos por el mensaje. Para un mensaje de petición de datos, RXByteCount siempre es establecido en ‘3’, porque el numero de bytes esperados esta arreglado. Para un mensaje de escritura de datos, RXByteCount se coloca en ‘3’, más el numero de bytes indicados por el byte DATA_LEN.

Si el byte recién recibido es el byte de DATA_OFFS (byte #2), una revisión es realizada para ver si el mensaje de petición de datos o el mensaje de

escritura de datos excede el tamaño de SensorBuf o CmdBuf. Si el mensaje excede el tamaño del buffer, el bit de estado de comm_stat.ovflw es seteado.

Si el numero de bytes recibidos es igual a RXByteCount, el fin del mensaje ha sido alcanzado. Si el valor de RXChecksum no es '0', el bit de estado comm_stat.chkfail en el byte COMM_STAT es seteado. Si un mensaje de escritura de dato fue enviado y RXChecksum es '0', luego el dato contenido en el mensaje es considerado válido y es transferido desde RXBuffer a CmdBuf.

Estado 3.

El estado 3 ocurre luego de que una condición de START válida ocurra en el bus y una dirección haya sido transmitida, causando una compatibilidad de dirección en el módulo SSP del elemento esclavo. El LSB del byte de dirección es '1', lo que indica una operación de lectura I2C. Esta condición indica que el elemento maestro desea leer bytes desde el electo esclavo.

Como se mencionó, el byte COMM_STAT siempre será el primer byte retornado durante una lectura del esclavo. Este byte es escrito a SSPBUF y el valor de TXChecksum es inicializado. El valor de SensBufIndex es colocado en '0' para operaciones de lectura en el futuro.

Estado 4.

En el estado 4, el nodo esclavo enviará bytes de datos en SensorBuf al maestro, basado en los valores de los bytes DATA_LEN y DATAOFFS. Cada byte que es enviado es añadido al valor de TXChecksum. Si el numero de bytes especificado en el byte DATA_LEN ha sido enviado, entonces el valor de 16 bits de TXChecksum es retornado. Si no existen mas bytes a ser retornados al maestro, entonces el esclavo simplemente retorna una dato inactivo.

Estado 5

El último estado detectado en la función SSP_Handler() es causado por una señal NACK desde el electo esclavo. Esta acción indica al esclavo que el maestro no desea recibir mas datos. El evento NACK es usado como una señal en este protocolo para indicar la finalización de una transacción en el bus I2C. Consecuentemente la bandera stat.wdtdis es encerada para rehabilitar el encerramiento del WatchDog Timer.

4.8.4 Diagrama General de Funcionamiento

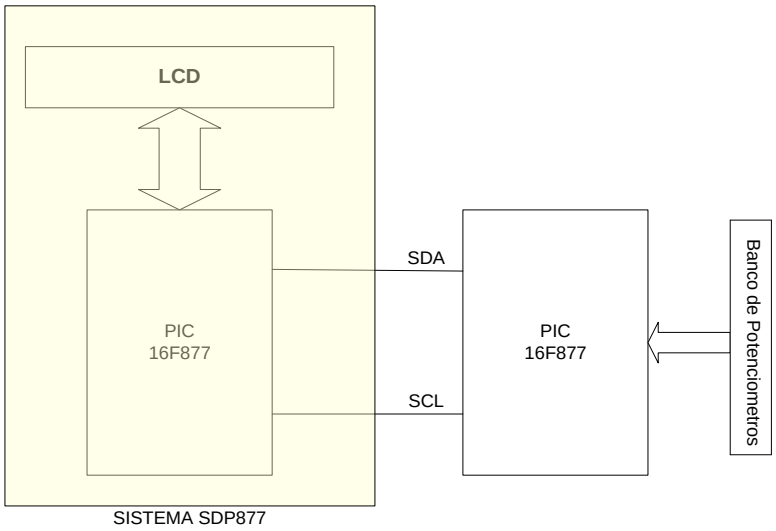

Ilustración 86 Diagrama de funcionamiento

Las entradas analógicas que se emplearan en la práctica serán las pertenecientes al PUERTO A, desde RA0 hasta RA4. Los valores que se modifiquen en los potenciómetros se visualizarán en el LCD.

Se emplearan potenciómetros de 50 KΩ en cada entrada analógica.

4.8.5 Diagramas de Flujo

Ver ANEXO 2.

4.8.6 Funcionamiento del Programa

Para la práctica se debe tener extremo cuidado en la configuración de los puertos, así como las conexiones de los mismos. Una falla de este tipo producirá un daño permanente en el puerto del PIC. El registro que configura el PUERTOA como entradas analógicas es el ADCON1, al setear un valor de 0x02.

Se añadió funciones de manejo del LCD, con el fin de presentar la información en el mencionado dispositivo y permitir una visualización clara para el usuario.

Los eventos I2C se producirán cada 100 ms, este valor se determina mediante software. En la Ilustración 87 se observa en conjunto los eventos.

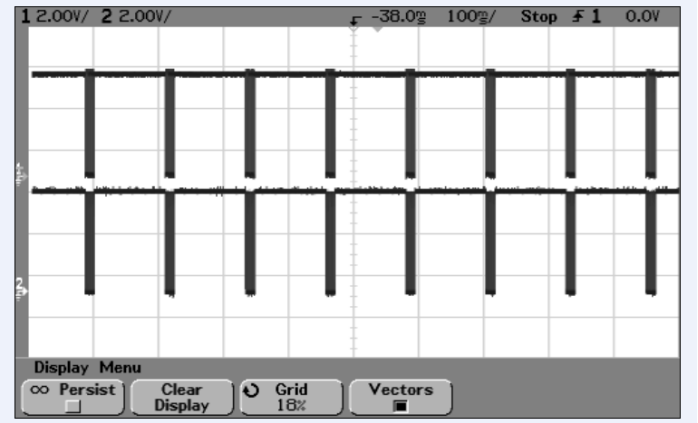

Ilustración 87 Conjunto de eventos I2C

A continuación se analizara un solo evento para verificar la situación de la comunicación, así como los datos que están ocupando el bus.

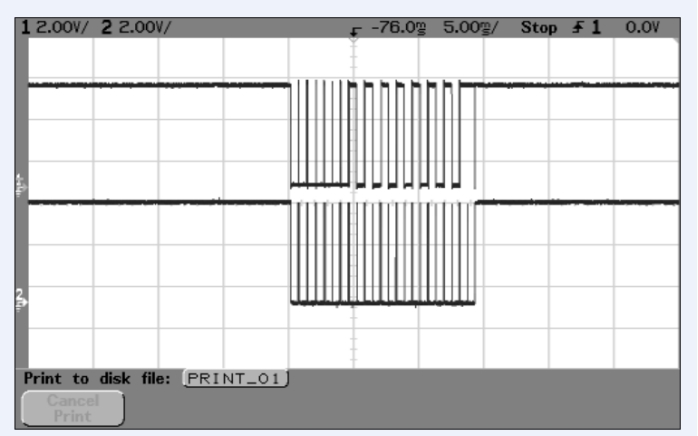

Ilustración 88 Evento I2C de comunicación PIC-PIC

Luego de la ya conocida señal de START que debe generarse para todo proceso de comunicación I2C, se procede a enviar la dirección del esclavo, junto a la operación a realizarse (escritura 0), además de la señal de reconocimiento, el noveno bit, en la línea de reloj.

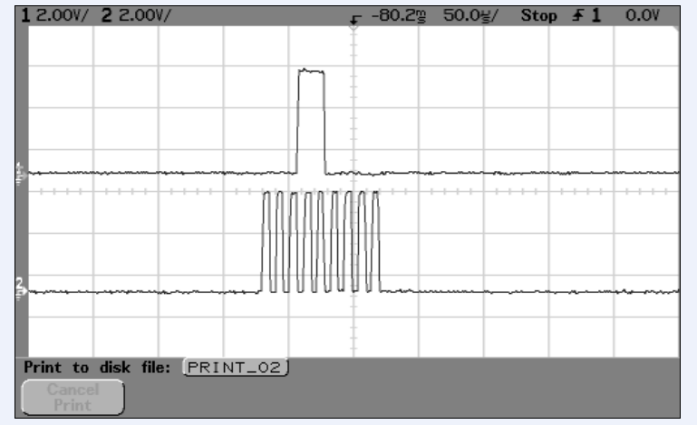

Ilustración 89 Dirección del esclavo. (0x18H)

Ahora se procede a observar las señales que los potenciómetros mediante el PIC ingresan al bus I2C, y este las transmite al PIC maestro para presentarlas en el LCD.

En las ilustraciones siguientes se observaran tres señales de las 5 que existen en la práctica.

La primera señal corresponde al potenciómetro 1, registrando un valor de 0b01001101, correspondiente a 0x4DH o 77 en decimal.

Ilustración 90 Señal del Potenciómetro 1 conectado a RA0

La segunda señal es la correspondiente al potenciómetro 2, conectado en la entrada RA1 del PIC esclavo. Esta entrada registra un valor de 0xF5H o 245 en decimal.

Ilustración 91 Señal del segundo potenciómetro conectado a la entrada RA1

La tercera señal es la del potenciómetro 3, conectado a la entrada RA2 del PIC esclavo, esta señal es la que se observa en el bus I2C y cuyo valor numérico se presenta en el LCD del sistema maestro.

Ilustración 92 Señal del potenciómetro conectado a RA2 del PIC esclavo

Al igual que en el resto de comunicaciones I2C, que se presentaron como ejemplos, es necesaria una señal de parada o STOP para finalizar el proceso de comunicación.

4.8.7 Programa No. 5

Ver ANEXO 3.

4.9 Transmisión – Recepción con rfPIC

Empleando el kit Microchip Development Tools, realizar el análisis de la transmisión de datos mediante los RF PIC's. Para la práctica se desea enviar las señales de dos potenciómetros de por medio de dos pulsadores en la tarjeta de Transmisión y en la tarjeta Receptora se deberá encender los leds correspondientes, indicando el valor enviado por el potenciómetro.

4.9.1 Información requerida.

Para la realización de esta práctica, se hace necesario el conocimiento de lo que son los rfPIC's. A continuación se hará un pequeño resumen del kit a emplear junto a una breve descripción de los microcontroladores empleados.

Microchip Technology, presenta una solución completa para sistemas de comunicaciones unidireccionales en RF de corta distancia. Se compone de tres microcontroladores con transmisores integrados y dos receptores. Los nuevos dispositivos proporcionan soluciones que soportan bandas de frecuencia de 260 a 930 MHz.

4.9.2 Kit de desarrollo rfPIC

4.9.1 Introducción

El kit de desarrollo rfPIC (rfPIC Development Kit1), mostrado en la Ilustración 93 provee a los ingenieros de diseño una forma fácil de evaluar un sensado remoto unidireccional y los enlaces de control inalámbrico, basados en los elementos rfPIC12F675 y rfRXD0420.

Consiste de bloques modulares para diferentes transmisores y receptores que pueden ser empleados para sistemas prototipo o para evaluar las diferentes alternativas al usar productos Microchip.

Los módulos receptores están basados en el elemento rfRXD0420, y están disponibles en dos opciones de frecuencia: 315 MHz ASK y 433 MHz ASK. Estos módulos se conectan directamente al tablero de desarrollo, ofreciendo de esta forma una manera fácil de evaluar los diferentes módulos de recepción con microcontroladores Microchip FLASH de 8 y 14 pines, así como la interfaz USB al conectarse al PC. Estos módulos están disponibles para la venta en forma separada para permitir el desarrollo de prototipos basados en el mismo módulo, sin la necesidad de realizar un diseño RF. Los archivos de diseño de estos módulos están disponibles para una fácil integración de los diseños al sistema.

Los módulos transmisores están basados en el rfPIC12F675 y soportan el mismo formato de frecuencia y modulación que los receptores. Los módulos transmisores presentan botones de entrada para funciones de control remoto, así como entradas analógicas para permitir la evaluación del módulo A/D y el comparador en el rfPIC12F675.

4.9.2 Características

Los aspectos concernientes al Kit de desarrollo rfPIC Kit 1 incluye:

- Un pequeño circuito en un tablero de 3^2 x 4.5^2 (9x20.25 pulgadas), con un tablero extra a fin de desarrollar prototipos, el cual es desprendible.

- Facilidad en el uso de la interfaz Windows®programming para programar los elementos Microchip, de la familia de los microcontroladores de 8/14 pines FLASH

4.9.3 Requerimientos del sistema

- Un PC con sistema operativo compatible con Intel Pentium®class o un procesador superior o equivalente.

- Mínimo 16 MB RAM
- Mínimo 40 MB disponibles en Disco duro
- CD ROM
- Puerto USB disponible
- Microsoft Windows®98, Windows NT®4.0, Windows
- 2000 o Windows XP
- Soporte para FLASH PICmicro®products de 8/14-pines, incluyendo: PIC12F629, PIC12F675, PIC16F630, PIC16F676, rfPIC12F675 y rfRXD0420/0920.

Ilustración 93 rfPIC Development Kit1

Combinando los transmisores/microcontroladores **rfPIC12F675** con los receptores **rfRXD0420** o **rfRXD0920**, los usuarios pueden crear una conexión de comunicación unidireccional inalámbrica para aplicaciones de control integrado.

Los receptores a su vez se pueden combinar con los dispositivos **rfPIC™** existentes para diseños de sensores y control remotos.

Los receptores RF, superheterodinos de conversión única, en la banda UHF, **rfRXD0420** (300-450 MHz) y **rfRXD0920** (850-930 MHz) permiten recibir datos a una velocidad máxima de 80 kbps. Tiene un consumo en reposo de 100nA, con un rango de funcionamiento desde 2,5V hasta 5,5V. En modo activo

consume desde 6,5mA hasta 8,2mA para el **rfRXD0420** y de 7,5mA a 9,2mA para el **rfRXD0920**.

Los microcontroladores **rfPIC** son dispositivos **PICmicro®** de 20 pines con un transmisor RF en la banda UHF, y una potencia de salida de 6dBm, integrados en el mismo encapsulado.

Los **rfPIC12F675K** (260-350MHz), **rfPIC12F675F** (390-450MHz) y **rfPIC12F675H** (850-930MHz) permiten transmitir a una velocidad de datos máxima de 40 kbps, un consumo en reposo de sólo 100nA y un rango de funcionamiento desde 2,0V hasta 5,5V. Los microcontroladores contienen 1,8 Kbytes de memoria de programa **Flash**, 64 bytes de **RAM** y 128 bytes de **EEPROM**. Estos dispositivos tienen otras características como son: un comparador analógico y 4 canales de A/D de 10 bits

En el proceso de transmisión se emplea el rfPIC 12F675, con distintas configuraciones que se hacen a sus registros, a continuación se hará un resumen de los registros modificados para el funcionamiento del programa en la tabla 10.

Registro	Función
GPIO/TRISIO	Configuración de Puertos I/O
OSCCAL	Calibra el oscilador interno
CMCON/VRCON	Configura módulo Comparador
ADCON0/ANSEL	Configura Conversor A/D
T1CON	Configura Timer 1
WPU	Configura Registro de Weak Pull up
OPTION_REG	Configura bits de control
PIE1	Configura los bits de interrupción periférica
INTCON	Habilita/Deshabilita las banderas de manejo del Timer0

Tabla 10 Registros empleados para la TX

El proceso de recepción hace uso de los mismos registros que en la etapa de transmisión, únicamente cambia el valor de prescaler, 1:2, y la configuración de entradas/salidas como digitales o análogas. El integrado 16F676 posee dos puertos A y C para que sean configurados como el usuario decida.

4.9.3 Diagrama general de Funcionamiento

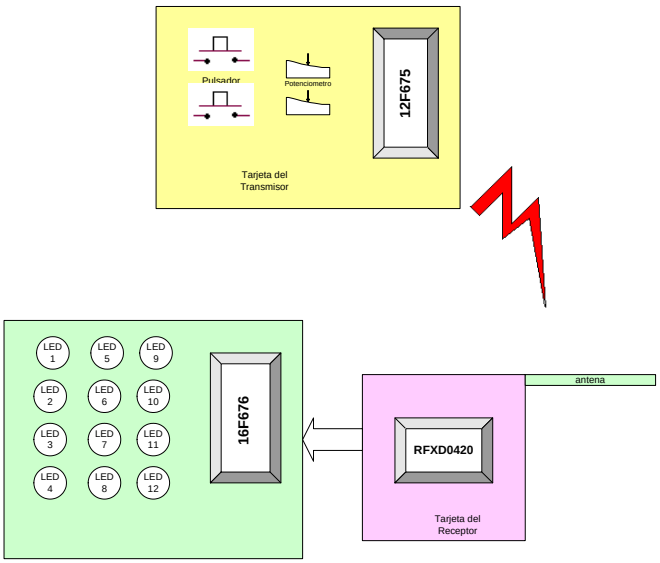

Ilustración 94 Diagrama de funcionamiento del RF KIT

Básicamente la operación a realizarse es la siguiente: desde el módulo transmisor se presionara uno de los pulsadores, la señal se enviara al módulo receptor y este tendrá la función de recibir la información y pasarla al PIC KIT, donde mediante el integrado 16F676 se presentará la información en los leds indicadores.

4.9.4 Diagramas de Flujo

Programa de transmisión xtmit.asm

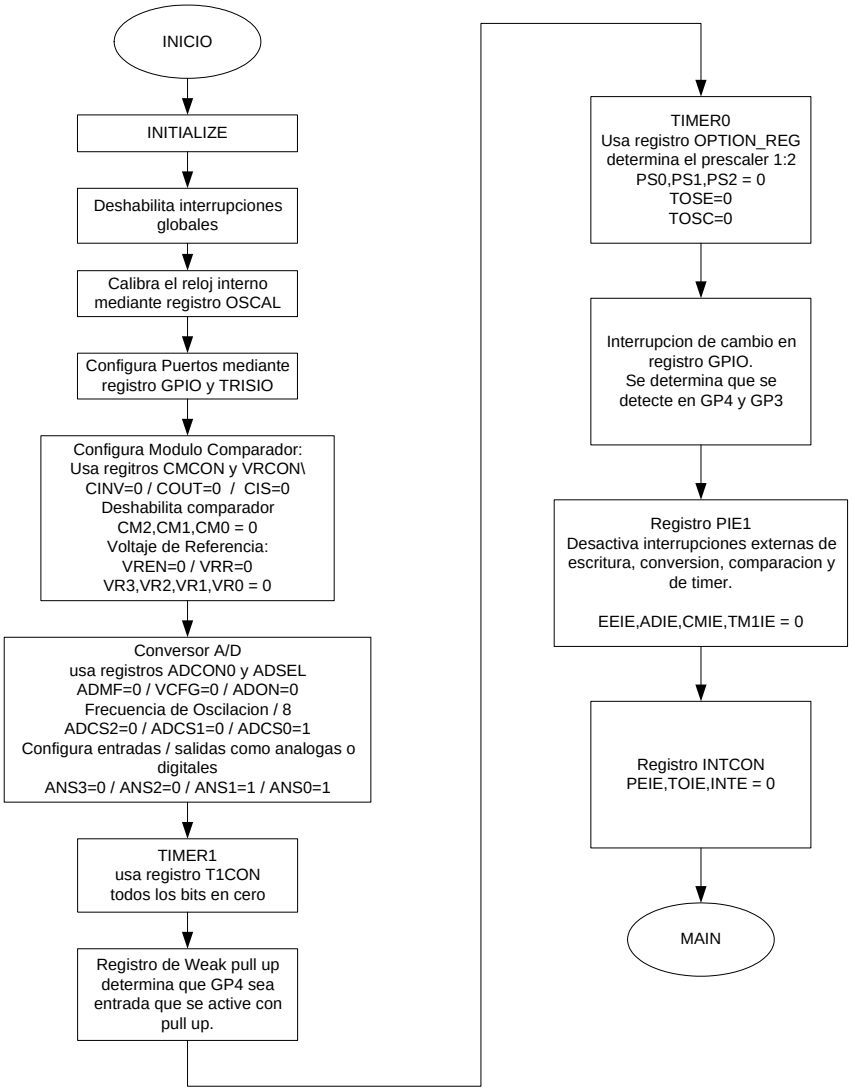

Ilustración 95 Diagrama de flujo del programa xtmit_demo - inicialización

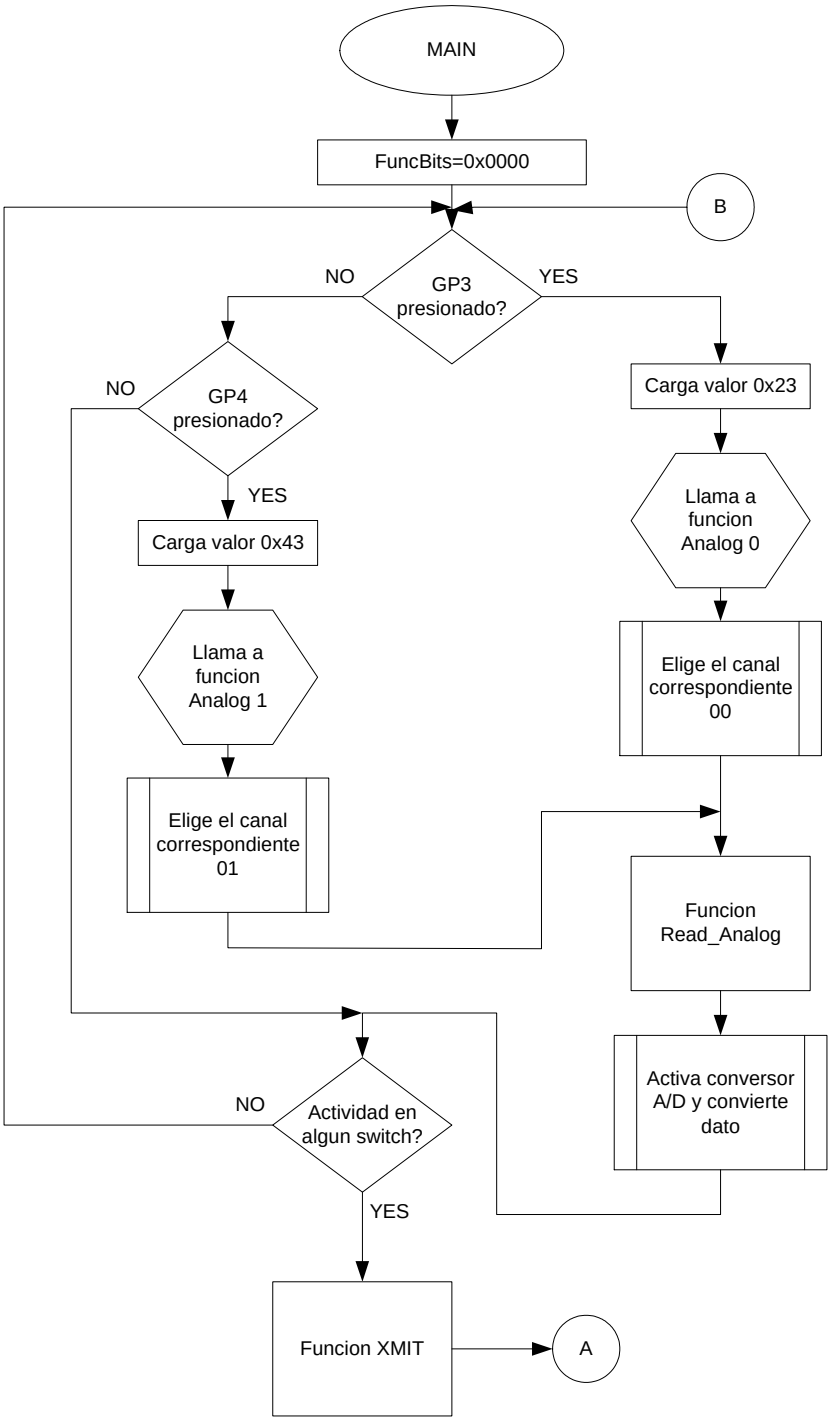

Ilustración 96 Diagrama de Flujo del programa xmit_demo – subrutina Main

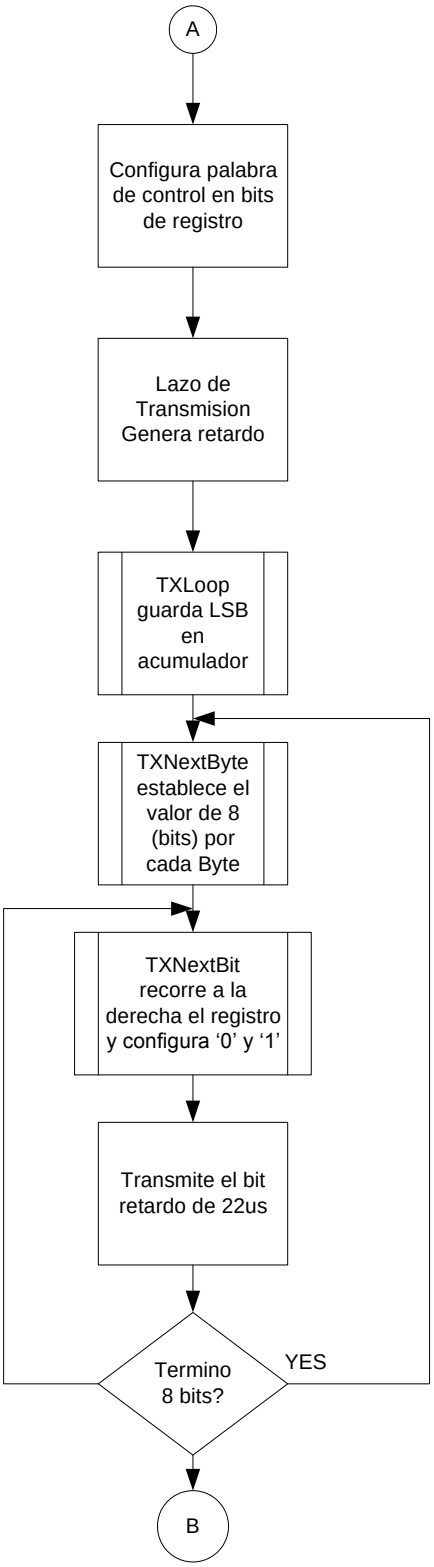

Ilustración 97 Diagrama de Flujo del programa xmit_demo – subrutina xmit

Programa de Recepción rcvr.asm

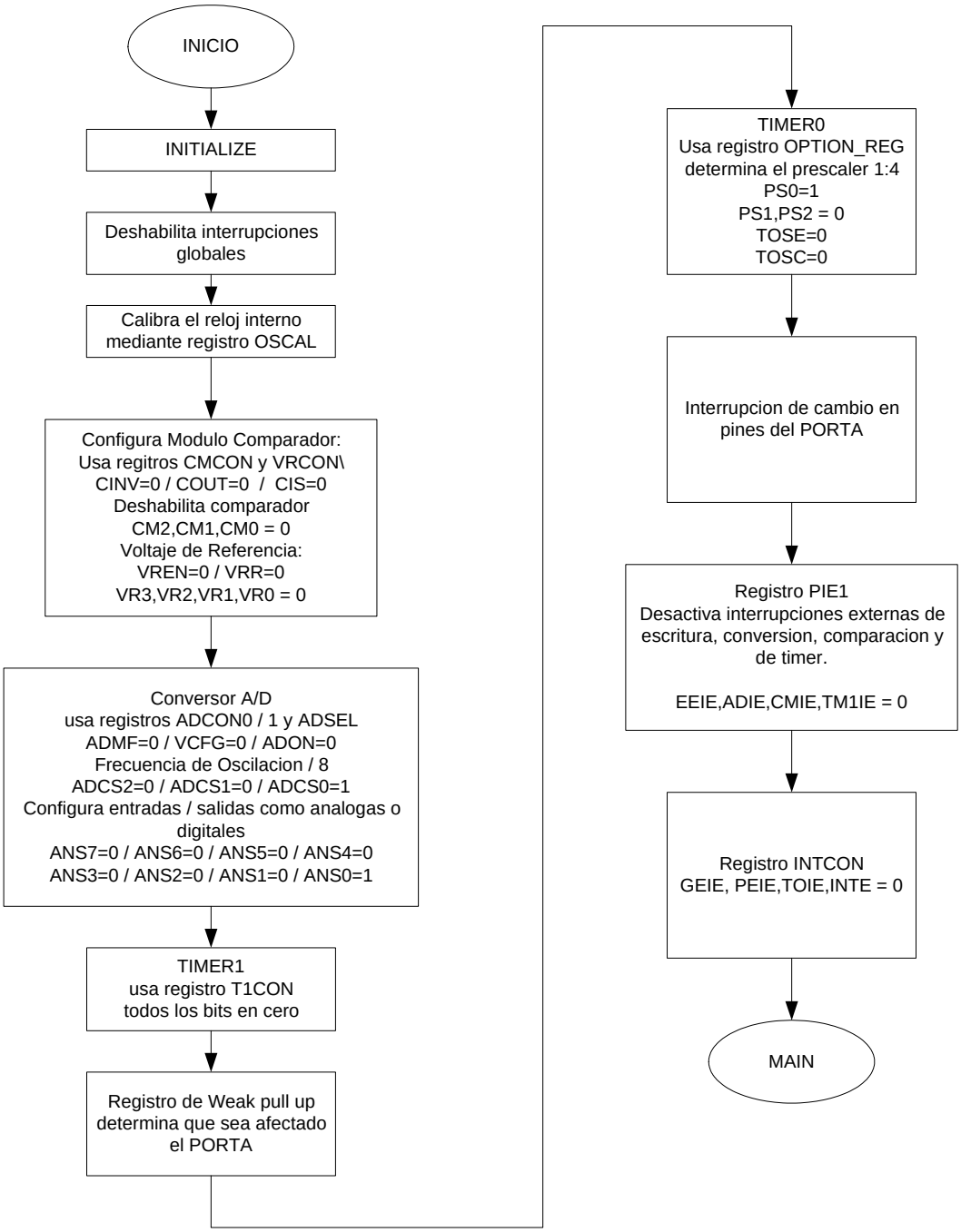

Ilustración 98 Diagrama de flujo del programa rcvr_demo – Inicialización

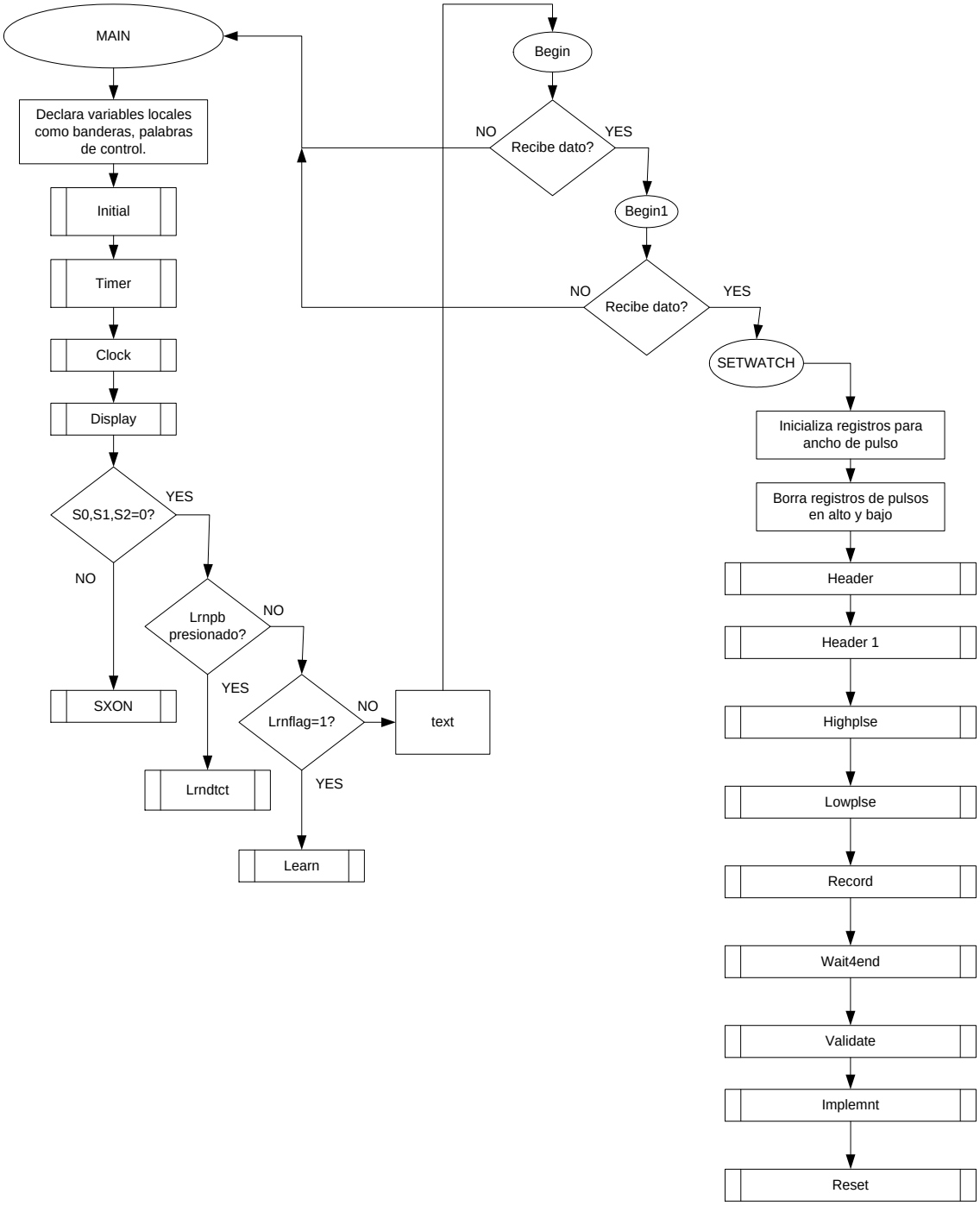

Ilustración 99 Diagrama de flujo del programa rcvr_demo - programa principal

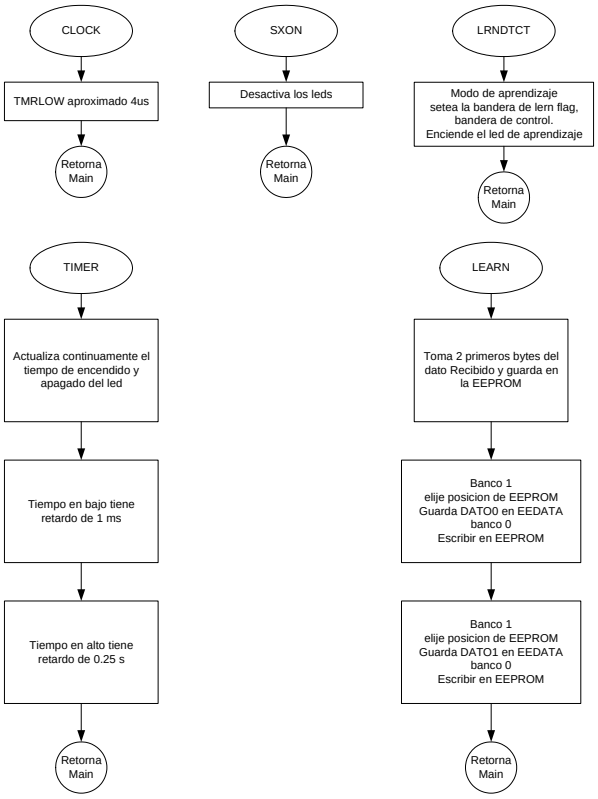

Ilustración 100 Diagrama de flujo del programa rcvr_demo – subrutinas 1

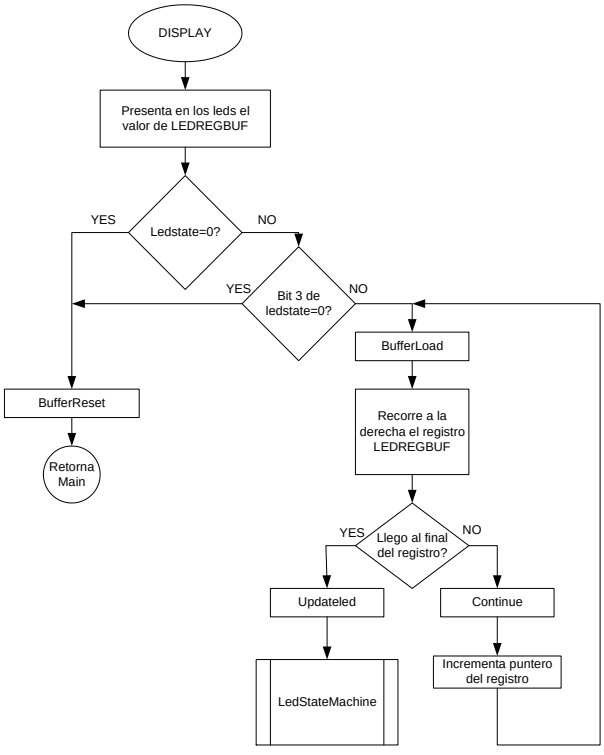

Ilustración 101 Diagrama de flujo del programa rcvr_demo – subrutinas 2

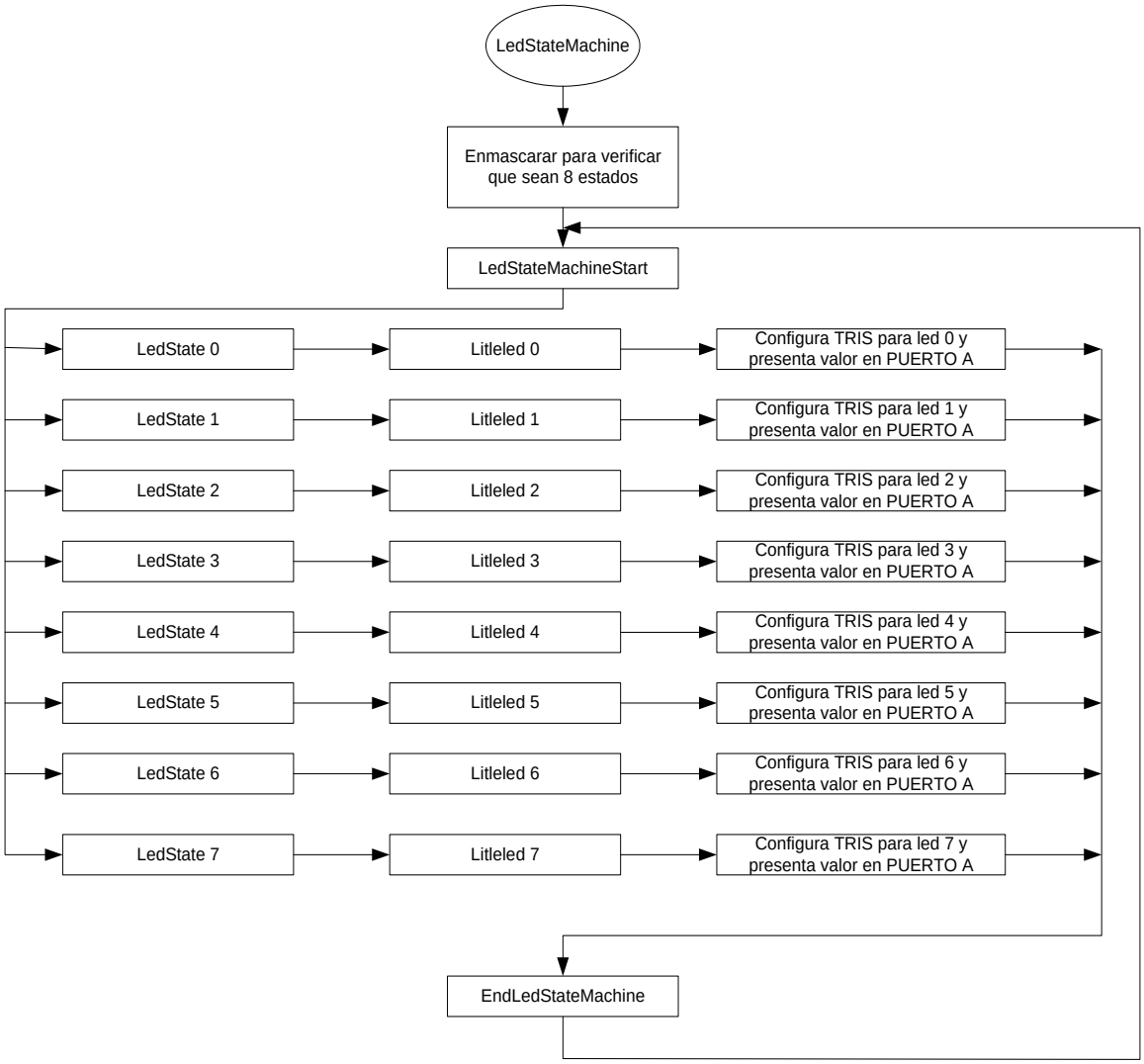

Ilustración 102 Diagrama de flujo del programa rcvr_demo – subrutinas 3

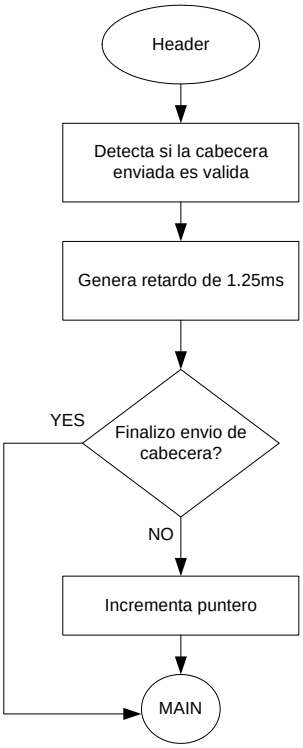

Ilustración 103 Diagrama de flujo del programa rcvr_demo – subrutinas 4

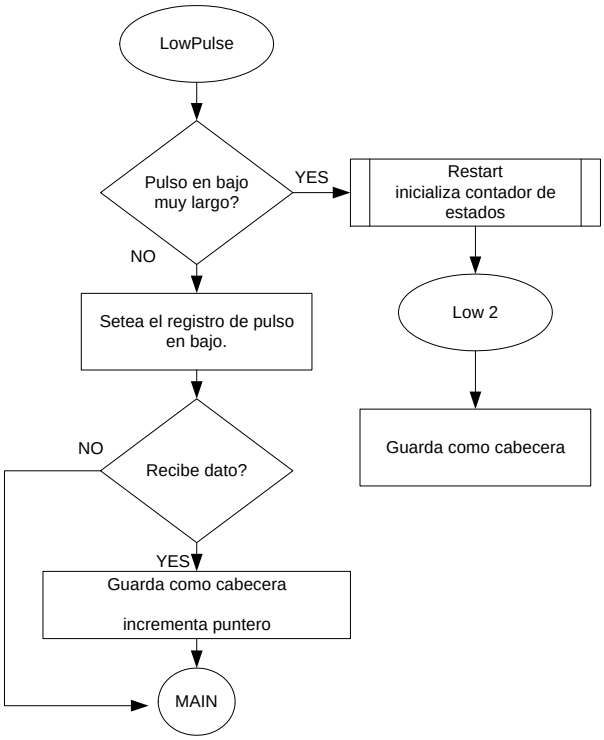

Ilustración 104 Diagrama de flujo del programa rcvr_demo – subrutinas 5

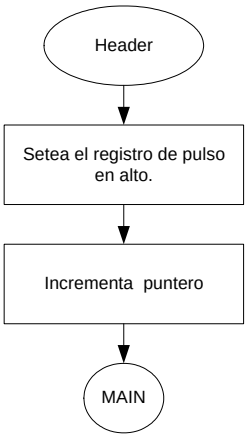

Ilustración 105 Diagrama de flujo del programa rcvr_demo – subrutinas 6

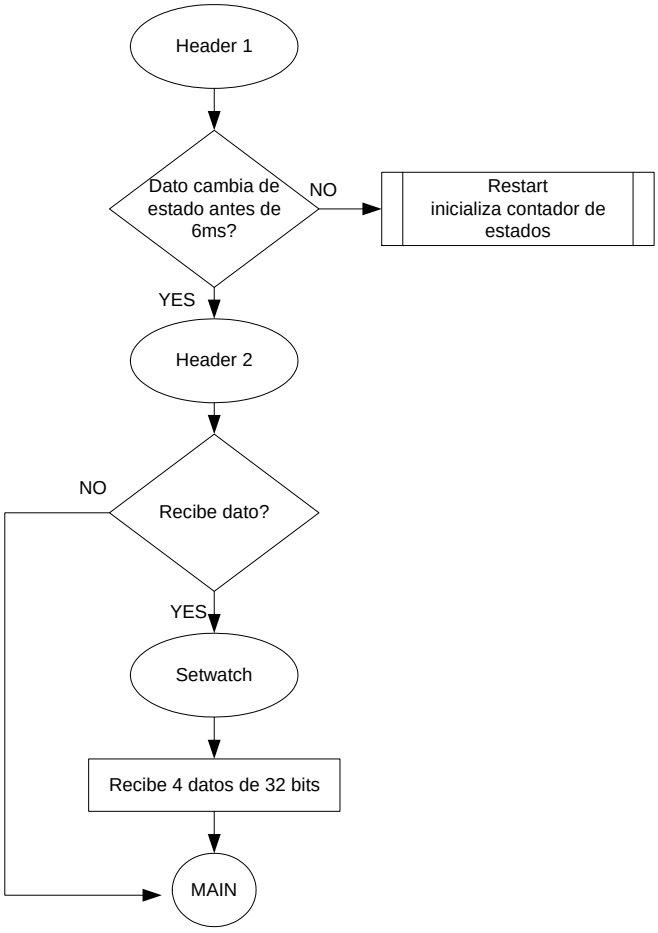

Ilustración 106 Diagrama de flujo del programa rcvr_demo – subrutinas 7

4.9.5 Funcionamiento del Programa

4.9.5.1 Etapa de transmisión

Este programa hace referencia a una aplicación de control y manejo de señales analógicas. Cuando un botón del módulo de transmisión es presionado, el led correspondiente en el PICKit se enciende. Al presionar el botón GP3 del transmisor, se encenderá el LED D0. Al presionar el botón GP4, se encenderá el LED D1.

Se inicia el programa con la configuración de los bits, luego se procede a la declaración de variables que manejaran registros, las direcciones que ocupan son las correspondientes a los registros de propósito general, esto es, desde la dirección 0x20.

De igual forma se definen nombres para determinados bits de los registros, como el caso del registro GPIO cuyos bits quedaran configurados de la siguiente manera:

U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
bit 7							bit 0
—	—	RFENA	PB4	PB3	TXD	POT1	POT0
—	—	0	1	1	0	1	1

Ilustración 107 Registro GPIO

GPIO es un puerto bidireccional de 6 bits de longitud. Su registro correspondiente es TRISIO. Al setear un bit del TRISIO como 1 hace al pin GPIO correspondiente una entrada, mientras que si se coloca un ‘0’ se establecerá como salida. La excepción es el pin GP3, pues es una entrada y el bit respectivo en el registro TRISIO será leído siempre como ‘1’.

En la práctica, cada uno de los bits del registro GPIO tienen un nombre asignado, dependiendo de la función que van a cumplir, es así que se observa potenciómetros (POT0, POT1), pulsadores (PB3, PB4), etc. A cada uno de estos bits se los determina como entradas o salidas al configurarlos con ‘1’ o ‘0’, respectivamente.

Los bits del registro quedan configurados como entradas o salidas, las entradas corresponderán a las señales de los potenciómetros y a los pulsadores, mientras que la salida será considerada la salida de datos en modulación FSK, y una salida de un led indicador.

El primer paso a ejecutar es la inicialización de los módulos internos del PIC. Así tenemos que inicializar el oscilador interno mediante el registro OSCCAL. Luego el módulo comparador haciendo uso de los registros CMCON en el banco 0 y VRCON en el banco 1.

U-0	R-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	COUT	—	CINV	CIS	CM2	CM1	CM0
bit 7							bit 0
—	0	—	0	0	1	1	1

Ilustración 108 Registro CMCON

Con el registro CMCON se configura la opción de no comparador, mediante el seteo de los bits CM0:CM2 con valores de ‘1’. El VRCON configura el voltaje de referencia para esta aplicación se configura con 00h y esto significa que no se usa voltaje de referencia.

El conversor Análogo – Digital permite la conversión de cualquier señal análoga con representación de 10 bits binarios. Se configura con los registros ADCON (Registro de control A/D) y ANSEL (Registro de selección análoga). Se determina el tiempo de conversión del reloj en Fosc/8 y se configuran los pines del GPIO como entradas análogas o digitales.

Para la aplicación el canal 0 está configurado como entrada análoga y se usa una frecuencia conversión de FOSC/8.

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	VCFG	—	—	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0
0	0	—	×	×	×	×	0

Ilustración 109 Registro ADCON0

U-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1
—	ADCS2	ADCS1	ADCS0	ANS3	ANS2	ANS1	ANS0
bit 7							bit 0
—	0	0	1	0	0	0	1

Ilustración 110 Registro ANSEL

El registro de control del TIMER1, T1CON, es usado para habilitar o deshabilitar el módulo y seleccionar diferentes características del TIMER1.

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0
—	0	—	0	0	0	0	0

Ilustración 111 Registro T1CON

El registro de Weak Pull up, WPU permite conectar un pull up en cada pin del GPIO, excepto GP3, y tienen una configuración individual para activar o desactivar esta característica. Los pull up son automáticamente desactivadas cuando el pin del puerto se configura como salida. Para el programa es necesario habilitar el pull up de GP4. GP3 no es necesario configurar porque ese pin siempre se lo considera entrada.

U-0	U-0	R/W-1	R/W-1	U-0	R/W-1	R/W-1	R/W-1
—	—	WPU5	WPU4	—	WPU2	WPU1	WPU0
bit 7							bit 0
—	—	0	1	—	0	0	0

Ilustración 112 Registro WPU (Weak Pull Up)

El registro OPTION_REG contiene los bits de control que configuran el weak pull up de GPIO, la interrupción externa, el TMR0 y el TMR0/WDT prescaler.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{GPPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0
—	1	0	0	0	0	0	0

Ilustración 113 Registro OPTION_REG

IOC es el registro de Interrupción por cambio; cada uno de los pines GPIO son individualmente configurados con ‘1’ para determinarlos como pines de interrupción por cambio.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0
bit 7							bit 0
—	—	1	1	1	1	1	1

Ilustración 114 Registro IOC

Entonces se determina que GP4 y GP3 sean señales de interrupción por variación en sus pines.

Registro de habilitación de interrupciones periféricas, PIE1, es un registro que contiene los bits de activación correspondientes, por ejemplo la escritura en EEPROM, interrupción por conversión A/D, por el comparador o por el desborde del TMR1.

R/W-0	R/W-0	U-0	U-0	R/W-0	U-0	U-0	R/W-0
EEIE	ADIE	—	—	CMIE	—	—	TMR1IE
bit 7							bit 0
0	0	—	—	0	—	—	0

Ilustración 115 Registro PIE1

Registro de control de Interrupciones, INTCON, contiene las banderas de activación o desactivación del TMR0, interrupción externa, interrupción de periféricos.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE	T0IE	INTE	GPIE	T0IF	INTF	GPIF
bit 7							bit 0
–	0	0	0	1	–	–	–

Ilustración 116 Registro INTCON

Ahora se analiza el programa principal.

En general mediante una función SCANPB se realiza la verificación que indica si se ha presionado el pulsador 1, en caso de no hacerlo se evalúa el siguiente pulsador. Cuando se presione el pulsador la señal será aceptada y se llamara a una función para leer esa señal análoga, para cualquiera de los dos casos, esa función es READ_ANALOG_AN0.

Mediante esta función se elige el canal análogo con el que se va a trabajar, que para este caso es AN0. Luego se procede a encender el módulo conversor A/D y a convertir la señal, finalmente se verifica que la transformación haya sido terminada y se apaga el módulo conversor.

Cuando se termina la conversión se procede a llamar a la función para transmitir el buffer respectivo. El primer proceso es llenar el buffer correspondiente; se cargan los valores en las variables declaradas al principio del programa, se guardan valores en CSR0, CSR4, CSR5, CSR6, CSR7 y CSR8. El dato se guarda en CSR2 y CSR3. Los bits de la palabra de control se guardan en CSR1.

A continuación se pasa al lazo de transmisión, donde se hace uso de un retardo, cuyo código genera un delay de 22 us. Luego se envía un patrón de 72 bits. Se carga el primer valor (LSB) en el acumulador y se emplea el direccionamiento indirecto para ir rotando hacia la derecha cada bit. Existen

además 2 funciones que configuran los tiempos de ‘0’ y ‘1’ que reconoce el integrado, de la forma en que se muestra en la figura siguiente:

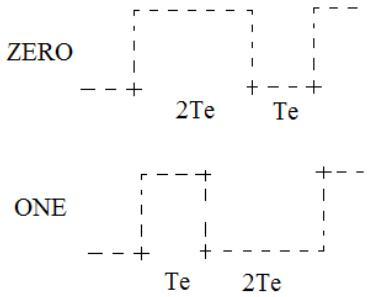

Ilustración 117 Configuración de '0' y '1'

Una vez que se configuran los tiempos de alto y bajo, se procede a la transmisión del dato por el pin de FSK_{OUT}, seguido siempre de un delay y verificando si se transmitieron los 8 bits. Una vez transmitidos todos los bits se procede a regresar a la función para transmitir otro byte.

Si la transmisión de todos los bytes ha culminado se sigue verificando el estado de los pulsadores continuamente.

4.9.5.2 Etapa de Recepción

En esta sección se analizara la tarea que realiza el integrado 16F676, que tiene grabado el programa rcvr.asm.

Básicamente la señal modulada se envía hacia la antena del receptor y a la tarjeta que contiene el PIC rfRXD0420. Este PIC tiene la tarea de recibir la señal y demodularla, una vez que esto es realizado, la señal se envía al integrado 16F676 para la presentación en los leds indicadores.

Al igual que sucede con el programa encargado de la transmisión, se inicia con un proceso de configuración y declaración de variables y constantes. Se manejan los mismos registros y de manera similar al transmisor. Esta inicialización puede ser observada en los diagramas de flujo respectivos.

El programa se basa en la presentación de la señal en los leds de la tarjeta del PICKit. La forma en la que se recibe el dato se analizara enseguida.

En las posiciones de memoria correspondientes a las de propósito general, se declaran variables y registros para el manejo de las señales y de los datos a recibir. Con referencia a los puertos se empleara el puerto A, que es un puerto bidireccional de 6 bits, su registro correspondiente en el banco 1 es el TRISA. Como ya se sabe, al configurar un ‘1’ se considera el pin como entrada; si se configura como ‘0’ se establece como salida. La excepción es el RA3 que es entrada y siempre será configurado como ‘1’.

El puerto C es un puerto de propósito general que tiene 6 pines bidireccionales. Estos pines pueden ser configurados como entradas o salidas tanto digitales como análogas; en funciones adicionales para el comparador o conversor A/D.

La subrutina TIMER se encarga de actualizar continuamente las variables que registran el tiempo de los pulsos en alto y en bajo para el encendido o apagado de los leds.

LRNDTCT, es una subrutina que detecta cuando un pulsador es presionado, colocando al programa en modo de aprendizaje, el cual dependerá de la bandera LERN, inicialización del registro STATCNTR, encender el led de aprendizaje y borrar el contenido de las variables que manejan los timers, SX1TMR y SX2TMR, correspondientes a los que registran el tiempo del pulso en alto y bajo para el encendido o apagado de los leds.

La función Display presenta la información de los registros que almacenan el dato en los leds indicadores, LEDREGBUF. Siendo el bit más significativo el

D7 y el menos significativo el D0. La presentación del dato se lo hace de bit en bit.

El registro LEDREGBUF vuelve a ser cargado luego de 8 llamadas a la función Display, esto significa que tiene 8 bits por dato, y una vez que los 8 bits se han presentado, el registro se carga con otro byte de información.

El programa principal, básicamente se encarga de recibir el dato y de enviarlo a una subrutina para presentación. Este proceso para llegar a la presentación de los datos en los leds, se ejecutara en base a una variable contador de estados, que se va incrementando luego de cada proceso.

Es importante indicar que el formato de recepción del dato es el siguiente:

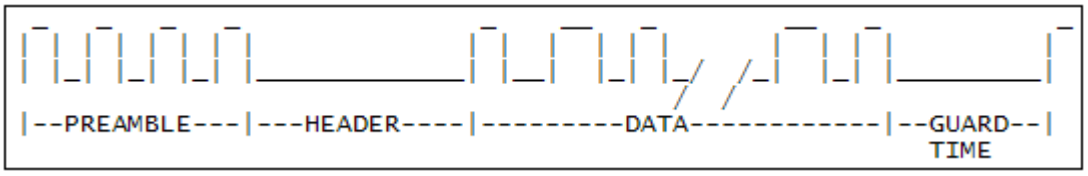

Ilustración 118 Formato del dato a presentar

Como se puede observar en la ilustración, se envía una cabecera, luego el dato y finalmente un tiempo de espera para el siguiente dato.

En el programa como tal, se inicia verificando que un dato se haya recibido, se procede a configurar el ancho de pulso en alto y bajo tanto del dato como de la activación o desactivación de los leds. Se crea una cabecera y verifica que el tiempo en bajo sea el adecuado para que sea considerada como dato valido.

Para el reconocimiento de la sección de Time Guard, se verifica la duración del pulso en bajo y en caso de ser un tiempo mayor a 6ms, será considerado como Time Guard.

Gracias a la función Record, se permite guardar los 4 datos enviados. Estos datos son recibidos de bit en bit, completando 32 bits en total. Cuando el proceso de grabación de los datos se finaliza, se procede a realizar un

RESTART para reiniciar el proceso de recepción, borrando las banderas de contador de estados.

Se verifica la palabra de control y se reinicia el proceso.

4.9.6 Programa No. 6

Ver ANEXO 4.

CAPITULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

Si bien existen muchos protocolos sincrónicos disponibles para su implementación y cada protocolo tiene sus ventajas y desventajas, el protocolo I2C sobresale cuando de conectar dispositivos a nivel de integrados se trata.

En el proyecto desarrollado se han cumplido varios objetivos parciales que a continuación se describen:

- ✓ El funcionamiento básico del protocolo I2C en operación de lectura desde el maestro, consiste en emitir en secuencia las señales de START, dirección del esclavo con el ultimo bit indicando lectura (‘1’), recepción de datos y envío de la señal de ACK con cada dato recibido. La parada de la comunicación se realiza cuando el maestro envía una señal de NACK seguida de la señal de STOP.

- ✓ Para la operación de escritura en el esclavo, se emite en secuencia las señales de START, dirección del esclavo con el ultimo bit indicando escritura (‘0’), envío de los datos y envío de la señal de ACK al finalizar la escritura de los datos en el esclavo, seguida de la señal de STOP para finalizar la comunicación.
- ✓ En la comunicación I2C existen solo dos líneas de comunicación que son requeridas: una línea de datos serial (SDA) y una línea de reloj

serial (SCL). Estas líneas conectan todos los elementos del bus; mientras que en comparación con la comunicación USART es necesario conectar algunas líneas por cada dispositivo.

- ✓ Se inicializó el RTC o Reloj de Tiempo Real, mediante la comunicación I2C, haciendo uso de las dos líneas del bus, escribiendo la palabra de configuración en el esclavo.

- ✓ Se leyeron los datos de horas, minutos y segundos desde el esclavo (RTC), presentando esta información en el LCD del sistema SDP877.

- ✓ Mediante el bus I2C se envió información al esclavo para igualar el reloj, guardando estos valores, ingresados por dip switch, en los registros correspondientes del RTC. Además cuando no se quiere escribir datos la información de horas, minutos y segundos se presenta en el LCD del sistema SDP877.

- ✓ Se hizo funcionar el programa principal del archivo AN736, “An I2C Network Protocol for Environmental Monitoring”, proporcionado por Microchip, luego de varios cambios al software pues presentaba errores. El programa es un sistema de monitoreo de varias señales analógicas y digitales.

- ✓ Se realizo un sistema de presentación visual en matriz de leds 20x7. El sistema consiste de una transmisión serial RS232 del mensaje desde la PC hacia un PIC maestro, y desde el PIC maestro hacia un PIC esclavo mediante la red I2C.

- ✓ Cada elemento conectado al bus es direccionable desde software por una única dirección y una simple relación maestro/esclavo existente en todo momento; el maestro pueden operar como maestro transmisor o maestro receptor.

- ✓ Existe la posibilidad de realizar un bus multi-maestro, que incluye detección de colisión y arbitraje para prevenir la corrupción de datos, cuando dos o más maestros inician la transferencia de datos simultáneamente.

- ✓ Este tipo de bus no puede alcanzar velocidades similares a las conseguidas con una estructura de bus en paralelo; pero requieren mucho menos cableado y el hardware es mucho más sencillo. El sistema de red de comunicación I2C es de bajo costo y de fácil implementación.

- ✓ Los distintos dispositivos conectados al bus se comunican entre sí mediante este protocolo que evita el bloqueo de información y garantiza la comunicación entre todos ellos.

- ✓ La red I2C tiene posibilidades de expansión, lo que nos indica que el sistema es escalable mediante las dos líneas de comunicación bidireccionales SDA y SCL, claro, tomando en cuenta la longitud máxima del bus y que no es mayor a 9.1 pies que equivale a 2.77 mts.

- ✓ Es un sistema limitado en distancia, pues la longitud del bus que conecta los diferentes esclavos tiene una longitud máxima que depende de las características del cable.

- ✓ Los elementos empleados en la construcción de este módulo de comunicación I2C, pueden ser conseguidos fácilmente en el mercado local, de esta forma se pueden tener expectativas de una producción a alto nivel.

Los rfPIC’s son elementos que nos permiten brindar una solución completa para sistemas de comunicaciones unidireccionales en RF de corta distancia. Las aplicaciones que se pueden dar a los mismos son múltiples, y de acuerdo al estudio realizado se llego a establecer lo siguiente:

- ✓ El programa de transmisión codifica los bits a transmitirse en un formato de modulación de ancho de pulso. Para transmitir un ‘1’ lógico la onda es la mitad que para transmitir un ‘0’ lógico. La tasa de transmisión de los bits se establecen mediante el potenciómetro GP0 para niveles de transmisión máximo, alto, medio o bajo, desde 1.27 a 52.6 GBps.

- ✓ El dato a transmitirse pasa por un proceso de conversión A/D, para luego poder ser enviado.

- ✓ El rfRXD0420 tiene un pin de salida para la señal remodulada. Esta señal consiste de un preámbulo, luego una cabecera (pulso en bajo entre 1.25 ms a 6 ms) y finalmente el dato en forma serial para terminar con una tiempo de espera de 1.25 ms.

- ✓ El PIC rfRXD0420, es el integrado receptor, que realiza la función de recibir el dato modulado, demodularlo y enviarlo a otro integrado (16F676) para que el proceso de presentación se lleve a cabo.

- ✓ El rfRXD0420 no es integrado con encapsulado dip, no es posible grabar ningún programa en él, solamente mediante la configuración de hardware se logra su funcionamiento. Para las modificaciones necesarias para una aplicación determinada se requiere la ayuda de otro integrado, el PIC 16F676.

- ✓ El rfPIC 12F675, que es el integrado transmisor, puede ser modificado al usar la versión DIP usando el ICPROG, mientras que para programar el rfPIC de montaje superficial se requiere usar la placa principal del Flash Starter Kit.

- ✓ En el Kit de Desarrollo, los puertos del 12F675, están ocupados en su totalidad por potenciómetros, pulsadores, leds, etc; por lo que cualquier cambio a realizar implica un cambio en la estructura de la tarjeta, mediante cables o desoldando los puntos de conexión.

- ✓ El sistema rfPIC es un sistema cerrado, que no permite mayores modificaciones debido a que los integrados están soldados a la tarjeta y para la grabación de otra aplicación es necesaria la implementación y construcción de otra tarjeta.

- ✓ La obtención de los materiales para la comunicación mediante rfPIC's presenta problemas en facilidad de obtención. El elemento transmisor, 12F675, se lo puede conseguir en el mercado local, pero el elemento receptor rfRXD0420, encargado de la demodulación de la señal solo se lo encuentra en el mercado internacional.

5.2 Recomendaciones

- Tener mucho cuidado al colocar los PIC's en el módulo, siguiendo la orientación correcta, pues una falla de este tipo puede causar daños irreparables a los PIC's.

- Asegurarse que el RTC luego de ser inicializado quede energizado por una batería de 3 [V] o en su defecto se puede emplear una fuente variable. En caso de no hacerlo, cuando se retire la energía del sistema, el RTC quedará desconfigurado.

- Si se desea implementar una aplicación con más caracteres en la presentación de mensajes, se puede construir exactamente el mismo sistema esclavo y realizar la conexión I2C a cada esclavo. El numero de esclavos estará limitado por la longitud del cable. Cada esclavo tendrá su dirección y el programa maestro debe modificarse para direccionar a cada esclavo y enviar la parte correspondiente del mensaje. De esta forma se tendrá un sistema más atractivo para cualquier empresa que requiera presentar información de este tipo.

- Para la comunicación RF se recomienda emplear el integrado 12F675 con encapsulado DIP para futuras aplicaciones, a fin de tener facilidad de conexiones y ampliar el alcance del proyecto. Es necesaria además la construcción de la antena, para lo cual adjunto la documentación necesaria.

GLOSARIO DE TÉRMINOS

ACK	Acknowledge
BRG	Baud Rate Generador – Generador de tasa de Transmisión
BSSP	Basic Synchronous Serial Port – Puerto serial sincrónico básico
EEPROM	Electrically Erasable Programmable Read – Memoria de lectura programable, borrable eléctricamente.
F/W	Firmware – Marca
I2C	Inter-Integrated Circuit – Circuito Inter Integrado
ISR	Interrupt Service Routine- Rutina de Servicio de Interrupción.
MCU	Microcontroller Unit – Unidad de Microcontrolador
MSSP	Master Synchronous Serial Port – Puerto serial sincrónico maestro.
NACK	Not Acknowledge
SDA	Serial Data Line – Línea de datos Serial
SCL	Serial Clock Line – Línea de reloj Serial
SSP	Synchronous Serial Port – Puerto Serial Sincrónico

REFERENCIAS BIBLIOGRAFICAS

LIBROS

ANGULO, Jose Ma., ***Microcontroladores PIC. Diseño práctico de aplicaciones***, segunda edición, editorial Mc Graw Hill, Madrid, 2000, pp 231.

MANUALES

Mid-Range MCU Reference Manual,Microchip Technology Inc., Document Number DS33023

AN735, "Using the PICmicro® MSSP Module for Master I2CTM Communications", Microchip Technology Inc., Document Number DS00735

AN734, "Using the PICmicro® SSP for Slave I2CTM Communication", Microchip Technology Inc., Document Number DS00734

INTERNET

http://www.microchip.com, Hojas de datos técnicos.
http://www.sagitron.es, Buses I2C.
http://www-us.semiconductors.com/i2c/PICmicroTM, Comunicación I2C
http://www.itutech.com, manejo de elementos mediante I2C.
http://www.parallaxinc.com, información sobre PIC's

EMPRESAS

MICROCHIP TECHONOLOGY INC.
2355 W. Chandler Blvrd.

Chandler, AZ 85224 USA.

Phone: 480-792-7200