**UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO**

INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA

# PROTOCOLO Y REALIZACIÓN DE COMUNICACIÓN SERIAL DISTRIBUIDA.

**T E S I S**

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

P R E S E N T A N

**AMERICA SALDAÑA SÁNCHEZ**

**LUIS ALBERTO PAREDES SALINAS**

ASESORES: DR. VIRGILIO LOPEZ MORALES

ING. JULIO CESAR RAMOS FERNANDEZ[1]

PACHUCA DE SOTO, HIDALGO. FEBRERO DE 2006

[1] Profesor-Investigador de la Universidad Tecnológica de Tula-Tepeji.

# Agradecimientos

A

Nuestros asesores de tesis el Dr. Virgilio López Morales y al Ing. Julio César Ramos Fernández, por el apoyo y tiempo brindado

A la Maestra Angélica Espejel Rivera que gracias a su conocimiento nos guió hacia la realización del presente trabajo

Y en especial a Laura Elena Muñoz Hernández gran amiga y colaboradora en el presente proyecto.

# Índice general

# Índice de tablas

# Índice de figuras

**Glosario de términos técnicos**

ASCII.- (American Standard Code for Information). Código Americano Estándar para el Intercambio de Información basado en el alfabeto latino.

BGR.- (Baud Rate Generator). Generador de tasa de bits.

BUFFER.- Ubicación de la memoria reservada para el almacenamiento temporal de información digital, mientras espera a ser procesada.

CAN. - (Controller Area Network). Red de Area Controlada.

CMOS.- (Complementary Metal Oxide Semiconductor). Tecnología creada para compuertas lógicas.

CSMS/CA. - (Carrier Sensor Multiple Access / Collision Avoidance). Acceso Múltiple por detección de acarreo/prevención de colisión.

DCE. - (Data Communication Equipment). Equipo de Comunicación de Datos

DTE. - (Data Terminal Equipment). Equipo Terminal de Datos.

EEPROM. - (Electrical Erasable Programmable Read Only Memory). Memoria de solo lectura programable y borrable eléctricamente.

EPROM. - (Erasable Programmable Read Only Memory). Memoria de solo lectura programable y borrable.

ISO. - (International Standards Organization). Organización de Estándares Internacionales.

I2C. - (Inter Integrated Circuits). Circuitos integrados interconectados.

FULL DUPLEX. - Capacidad de un equipo para enviar y recibir simultáneamente datos.

LCD.- (Liquid Cristal Display). Pantalla de Cristal Líquido.

LSB.-(Least Significant Bit). Bit menos significativo.

MEMORIA FLASH. - Memoria que permite que múltiples posiciones de memoria sean escritas o borradas en una misma operación de programación mediante impulsos eléctricos.

MSB. - (Most Significant Bit). Bit más significativo.

NRZ. - (Non Return to Zero). No retorno a cero.

OSI. - (Open Systems Interconnection). Interconexión de sistemas abiertos

PIC.- (Peripheral Interface Controller). Controlador de interfaz de periféricos

PLC. - (Programmable Logic Controller). Controlador Lógico Programable.

PROFIBUS.- (Process Field Bus). Norma internacional de buses de campo.

PROTOCOLO. - Instrucción de procedimiento para la transferencia de datos.

PROPIETARIO.- Sentido cerrado del protocolo o estándar.

RAM. - (Read Access Memory). Memoria volátil en la que se puede leer como escribir, es decir, pierde su contenido al desconectar la energía eléctrica.

RISC.- (Reduced Instruction Set Computer). Computadores de juego de instrucciones reducido.

SCI. - (Serial Comunication Interface). Interfaz de Comunicación Serial.

SISTEMA ABIERTO. - Comunicación entre equipos de tipos y/o constructores diferentes y sus reglas de comunicación son públicas.

USART. - (Universal Synchronous Asynchronous Receiver Transmitter). Transmisor Receptor Universal Síncrono Asíncrono.

VAN. - (Value Add Network). Red de Valor Agregado.

### Contexto de la tesis

Este proyecto de tesis nace como una necesidad de contar con protocolo de comunicación entre diferentes sistemas mínimos, los cuales se hallaban monitoreando y controlando procesos de forma autónoma sin que tuvieran comunicación entre ellos.

El presente proyecto de tesis forma parte de un proyecto global llamado Automatización y supervisión en un invernadero, para lo cual se ha dividido en diferentes procesos como son: La construcción de los sensores, la manipulación de los actuadores, y las leyes de control para manejo de actuadores automáticamente. Para poder hacer todo esto necesitamos la comunicación entre sensores, actuadores y una computadora.

Es por eso que el presente proyecto de tesis se basa en la implementación de dicho protocolo y la interconexión de estos sistemas mínimos a un nodo central que soporte la cantidad de información enviada desde el sistema mínimo y hacia éste, el cual es una computadora personal.

### Objetivos

Diseñar un protocolo de comunicación serial distribuida, basado en microcontroladores, que permitan la interacción entre los diferentes sistema mínimos que monitorean y controlan diferentes variables físicas como temperatura, humedad y radiación solar que están presentes en un invernadero.

Contribuir con este protocolo para el desarrollo de la automatización de un invernadero, brindado la comunicación entre los sensores y el control de los actuadores.

**Justificación**

Se sabe que existen diferentes protocolos de comunicación serial distribuida como lo son: BATIBUS, BITBUS, EIB, FIP bus, J1850, LON, PROFIBUS, VAN, CAN., sin embargo los inconvenientes que presentan son sus altos costos, ya que son de arquitectura cerrada y por lo tanto no pueden ser modificados, además de que consumen altos recursos de cómputo y necesitan de dispositivos o componentes especiales para su implementación.

Es por eso que la necesidad de implementar sistemas de monitoreo menos complejos, confiables y más accesibles a las condiciones económicas del país ha llevado como consecuencia el desarrollo de éste proyecto de Tesis.

Así con el presente trabajo se busca obtener en un cierto grado, independencia tecnológica, además de la mayor flexibilidad para la manipulación y/o expansión de la presente arquitectura.

**Resumen**

En esta tesis se aborda los principios básicos sobre la realización de un protocolo de comunicación serial distribuida propio.

Para entender este protocolo, y su realización, se empieza por conocer un poco sobre el desarrollo de protocolos de comunicación y la incompatibilidad en las redes de comunicación industrial. También se estudia el uso de un microcontrolador, las normas de comunicación y la logística del protocolo.

En el Capítulo 1, se describe el camino de diseño que sé siguió, como se integró a la realización el modelo OSI para llegar a su elaboración final. También, se abordará el concepto de un microcontrolador y las razones por las que se escoge el microcontrolador 16F877. Además, se menciona las herramientas de software usadas para trabajar con los microcontroladores.

En el Capítulo 2, se da un panorama de los dispositivos que se utilizaron en el desarrollo del proyecto, entre los cuales se encuentran:

- Los sistemas mínimos propuestos,
- Dispositivos interconectados a los sistemas mínimos como la pantalla de LCD, los módulos de comunicación serial, las entradas analógicas y las entradas/salidas digitales.
- Y finalmente, los microcontroladores que realizan todas las tareas.

Se describen las características pertenecientes a éste, como la memoria que poseen, ya que es un circuito integrado programable, tiene registros, los cuales se describen en este capítulo.

Los dispositivos interconectados a los sistemas mínimos, como la pantalla de LCD, en la cual se puede visualizar los procesos y variables que están involucradas en éstos, los convertidores de norma RS-232 y RS-485 para la transferencia de datos.

Es así como éste capítulo, se brindan los conocimientos que en los siguientes capítulos serán de utilidad para comprender cómo se diseña e implementan estos dispositivos en el proyecto.

En el Capítulo 3, se aborda las particularidades de la norma RS-485, la cual nos menciona las reglas para tener una comunicación con más de 2 dispositivos, que para este trabajo de tesis se usan 5 módulos, a los que se denominan sistemas mínimos.

También se menciona en este capítulo los diferentes usos de 8 y 9 bits que tienen el microcontrolador 16F877, entre ellos el de detectar direcciones para poder transmitir y recibir datos con otro microcontrolador.

Otro punto importante que se abordará en este capitulo es el uso del Convertidor Analógico/Digital, con el que internamente cuenta el microcontrolador usado, ya que este convertidor es el que tiene más aplicaciones en el microcontrolador en la vida diaria. La mayoría de las mediciones obtenidas por los sensores son analógicas, las cuales requieren ser convertidas a señales digitales para ser procesadas por los sistemas de control.

Al final de este capítulo se analiza la estructura lógica que se realizó para lograr la comunicación con los diferentes módulos, además de la logística del protocolo para asignar las direcciones a cada módulo.

Finalmente el Capítulo 4, describe detalladamente al protocolo de comunicación mostrando las diferentes rutinas que se programaron en lenguaje C para configurar al PIC16F877 y los diferentes módulos de LABview que se emplearon en el desarrollo del protocolo.

# Capítulo 1

# Introducción a los protocolos de comunicación

### Introducción

En sus inicios, el desarrollo de redes sucedió con desorden en muchos sentidos. A principios de la década de 1980 se produjo un enorme crecimiento en la cantidad y el tamaño de las redes. A medida que las empresas tomaron conciencia de las ventajas de usar tecnologías de red, éstas se agregaban o expandían a casi la misma velocidad a la que se introducían otras nuevas tecnologías. Para mediados de la década de 1980, estas empresas comenzaron a sufrir las consecuencias de la rápida expansión.

De la misma forma en que las personas que no hablan un mismo idioma tienen dificultades para comunicarse, las redes que utilizaban diferentes especificaciones e implementaciones tenían dificultades para intercambiar información. El mismo problema surgía con las empresas que desarrollaban tecnologías de redes privadas o propietarias. Tecnologías propietarias significa que una sola empresa o un pequeño grupo de empresas controla todo uso de la tecnología. Las tecnologías de red que respetaban reglas propietarias en forma estricta no podían comunicarse con tecnologías que usaban reglas propietarias diferentes.

Para enfrentar el problema de incompatibilidad de redes, la Organización Internacional para la Normalización (ISO) investigó modelos de networking, a fin de encontrar un conjunto de reglas aplicables de forma general a todas las redes. En base a esta investigación, la ISO desarrolló un modelo de red que ayuda a los fabricantes a crear redes que sean compatibles con otras redes.

## 1.1. Modelo OSI

El modelo de referencia de Interconexión de Sistemas Abiertos (OSI) lanzado en 1984, fue el modelo de red descriptivo creado por ISO. Proporcionó a los fabricantes un conjunto de estándares que aseguraron una mayor compatibilidad e interoperabilidad entre los distintos tipos de tecnología de red, producidos por las empresas a nivel mundial.

El modelo de referencia OSI se ha convertido en el modelo principal para las comunicaciones por red. Aunque existen otros modelos, la mayoría de los fabricantes de redes relacionan sus productos con el modelo de referencia de OSI. Esto es en particular así cuando lo que buscan es enseñar a los usuarios a utilizar sus productos. El modelo en sí mismo no puede ser considerado una arquitectura, ya que no especifica el protocolo que debe ser usado en cada capa, sino que suele hablarse de modelo de referencia. Este modelo maneja dos aspectos fundamentales:

1. Organización en niveles o capas, y la
2. Utilización de un protocolo para la comunicación entre los niveles.

Éstos niveles permiten independizar unas funciones de otras, de forma que sea más fácil el tratamiento de datos en cada uno de los niveles.

En los protocolos y, dado que son reglas con las que se logra una comunicación, se definen los siguientes puntos:

1. Sintaxis: formato de los datos,
2. Semántica: estructura de la información para el manejo de errores,
3. Temporización: secuencias de comunicación y adaptación de velocidades.

El modelo OSI utiliza la arquitectura de niveles, concretamente de siete niveles. Se dice que es un modelo y no un estándar, porque en sí no especifica, de una forma exacta, los servicios, protocolos, normas que se deben ejecutar y utilizar en cada nivel. Las formas en las que se implementan estos servicios las describe un protocolo (Cf. [Morcillo, 2000]). Los siete niveles pueden observarse en la Tabla 1.1.

| No. de Capa | Modelo ISO/OSI |
|---|---|
| 7 | Aplicación |
| 6 | Presentación |
| 5 | Sesión |
| 4 | Transporte |
| 3 | Red |
| 2 | Comunicación de datos |
| 1 | Física |

Tabla 1.1: ISO

1. **Nivel Físico.** Se definen las características eléctricas (valores eléctricos y codificación), mecánicas (conectores), funcionales de los circuitos (que compone la tarjeta del interfaz) y los métodos para especificar los flujos de intercambio de bits (moduladores) en el medio físico (cable).

2. **Nivel de Comunicación de Datos.** Hace que el nivel físico sea seguro. Su principal objetivo es la recolección de datos de nivel de red y para su transmisión formando los tramas. Además impone los métodos de direccionamiento, códigos detectores y de recuperación de errores, y fija el orden a las tramas trasmitidas.

3. **Nivel de Red.** Son los medios para establecer, mantener y liberar la conexión, entre los extremos, a través de una red de comunicación, compuesta por nodos y caminos. Se encarga de controlar la ruta que seguirá la comunicación. Las estaciones, por medio de este nivel, dialogan con la red especificando las direcciones, obteniendo de la red el encaminamiento adecuado (a través de los nodos) y prestaciones especiales.

4. **Nivel de Transporte.** Este nivel es el responsable del intercambio de datos entre extremos. Si es orientado a la conexión, las unidades de datos de este nivel deben entregarse correctamente en secuencia, sin duplicados y sin errores. También es responsable de algunos de los servicios que solicita a la red, como retardos y tasas de errores, así como otros que le sean reclamados por el nivel de sesión.

5. **Nivel de Sesión.** Permite el intercambio de una forma organizada de datos entre usuarios extremos. Organiza y sincroniza la comunicación entre los mismos. En este nivel se solicita la conexión de una estación con otra. Este nivel proporciona tres grupos de funciones que son:

   - Tipo de diálogo (en ambos sentidos, full- duplex o con espera alternada semi-dúplex).
   - Recuperación, ante fallos.

- Y finalmente verificadores de agrupación de datos relacionados, de las claves de acceso, de comunicación entre la red y el sistema operativo.

6. **Nivel de Presentación.** Utilizado para el cambio de información, cuando se utilizan sistemas de transferencia con códigos distintos. No se ocupa de la semántica sino de la sintaxis. Se obtiene independencia de la representación de datos de los procesos de aplicación. Lo que ofrece este nivel es un conjunto de transformación, cifrado y comprensión de datos.

7. **Nivel de Aplicación.** Soporta los procesos de aplicación de los usuarios. Es el camino de entrada de estas aplicaciones a todo el sistema de comunicación. En este nivel se sitúan las aplicaciones (programas específicos) como acceso a terminales remotos, transferencia de ficheros, servidores de impresión, comandos del sistema operativo de red, sentencias enviar/recibir de algunos lenguajes de programación, etc.

Los niveles en los que se debe tener especial cuidado se sitúan en:

- Nivel 1, nivel físico: selección del cable, realización de conectores, la conexión, verificar y configurar la velocidad de transmisión, etc.

- Nivel 2, la supervisión sobre el enlace de datos y la toma de decisiones sobre los números de participantes, donde haga falta.

- Nivel 7, aplicación: los fabricantes de dispositivos de comunicación industrial, ofrecen varios tipos de software de comunicación con los dispositivos. La visión del usuario establecen la comunicación con el sistema final, y utilizan herramientas que le permiten incorporar nuevos dispositivos, establecer funciones de diagnóstico, la verificación de diferentes parámetros de red de control. Estas opciones son a nivel del software.

Los protocolos que siguen el modelo OSI facilitan una estructura abierta.

## 1.2. Antecedentes de los protocolos

Las primeras redes industriales propietarias fueron desarrolladas a finales de los años 70, entre los controladores, estaba el PLC (Modbus-MODICON). Para 1980 las redes propietarias de PLC, estaban formadas por: Telway-Unitelway (Telemecanique), Data Highway (Allen Bradley), Sinec (Siemens) y Tiway (Texas).
Para 1982 se crea un grupo de trabajo en Francia para obtener un bus industrial único FIP (Factory Instrumentation Protocol). En 1984 se crean las especificaciones del CAN (Controller Area Network) de Bosch. En 1985 se forma el grupo PROFIBUS (Alemania).

Para 1990 se crearon diversos protocolos no compatibles, basados en productos existentes o prototipos como lo fue: MIL1553B, Hart (Rousemount), Bitbus (Intel). Propuestas completas: FIB, PROFIBUS.

### 1.2.1. CAN

Creado a mediados de 1980 con el objetivo de brindar conexión y disminuir los costos de cableado entre dispositivos dentro de automóviles. Se difundió posteriormente a otras áreas, por ejemplo control de plantas industriales, aplicaciones domésticas, control de ascensores, control de sistemas de navegación, etcétera. Sus principales características son:

- Estándar ISO.
- Amplia disponibilidad de dispositivos comerciales.
- Rápido.
- Importante mecanismo de control de errores.
- Velocidad hasta 1 Mbps.
- Protocolo de comunicaciones orientado a los mensajes.
- Arbitraje por prioridad de mensajes (CSMA/AMP).
- Resolución de colisiones.
- Alta probabilidad de detección de errores.
- Capacidad de implementar control en tiempo real.
- Escalabilidad.

### 1.2.2. PROFIBUS

La base de la especificación del estándar PROFIBUS fue un proyecto de investigación (1987-1990) llevado a cabo por los siguientes participantes: ABB, AEG, Bosch, Honeywell, Moeller, Landis Gyr, Phoenix Contact, Rheinmetall, RMP, Sauter-cumulus, Schleicher, Siemens y cinco institutos alemanes de investigación. Hubo además un pequeño apoyo por parte del gobierno alemán.

El resultado de este proyecto fue el primer borrador de la norma DIN 19245, el estándar PROFIBUS, partes 1 y 2. La parte 3, PROFIBUS-DP, se definió en 1993. Recientes estudios de mercado llevados a cabo por empresas ajenas a la organización de usuarios

de PROFIBUS señalan a éste como el bus con más futuro en el campo de los procesos industriales.

Soporta una gran variedad de equipos que van desde PC y PLC hasta robots, pasando por todo tipo de elementos de campo, es decir, casi la mayoría de las aplicaciones industriales.

## 1.3. Los microcontroladores

Actualmente los controladores están presentes en nuestro trabajo, en la casa y en nuestra vida en general. Se pueden encontrar controlando el funcionamiento de los ratones y teclados de las computadoras, en teléfonos, hornos de microondas y televisores. Pero la invasión acaba de comenzar y el nacimiento del siglo XXI será testigo de la conquista de estos diminutos computadores que gobernarán la mayor parte de los aparatos que se fabricarán y usarán.

Generalmente, recibe el nombre de controlador el dispositivo que se emplea para el control de uno o varios procesos. Aunque el concepto no ha cambiado a través del tiempo, su implementación física se ha ido modificando frecuentemente. Hace tres décadas, los controladores se construían exclusivamente con componentes de lógica discreta, posteriormente se emplearon los microprocesadores, que se rodeaban con chips de memoria y Entrada/Salida sobre una tarjeta de circuito impreso. En la actualidad, todos los elementos del controlador se han podido incluir en un chip, el cual recibe el nombre de microcontrolador, que realmente consiste en un sencillo pero completo computador contenido en el corazón (chip) de un circuito integrado.

Cada vez existen más productos que incorporan un microcontrolador con el fin de aumentar sustancialmente sus prestaciones, reducir su tamaño y costo, mejorar su fiabilidad y disminuir el consumo. Algunos fabricantes de microcontroladores superan el millón de unidades de un modelo determinado producidas en una semana.

En la práctica cada fabricante de microcontroladores oferta un elevado número de modelos diferentes, desde los más sencillos hasta los más poderosos y es posible seleccionarlos por sus características. Por ello, un aspecto muy destacado en el diseño es la selección del microcontrolador a utilizar.

En cuanto a las técnicas de fabricación, cabe decir que prácticamente la totalidad de los microcontroladores actuales se fabrican con tecnología CMOS debido a su alta inmunidad al ruido.

Al escoger el microcontrolador en un diseño concreto hay que tener en cuenta una multitud de factores, algunos de estos pueden ser:

- Costos.
- Aplicación.
- Procesamiento de datos.
- Cantidad de líneas de entrada/salida.
- Consumo.
- Memoria.

Por lo cual existen diferentes modelos de microcontroladores que ofrecen éstas características fabricados por distintas compañías, dentro de las más populares son sin duda:

- Microcontroladores fabricados por Intel.
  Por ejemplo el 8048, tiene un precio, disponibilidad y herramientas de desarrollo que hacen que todavía sea muy popular. El 8051 es el microcontrolador más popular de Intel, fácil de programar y potente. Está bien documentado y posee cientos de variantes e incontables herramientas de desarrollo.
- Microcontroladores fabricados por Motorola y Toshiba.
  El 68HC11 es un microcontrolador de 8 bits potente y popular con gran cantidad de variantes: 683xx, surgido a partir de la popular familia 68k, a la que se incorporan algunos periféricos. Son microcontroladores de altísimas prestaciones.
- Microcontroladores fabricados por Microchip (PIC).
  Esta familia de microcontroladores que gana popularidad día a día, fueron los primeros microcontroladores con arquitectura RISC. Es preciso resaltar en este punto que existen innumerables familias de microcontroladores, cada una de las cuales posee un gran número de variantes.

### 1.3.1. Surgimiento de la empresa Microchip y del PIC

En 1965, la empresa GI creó una división de microelectrónica, GI Microelectronics División, que comenzó fabricando memorias EPROM y EEPROM, que conformaban las familias AY3-XXXX y AY5-XXXX.

A principios de los años 70 diseñó el microprocesador de 16 bits CP1600, razonablemente bueno pero que no manejaba eficazmente las entradas y salidas. Para solventar este problema, en 1975 diseñó un chip destinado a controlar Entrada/Salida: el PIC (Peripheral Interface Controller). Se trataba de un controlador rápido pero limitado y con pocas instrucciones, se diseño para trabajar en combinación con el CP1600.

La arquitectura del PIC, que se comercializó en 1975, era sustancialmente la misma que la de los actuales modelos PIC16C5X. En aquel momento se fabricaba con tecnología NMOS y el producto sólo se ofrecía con memoria ROM y con un pequeño pero robusto microcódigo.

La década de los 80 no fue buena para GI, que tuvo que reestructurar sus negocios, concentrando sus actividades en los semiconductores de potencia. La GI Microelectronics Division se convirtió en una empresa subsidiaria, llamada GI Microelectronics Inc.

Finalmente, en 1985, la empresa fue vendida a un grupo de inversores de capital de riesgo, los cuales, tras analizar la situación, rebautizaron a la empresa con el nombre de Arizona Microchip Technology y orientaron su negocio a los PIC, las memorias EPROM paralelo y las EEPROM serie. Se comenzó rediseñando los PIC, que pasaron a fabricarse con tecnología CMOS, surgiendo la familia de gama baja PIC16CSX, considerada como la clásica.

Una de las razones del éxito de los PIC se basa en su utilización. Cuando se aprende a manejar uno de ellos, conociendo su arquitectura y su repertorio de instrucciones, es muy fácil emplear otro modelo.

Microchip cuenta con su fabrica principal en Chandler, Arizona, en donde se fabrican y prueban los chips con los más avanzados recursos técnicos. En 1993 construyó otra fabrica de similares características en Tampa, Arizona. También cuenta con centros de ensamblaje y ensayos en Taiwan y Tailandia. Para tener una idea de su alta producción, hay que tener en cuenta que ha superado el millón de unidades por semana en productos CMOS de la familia PIC16CSX (Cf. [Campos, 1998]).

### 1.3.2. Arquitectura básica de un microcontrolador

Aunque inicialmente todos los microcontroladores adoptaron la arquitectura clásica de Von Neumann, en el momento presente se impone la arquitectura Harvard. La arquitectura de Von Neumann se caracteriza por disponer de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control).

La arquitectura Harvard dispone de dos memorias independientes una, que contiene sólo instrucciones y otra, sólo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias (Cf. [Angulo, 2000]).
Los microcontroladores PIC responden a la arquitectura Harvard (ver Figura 1.1).

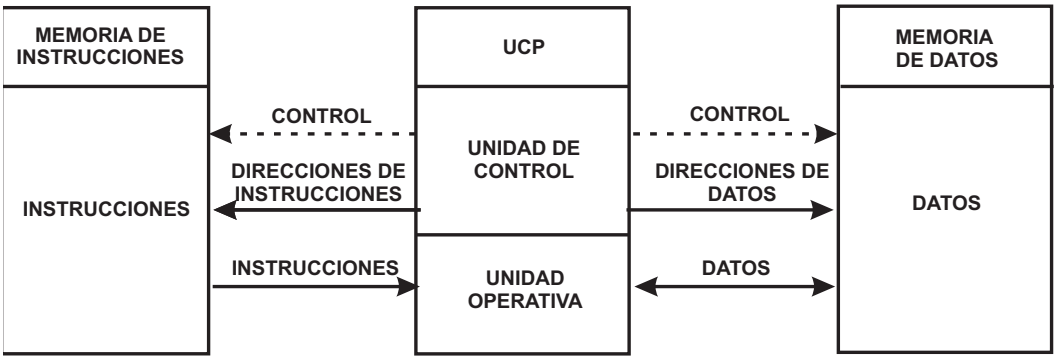



Figura 1.1: Arquitectura Harvard.
*Fuente: [Campos, 1998].*

### 1.3.3. El procesador

Es el elemento más importante del microcontrolador y determina sus principales características, tanto a nivel hardware como software.
Se encarga de direccionar la memoria de instrucciones, recibir el código OP de la instrucción en curso, su decodificación y la ejecución de la operación que implica la instrucción, así como la búsqueda de los operandos y el almacenamiento del resultado.
Existen tres orientaciones en cuanto a la arquitectura y funcionalidad de los procesadores actuales.

1. **CISC:** Un gran número de procesadores usados en los microcontroladores están basados en la filosofía CISC (Computadores de Conjunto de Instrucciones Complejo). Disponen de más de 80 instrucciones máquina en su repertorio, algunas de las cuales son muy sofisticadas y potentes, requiriendo muchos ciclos para su ejecución. Una ventaja de los procesadores CISC es que ofrecen al programador instrucciones complejas que actúan como macros.

2. **RISC:** Tanto la industria de los computadores comerciales como la de los microcontroladores están cambiando hacia la filosofía RISC (Computadores de Conjunto de Instrucciones Reducido). En estos procesadores el repertorio de instrucciones máquina es muy reducido y las instrucciones son simples y, generalmente, se ejecutan en un ciclo. La sencillez y rapidez de las instrucciones permiten optimizar el hardware y el software del procesador.

3. **SISC:** Son los microcontroladores destinados a aplicaciones muy concretas, el conjunto de instrucciones, además de ser reducido, es específico, o sea, las instrucciones se adaptan a las necesidades de la aplicación prevista. Esta filosofía se

ha bautizado con el nombre de SISC (Computadores de Conjunto de Instrucciones Específico).

### 1.3.4. Memoria

En los microcontroladores la memoria de instrucciones y datos está integrada en el propio chip. Una parte debe ser no volátil, tipo ROM, y se destina a contener el programa de instrucciones que gobierna la aplicación. Otra parte de memoria será tipo RAM, volátil, y se destina a guardar las variables y los datos.

Hay dos peculiaridades que distinguen a los microcontroladores de los computadores personales:

1. No existen sistemas de almacenamiento masivo como disco duro o disquetes.

2. Como el microcontrolador sólo se destina a una tarea en la memoria ROM, sólo hay que almacenar un único programa de trabajo.

La RAM en estos dispositivos es de poca capacidad pues sólo debe contener las variables y los cambios de información que se produzcan en el transcurso del programa. Por otra parte, como sólo existe un programa activo, no se requiere guardar una copia del mismo en la RAM pues se ejecuta directamente desde la ROM.
Los usuarios de computadores personales están habituados a manejar Megabytes de memoria, pero, los diseñadores con microcontroladores trabajan con capacidades de ROM comprendidas entre 512 bytes y 8 k bytes y de RAM comprendidas entre 20 y 512 bytes.

Según el tipo de memoria ROM que dispongan los microcontroladores, la aplicación y utilización de los mismos es diferente. Se describen las cinco versiones de memoria no volátil que se pueden encontrar en los microcontroladores del mercado (Cf. [Campos, 1998]).

1. **ROM con máscara:** Es una memoria no volátil de sólo lectura cuyo contenido se graba durante la fabricación del chip. El elevado costo del diseño de la máscara sólo hace aconsejable el empleo de los microcontroladores con este tipo de memoria cuando se precisan cantidades superiores a varios miles de unidades.

2. **OTP:** El microcontrolador contiene una memoria no volátil de sólo lectura *programable una sola vez* por el usuario. OTP (One Time Programmable). Es el usuario quien puede escribir el programa en el chip mediante un sencillo grabador controlado por un programa desde un PC.

La versión OTP es recomendable cuando es muy corto el ciclo de diseño del producto, o bien, en la construcción de prototipos y series muy pequeñas.

Tanto en este tipo de memoria como en la EPROM, se suele usar la encriptación mediante fusibles para proteger el código contenido.

3. **EPROM:** Los microcontroladores que disponen de memoria EPROM (Erasable Programmable Read Only Memory) pueden borrarse y grabarse muchas veces. La grabación se realiza, como en el caso de los OTP, con un grabador gobernado desde un PC. Sí, posteriormente, se desea borrar el contenido, disponen de una ventana de cristal en su superficie por la que se somete a la EPROM a rayos ultravioleta durante varios minutos. Las cápsulas son de material cerámico y son más caros que los microcontroladores con memoria OTP que están hechos con material plástico.

4. **EEPROM:** Se trata de memorias de sólo lectura, programables y borrables eléctricamente EEPROM (Electrical Erasable Programmable Read Only Memory). Tanto la programación como el borrado, se realizan eléctricamente desde el propio grabador y bajo el control programado de un PC. Es muy cómoda y rápida la operación de grabado y la de borrado. No disponen de ventana de cristal en la superficie.

   Los microcontroladores dotados de memoria EEPROM una vez instalados en el circuito, pueden grabarse y borrarse cuantas veces se quiera sin ser retirados de dicho circuito. Para ello se usan "grabadores en circuito"que confieren una gran flexibilidad y rapidez a la hora de realizar modificaciones en el programa de trabajo.

   El número de veces que puede grabarse y borrarse una memoria EEPROM es finito, por lo que no es recomendable una reprogramación continua. Son muy idóneos para la enseñanza y la ingeniería de diseño.

   Cada vez más, se va extendiendo en los fabricantes la tendencia de incluir una pequeña zona de memoria EEPROM en los circuitos programables, para guardar y modificar cómodamente una serie de parámetros, que adecúan el dispositivo a las condiciones del entorno.

   Este tipo de memoria es relativamente lenta.

5. **FLASH:** Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar. Funciona como una ROM y una RAM pero consume menos energía y es más pequeña.

   A diferencia de la ROM, la memoria FLASH es programable en el circuito. Es más rápida y de mayor densidad que la EEPROM.

La alternativa FLASH está recomendada frente a la EEPROM cuando se precisa gran cantidad de memoria de programa no volátil. Es más veloz y tolera más ciclos de escritura/borrado.

Las memorias EEPROM y FLASH son muy útiles al permitir que los microcontroladores que las incorporan puedan ser reprogramados *en circuito*, es decir, sin tener que sacar el circuito integrado de la tarjeta. Así, un dispositivo con este tipo de memoria incorporado al control del motor de un automóvil permite que pueda modificarse el programa durante la rutina de mantenimiento periódico, compensando los desgastes y otros factores tales como la compresión, la instalación de nuevas piezas, etc. La reprogramación del microcontrolador puede convertirse en una labor rutinaria dentro de la puesta a punto.

### 1.3.5. Stack

En los microcontroladores PIC, el stack es una memoria interna dedicada, de tamaño limitado, separada de las memorias de datos y del programa, la cual es inaccesible al programador, y organizada en forma de pila, que es utilizada solamente, y en forma automática, para guardar las direcciones de retorno de subrutinas e interrupciones.

Cada posición es de 11 bits y permite guardar una copia completa del PC. Como en toda memoria tipo pila, los datos son accedidos de manera tal que el primero que entra es el último que sale.

En los 16C5X el stack es de sólo dos posiciones, mientras que en los 16CXX es de 8 posiciones y en los 17CXX es de 16 posiciones. Esto representa, en cierta medida, una limitación de estos microcontroladores, ya que no permite hacer uso intensivo del anidamiento de subrutinas.

En los 16C5X, solo se pueden anidar dos niveles de subrutinas, es decir que una subrutina que es llamada desde el programa principal, puede a su vez llamar a otra subrutina, pero esta última no puede llamar a una tercera, porque se desborda la capacidad del stack, que sólo puede almacenar dos direcciones de retorno.

Esto de hecho representa una traba para el programador y además parece impedir o dificultar la programación estructurada, sin embargo es una buena solución de compromiso ya que estos microcontroladores están diseñados para aplicaciones de alta velocidad en tiempo real, en las que el overhead (demoras adicionales) que ocasiona un excesivo anidamiento de subrutinas es inaceptable.

Por otra parte existen técnicas de organización del programa que permiten mantener la claridad de la programación estructurada, sin necesidad de utilizar tantas subrutinas

anidadas.

El stack y el puntero interno que lo direcciona, son invisibles para el programador, sólo accede automáticamente para guardar o rescatar las direcciones de programa cuando se ejecutan las instrucciones de llamada o retorno de subrutinas, o cuando se produce una interrupción o se ejecuta una instrucción de retorno de ella.

### 1.3.6. Reloj principal

Todos los microcontroladores disponen de un circuito oscilador que genera una onda cuadrada de alta frecuencia, que configura los impulsos de reloj usados en la sincronización de todas las operaciones del sistema.

Generalmente, el circuito de reloj está incorporado en el microcontrolador y sólo se necesitan unos pocos componentes exteriores para seleccionar y estabilizar la frecuencia de trabajo. Dichos componentes suelen consistir en un cristal de cuarzo junto a elementos pasivos o bien un resonador cerámico o una red RC (circuito formado por una resistencia y un capacitor).

Aumentar la frecuencia de reloj supone disminuir el tiempo en que se ejecutan las instrucciones pero lleva un incremento en el consumo de energía.

### 1.3.7. Recursos específicos

Cada fabricante oferta numerosas versiones de una arquitectura básica de microcontrolador. En algunas amplía las capacidades de las memorias, en otras incorpora nuevos recursos, en otras reduce las prestaciones al mínimo para aplicaciones muy simples, etc. La labor del diseñador es encontrar el modelo mínimo que satisfaga todos los requerimientos de su aplicación. De esta forma, minimizará el costo, el hardware y el software.

Los principales recursos específicos que incorporan los microcontroladores son:

- Temporizadores o "Timers": Se emplean para controlar periodos de tiempo (temporizadores) y para llevar la cuenta de acontecimientos que suceden en el exterior (contadores).

  Para la medida de tiempos se carga un registro con el valor adecuado y a continuación dicho valor se va incrementando o decrementando al ritmo de los impulsos de reloj o algún múltiplo hasta que se desborde y llegue a 0, momento en el que se produce un aviso.

  Cuando se desean contar acontecimientos que se materializan por cambios de nivel en alguna de los pines del microcontrolador, el mencionado registro se va incre-

mentando o decrementando al ritmo de dichos impulsos.

- Perro guardián o "Watchdog": Cuando el computador personal se bloquea por un fallo del software u otra causa, se pulsa el botón del reset y se reinicializa el sistema. Pero un microcontrolador funciona sin el control de un supervisor y de forma continua las 24 horas del día. El perro guardián consiste en un temporizador que, cuando se desborda y pasa por 0, provoca un reset automáticamente en el sistema. Se debe diseñar el programa de trabajo que controla la tarea de forma que refresque o inicialice al perro guardián antes de que provoque el reset. Si falla el programa o se bloquea, no se refrescará al perro guardián y, al completar su temporización, *ladrará y ladrará* hasta provocar el reset.

- Protección ante fallo de alimentación o "Brownout": Se trata de un circuito que reinicializa al microcontrolador cuando el voltaje de alimentación (VDD) es inferior a un voltaje mínimo (brownout). Mientras el voltaje de alimentación sea inferior al de brownout el dispositivo se mantiene inactivo, comenzando a funcionar normalmente cuando sobrepasa dicho valor.

- Estado de reposo o de bajo consumo: Son abundantes las situaciones reales de trabajo en que el microcontrolador debe esperar, sin hacer nada, hasta que se produzca algún acontecimiento externo que le ponga de nuevo en funcionamiento. Para ahorrar energía, (factor clave en los aparatos portátiles), los microcontroladores disponen de una instrucción especial (SLEEP en los PIC), que les pasa al estado de reposo o de bajo consumo, en el cual los requerimientos de potencia son mínimos. En dicho estado se detiene el reloj principal y se congelan sus circuitos asociados. Al activarse una interrupción ocasionada por el acontecimiento esperado, el microcontrolador se despierta y reanuda su trabajo.

- Convertidor Analógico/Digital: Los microcontroladores que incorporan un Convertidor Analógico/Digital pueden procesar señales analógicas. Suelen disponer de un multiplexor que permite aplicar a la entrada del Convertidor Analógico/-Digital diversas señales analógicas desde los pines del circuito integrado.

- Convertidor Digital/Aanlógico: Transforma los datos digitales obtenidos del procesamiento del computador en su correspondiente señal analógica que envía al exterior por uno de los pines del microcontrolador.

- Comparador analógico: Algunos modelos de microcontroladores disponen internamente de un amplificador operacional que actúa como comparador entre una

señal fija de referencia y otra variable que se aplica por uno de los pines de la cápsula. La salida del comparador proporciona un nivel lógico 1 ó 0 según una señal sea mayor o menor que la otra.

También hay modelos de microcontroladores con un módulo de tensión de referencia que proporciona diversas tensiones de referencia que se pueden aplicar en los comparadores.

- Modulador de ancho de pulso o PWM: Son circuitos que proporcionan en su salida impulsos de variable, que se ofrecen al exterior a través de los pines del microcontrolador.

- Puertas de Entrada/Salida digitales: Todos los microcontroladores destinan algunas de sus pines a soportar líneas de Entrada/Salida digitales. Por lo general, estas líneas se agrupan de ocho en ocho formando Puertas.

  Las líneas digitales de las puertas pueden configurarse como entrada o como salida cargando un 1 ó un 0 en el bit correspondiente de un registro destinado a su configuración.

- Puertas de comunicación: Con objeto de dotar al microcontrolador de la posibilidad de comunicarse con otros dispositivos externos, otros buses de microprocesadores, buses de sistemas, buses de redes y poder adaptarlos con otros elementos bajo otras normas y protocolos. Algunos modelos disponen de recursos que permiten directamente esta tarea, entre los que destacan:

  1. UART, adaptador de comunicación serie asíncrona.
  2. USART, adaptador de comunicación serie síncrona y asíncrona.
  3. USB (Universal Serial Bus), que es un moderno bus serie para las computadoras.
  4. Bus I2C, que es un interfaz serie de dos hilos desarrollado por la empresa Philips.
  5. CAN (Controller Area Network), para permitir la adaptación con redes, desarrollado conjuntamente por Bosch e Intel para el cableado de dispositivos en automóviles, principalmente.

### 1.3.8. Características relevantes del PIC

Dentro de las características más representativas del PIC se encuentran:

**Segmentación**

Se aplica la técnica de segmentación (pipe-line) en la ejecución de las instrucciones.

La segmentación permite al procesador realizar al mismo tiempo la ejecución de una instrucción y la búsqueda del código de la siguiente. De esta forma se puede ejecutar cada instrucción en un ciclo (un ciclo de instrucción equivale a cuatro ciclos de reloj).

Las instrucciones de salto ocupan dos ciclos al no conocer la dirección de la siguiente instrucción hasta que no se haya completado.

**Formato de las instrucciones**

El formato de todas las instrucciones es de la misma longitud

Todas las instrucciones de los microcontroladores de la gama baja tienen una longitud de 12 bits. Las de la gama media tienen 14 bits. Esta característica es muy ventajosa en la optimización de la memoria de instrucciones y facilita enormemente la construcción de ensambladores y compiladores.

**Conjunto de instrucciones**

Procesador RISC (Computador de Conjunto de Instrucciones Reducido).

Los modelos de la gama baja disponen de un repertorio de 33 instrucciones, 35 los de la gama media y casi 60 los de la alta.

**Arquitectura basada en un banco de registros**

Esto significa que todos los objetos del sistema (puertas de E/S, temporizadores, posiciones de memoria, etc.) están implementados físicamente como registros.

La arquitectura Harvard y la técnica de segmentación son los principales recursos en los que se apoya el elevado rendimiento que caracteriza estos dispositivos programables, mejorando dos características esenciales:

1. Velocidad de ejecución.

2. Eficiencia en la compactación del código.

## 1.4. Herramientas para trabajar con el PIC

Uno de los factores que más importancia tiene a la hora de seleccionar un microcontrolador entre todos los demás, es el soporte tanto software como hardware de que dispone. Un buen conjunto de herramientas de desarrollo puede ser decisivo en la elección, ya que pueden suponer una ayuda inestimable en el desarrollo del proyecto. Las principales herramientas de ayuda al desarrollo de sistemas basados en microcontroladores son:

- **Ensamblador.** La programación en lenguaje ensamblador puede resultar un tanto ardua para el principiante, pero permite desarrollar programas muy eficientes, ya que otorga al programador el dominio absoluto del sistema. Los fabricantes suelen proporcionar el programa ensamblador de forma gratuita y en cualquier caso siempre se puede encontrar una versión gratuita para los microcontroladores más populares.
- **Compilador.** La programación en un lenguaje de alto nivel (como el lenguaje C) permite disminuir el tiempo de desarrollo de un producto. No obstante, si no se programa con cuidado, el código resultante puede ser mucho más ineficiente que el programado en ensamblador. Las versiones más potentes suelen ser muy caras, aunque para los microcontroladores más populares pueden encontrarse versiones demo limitadas e incluso compiladores gratuitos.
- **Depuración.** Debido a que los microcontroladores van a controlar dispositivos físicos, los desarrolladores necesitan herramientas que les permitan comprobar el buen funcionamiento del microcontrolador cuando es conectado al resto de circuitos.
- **Simulador.** Son capaces de ejecutar en una PC programas realizados para el microcontrolador. Los simuladores permiten tener un control absoluto sobre la ejecución de un programa, siendo ideales para la depuración de los mismos. Su gran inconveniente es que es difícil simular la entrada y salida de datos del microcontrolador. Tampoco cuentan con los posibles ruidos en las entradas, pero, al menos, permiten el paso físico de la implementación de un modo más seguro y menos costoso, puesto que se ahorra en grabaciones de chips (Cf. [Angulo, 1999]).

### 1.4.1. MPLAB IDE

La disponibilidad de herramientas eficaces y económicas justifica la espectacular aceptación de los PIC. Por eso Microchip se ha esforzado en acompañar a los PIC con el complemento adecuado para desarrollar las aplicaciones de forma sencilla y cómoda. Para ello, trabaja en su conocido entorno MPLAB Integrated Development Environment (IDE).

MPLAB IDE es un programa de software que funciona en una PC. Desarrollado para aplicaciones de los microcontroladores Microchip. Este es llamado un Entorno de Desarrollo Integrado, o IDE, porque proporciona un sólo ambiente integrado de desarrollo de código para el microcontrolador.

Un sistema de desarrollo integrado (IDE) para controladores, es un sistema de programas que funciona en una PC, ayudando a escribir, editar, depurar y crear el código del programa en un microcontrolador (Cf. [Microchip, 2005]).

MPLAB IDE, funciona en una PC y contiene todos los componentes necesarios para diseñar un proyecto con aplicaciones de sistemas empotrados.

La principal tarea para desarrollar una aplicación de controladores incluidos son:

1. Crear un alto nivel de diseño. Con los rasgos y el desempeño deseado, decidir cuál dispositivo PICmicro o dsPIC es el mejor para la aplicación, el diseño asociado con la circuitería del hardware. Después de determinar que periféricos y pines controlarán el hardware, escribir el programa, el software que controlará los aspectos de la aplicación incluida. Un lenguaje de herramientas como el ensamblador, el cual es directamente traducido a código máquina, o a un compilador que permita un lenguaje más natural para crear programas que puedan ser usados para escribir y editar código. El ensamblador y el compilador ayudan a hacer al código entendible, permitiendo nombres a las funciones para identificar los códigos de rutinas con variables que tienen nombres asociados con su uso, y con la construcción de la ayuda, organiza el código en una estructura mantenible.

2. Compilar, ensamblar y pasar el software usando el ensamblador y/o compilador y el enlace para convertir el código en (1 y 0, código máquina). Este código máquina eventualmente empezará la firmware ( el código programado adentro del microcontrolador).

3. Probar el código. Usualmente un programa complejo no trabaja exactamente a lo imaginado, y los errores necesitan ser removidos del diseño para obtener los resultados esperados. El depurador permite ver la serie de 1 y 0 ejecutándose, relacionando al código fuente que se escribió, con los símbolos y nombres de funciones del programa. La depuración permite experimentar con el código, para ver la evaluación de las variables y varios puntos en el programa, y hacer varias pruebas, cambiando los valores de la variable y pasando a través de las rutinas.

4. Programar el código en el microcontrolador y verificar que éste ejecute correctamente en la aplicación terminada.

**Lenguaje de herramientas**

Los lenguajes de herramientas son programas como los ensambladores y compiladores. La mayoría de los programadores está familiarizado con herramientas de lenguajes como compiladores de Visual Basic o C. Las herramientas de lenguaje producen una depuración de archivos que MPLAB IDE usa para relacionar las instrucciones máquina y las locaciones de memoria con un código fuente. Este bit de integración permite que el editor de MPLAB ponga puntos de ruptura, permitiendo ver el contenido de las variables en la ventana, permite ir paso a paso a través del código fuente, viendo ejecutar la aplicación.

**Depuración designada**

En el desarrollo del ambiente, la ejecución del código es probada en un depurador. El depurador puede ser un programa de software que simule la operación del microcontrolador para probarlo, o puede ser un instrumento especial para analizar el programa como si se ejecutara en la aplicación.
Los simuladores son construidos dentro de MPLAB IDE, por lo tanto un programa puede ser probado sin ningún hardware adicional. Un simulador es un software depurador, las funciones del depurador para el simulador siempre idénticas al depurador de hardware, permitiendo una nueva herramienta que será aprendida fácilmente. Usualmente un simulador funciona un poco más lento que un microcontrolador real.

### 1.4.2. Compilador de C

HI-TECH PICC es un compilador de C de alto rendimiento para los PIC de Microchip de las series de microcontroladores 10/12/14/16/17. HI-TECH PICC es una fuerte industria del compilador ANSI C, no es una implementación como algunos otros compiladores para PIC. El compilador implementa de lleno el ISO/ANSI C, con excepción de la repetición. Todos los tipos de datos son soportados, incluyendo los de 24 y 32 bits de acuerdo al estándar IEEE de punto flotante. HI-TECH PICC usa todas las características específicas del PIC y con una optimización inteligente, puede generar una alta calidad de código que facilita más su uso, sobre el lenguaje ensamblador. La dirección automática de la selección de la página y del banco libera el programador de los detalles triviales del código ensamblador.

El compilador de PICC funciona con todos los microcontroladores de serie del microchip PIC10xx, de PIC12xx, de PIC14000, de PIC16xx y de PIC17xx. El software de HI-TECH está desarrollando continuamente los compiladores de C de alta calidad para funcionar con todos los microcontroladores actuales y futuros del microchip (Cf. [PICC, 2005]).

El compilador HI-TECH PICC-Lite es una versión libre del compilador PICC disponible para Windows, Linux, y Mac OS X. El compilador PICC-Lite es igual en cada respecto que el compilador completo HI-TECH PICC, excepto que tiene ayuda para solamente un subconjunto limitado de procesadores, de ahí que tenga algunas limitaciones en la cantidad de memoria que puede ser utilizada y el código fuente para los estándares de la biblioteca no se proporciona. Sus principales características son:

1. ANSI C. - Es completo y portable.
2. Confiable.
3. Niveles múltiples de optimización en C.
4. Un ensamblador óptimo.
5. Permite hacer enlaces completos, con la sobre-posición de variables locales para reducir el uso mínimo de RAM.
6. Biblioteca comprensiva de C con todo el código de fuente proporcionado.
7. Incluye soporte para 24 bits y 32 bits de punto flotante y 32 bits del tipo de datos largos.
8. Programación mezclada de C y ensamblador.
9. Número ilimitado de los archivos fuente.
10. Listados que demuestran el ensamblador generado.
11. Compatible.- Integra en el MPLAB IDE, MPLAB ICD y la mayoría de las herramientas de desarrollo 3rd-party.

En el Anexo B se incluye algunas divergencias que maneja PICC-LITE con relación al estándar ANSI C.

# Capítulo 2

# Sistema mínimo

### Introducción

En el desarrollo de un protocolo de comunicación, la simplificación de elementos y su organización es necesaria para facilitar el acceso a los diferentes medios que interactúan con ellos.

El sistema mínimo propuesto en este proyecto de tesis, cumple con estas facilidades de acceso físico a componentes y dispositivos electrónicos tales como pantallas LCD, interfaz de comunicación, entradas/salidas analógicas y digitales, entre otros. Estos accesos están basados y orientados al uso del microcontrolador PIC16F877 fabricado por Microchip.

Existen sistemas mínimos desarrollados o fabricados por diferentes industrias tales como Microchip o Motorola, pero su principal desventaja es que son de alto costo y de arquitectura cerrada, lo que significa que no se le pueden hacer modificaciones o cambios en la misma.

Sin embargo, el sistema mínimo propuesto no solamente es de arquitectura abierta, sino además es de bajo costo, ya que cuenta con todos los dispositivos necesarios para la implementación de protocolos de comunicación.

En el presente Capítulo se da un panorama del sistema mínimo propuesto detallando cada una de las partes que lo componen, así como sus dispositivos interconectados.

## 2.1. Sistemas mínimos

Un sistema mínimo es un dispositivo que ayuda a una simplificación de elementos que interactúan en un proceso, en este caso, el sistema mínimo que se propone está orientado al microcontrolador PIC16F877.

El sistema mínimo está diseñado para poder conectar diferentes dispositivos (una pantalla de LCD, teclado matricial, interfaces para la comunicación con otros dispositivos, entrada de señales analógicas), algunos dispositivos se ilustran en la Figura 2.1.

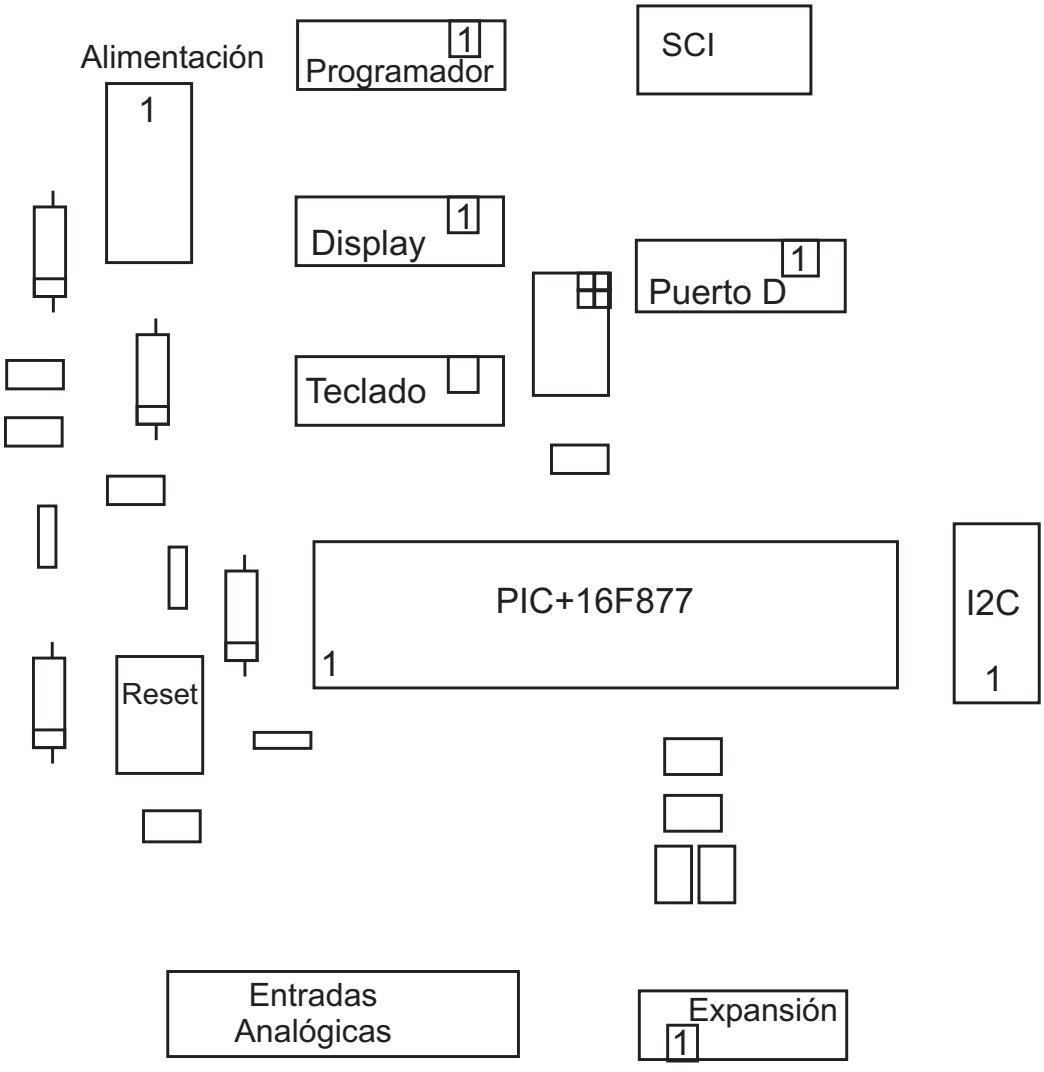



Figura 2.1: Diagrama del sistema mínimo orientado al PIC16F877.

En la Figura 2.1 se puede observar que el sistema mínimo tiene acceso disponible a diferentes dispositivos como pantalla LCD, teclado matricial, comunicación serial, entradas analógicas y digitales, los cuales son activados o desactivados mediante la programación del microcontrolador. Se puede observar físicamente en la Figura 2.2, el sistema mínimo con sus entradas y salidas analógicas (1) y digitales (2), la ubicación del microcontrolador PIC16F877 (3) y los convertidores RS-232 a RS-485 (4).

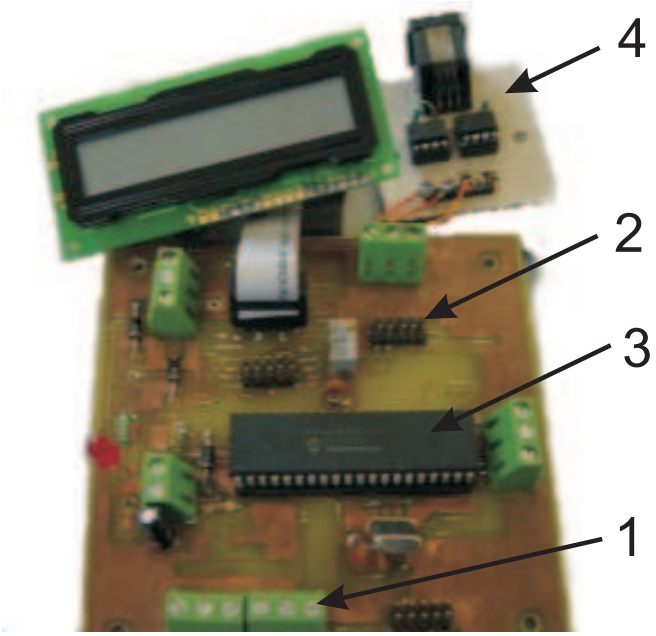



Figura 2.2: Sistema Mínimo propuesto.

## 2.2. Componentes del sistema mínimo

El sistema mínimo que se propone en el presente trabajo de tesis, es el que se muestra en la Figura 2.1, que contiene varios componentes o dispositivos útiles en la puesta en marcha de este protocolo. Algunos de los dispositivos que se implementaron son:

- Pantalla de LCD.
- Microcontrolador PIC16F877.
- Interfaz de comunicación RS-232.
- Convertidor de interfaz RS-232 a RS-485.
- Entradas y salidas analógicas y digitales.

Todos estos dispositivos tienen una función específica y para cada uno de ellos existe una parte exclusiva para su uso y aplicación. Más adelante veremos cómo y cuál es la función de cada uno de ellos dentro de éste proyecto de tesis.

### 2.2.1. Diferencia entre un microprocesador y un microcontrolador

El microprocesador es un circuito integrado que contiene la Unidad Central de Proceso (UCP), también llamada procesador de un computador. La UCP está formada por la Unidad de Control, que interpreta las instrucciones, y el Bus de datos que las ejecuta. Esta contiene líneas de buses de direcciones, datos y control para permitir conectarse con la memoria y los módulos de E/S. Esta unidad sirve para configurar al computador implementado por varios circuitos integrados, además se dice que es un sistema abierto porque su configuración puede variar o modificarse de acuerdo a la aplicación a la que esté destinada, como se observa en la Figura 2.3 (Cf. [Angulo, 1999]).

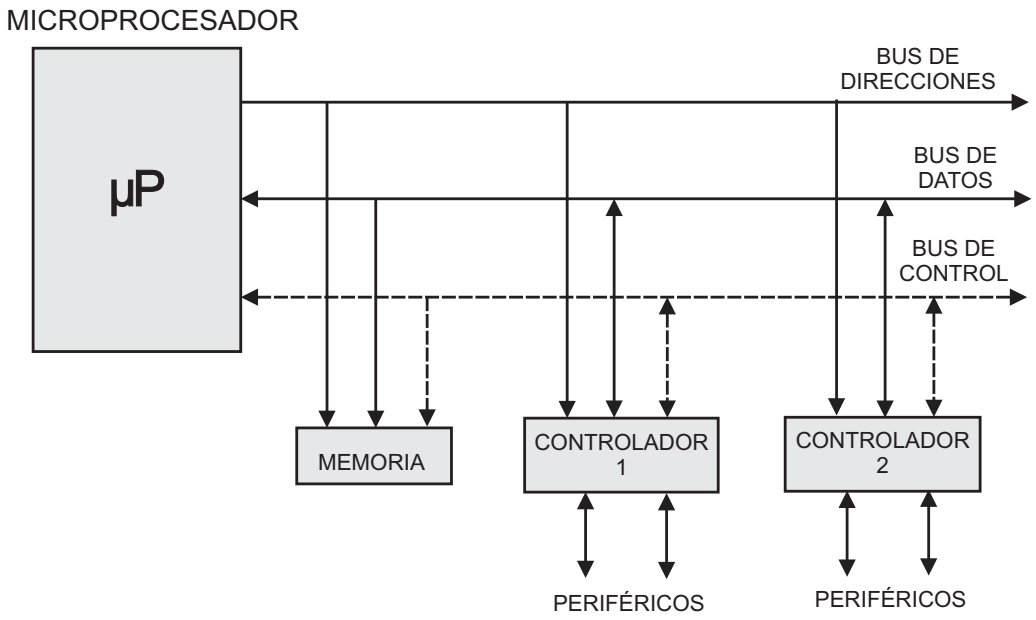



Figura 2.3: Estructura de un microprocesador.
*Fuente: [Angulo, 1999].*

### 2.2.2. Microcontrolador PIC16F877

Un microcontrolador es un circuito integrado programable que contiene todos los componentes de un computador. Se emplea para controlar el funcionamiento de una tarea determinada y, debido a su reducido tamaño, suele ir incorporado en el propio dispositivo al que gobierna, y es por tanto un computador dedicado. En su memoria sólo reside un programa destinado a gobernar una aplicación determinada; sus líneas de entrada/salida soportan la conexión de los sensores y actuadores del dispositivo a controlar, y todos los recursos complementarios disponibles tienen como una finalidad atender sus requerimientos. Una vez programado y configurado el microcontrolador solamente sirve para gobernar la tarea asignada.

El número de productos que funcionan en base a uno o varios microcontroladores aumentan de forma exponencial. La industria informática acapara gran parte de los microcontroladores que se fabrican. Casi todos los periféricos del computador, desde el ratón o el teclado hasta la impresora, son regulados por el programa de un microcontrolador.

Existen un centenar de empresas fabricantes de microcontroladores, pero lo cierto es que en la primera década del siglo XXI, los PIC fabricados por Microchip ocupan un lugar importante, compitiendo con los microcontroladores de Intel y Motorola. Dentro de los diferentes tipos de microcontroladores existen los que procesan datos de 4, 8, 16 y 32 bits, sin embargo, el más representativo y popular es el de 8 bits, al que pertenecen los PIC's (Cf. [Angulo, 1999]). La Figura 2.4 muestra el diagrama del microcontrolador PIC16F877.

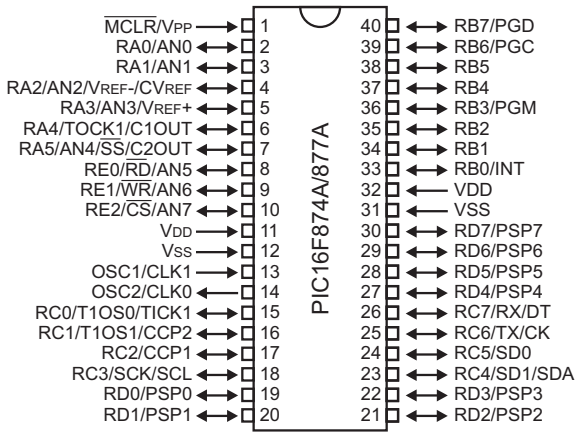



Figura 2.4: Diagrama del microcontrolador PIC16F877.
*Fuente: [Microchip, 2001].*

Microchip ofrece cuatro gamas de microcontroladores de 8 bits para adaptarse a las necesidades de las aplicaciones (*Ibidem.*). Estas cuatro gamas son las siguientes:

1. Gama baja o básica con instrucciones de 12 bits.

2. Gama media con instrucciones de 14 bits.

3. Gama alta con instrucciones de 16 bits.

4. Gama mejorada con instrucciones de 16 bits.

En el presente trabajo de tesis se utiliza el microcontrolador PIC16F877 de la subfamilia PIC16F87X, la cual trata de un conjunto de cuatro modelos de PIC con memoria FLASH, que se ubica dentro de la gama media de microcontroladores de 8 bits.

### Características del microcontrolador PIC16F877

Un microcontrolador posee todos los componentes de un computador, pero con características fijas que no pueden alterarse. El alto rendimiento y elevada velocidad que alcanzan los modernos procesadores, como el que poseen los microcontroladores PIC, se debe a la conjunción de tres técnicas la arquitectura Harvard, el computador tipo RISC y su segmentación(*Ibidem.*). En la Figura 2.5 se puede observar la arquitectura aplicada por Microchip en sus microcontroladores, que se caracteriza por la independencia entre la memoria de código y la de datos (Cf. [Angulo, 2000]).

### Organización de la memoria

#### Memoria de programa FLASH

El microcontrolador está diseñado para que en su memoria de programa se almacenen todas las instrucciones del programa de control. En estos microcontroladores no hay posibilidad de utilizar memorias externas de almacenamiento, por lo que el programa a ejecutarse debe estar grabado permanentemente. El microcontrolador PIC16F877 cuenta con memoria FLASH que es una memoria no volátil en la que se graba el programa de aplicación, y tiene una capacidad de 8K palabras de 14 bits cada una, esta memoria está divida en 4 páginas de 2K palabras (Cf. [Angulo, 2000]).

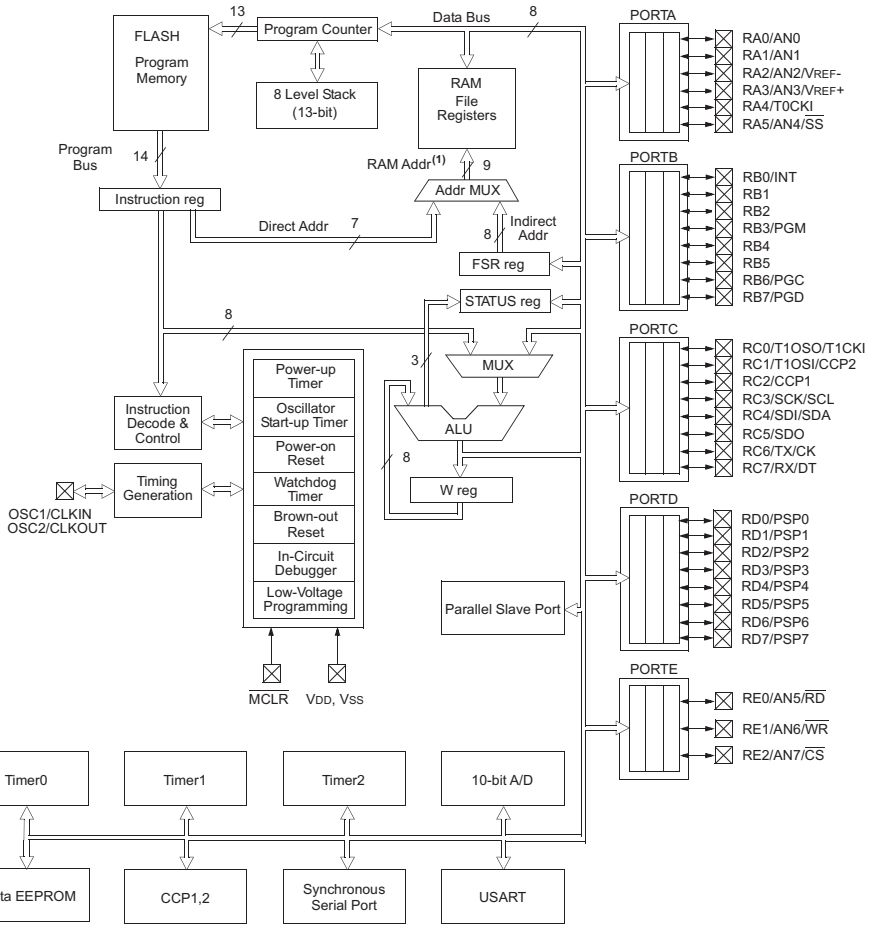



Figura 2.5: Arquitectura abierta de los microcontroladores PIC16F877.
*Fuente: [Microchip, 2001].*

En la Figura 2.6, se observa la organización de la memoria tipo FLASH de los PIC16F877, que está dividida en las 4 páginas con sus correspondientes vectores de reset y de interrupciones (Cf. [Angulo, 1999]).

### Memoria de datos RAM

La memoria de datos tiene posiciones implementadas en RAM, donde se alojan registros operativos fundamentales en el funcionamiento del procesador y en el manejo de los periféricos. La RAM estática consta de cuatro bancos con 128 bytes cada uno, en estos se ubican registros específicos que gobiernan al procesador y los recursos (Cf. [Angulo, 2000]).

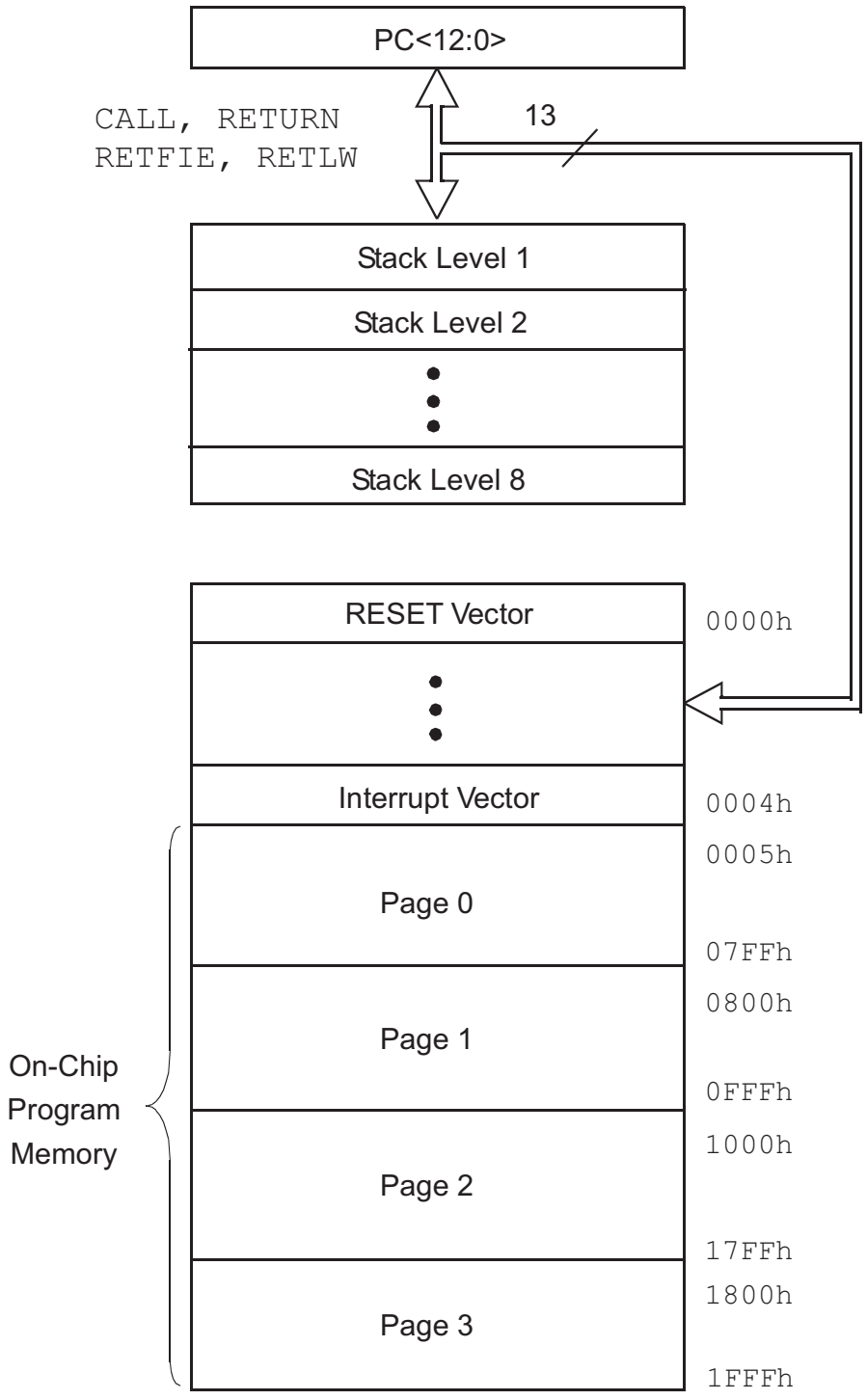



Figura 2.6: Organización de la memoria FLASH de los PIC16F877.
*Fuente: [Microchip, 2001].*

En la Figura 2.7 se muestra la distribución de la memoria RAM en los cuatro bancos.

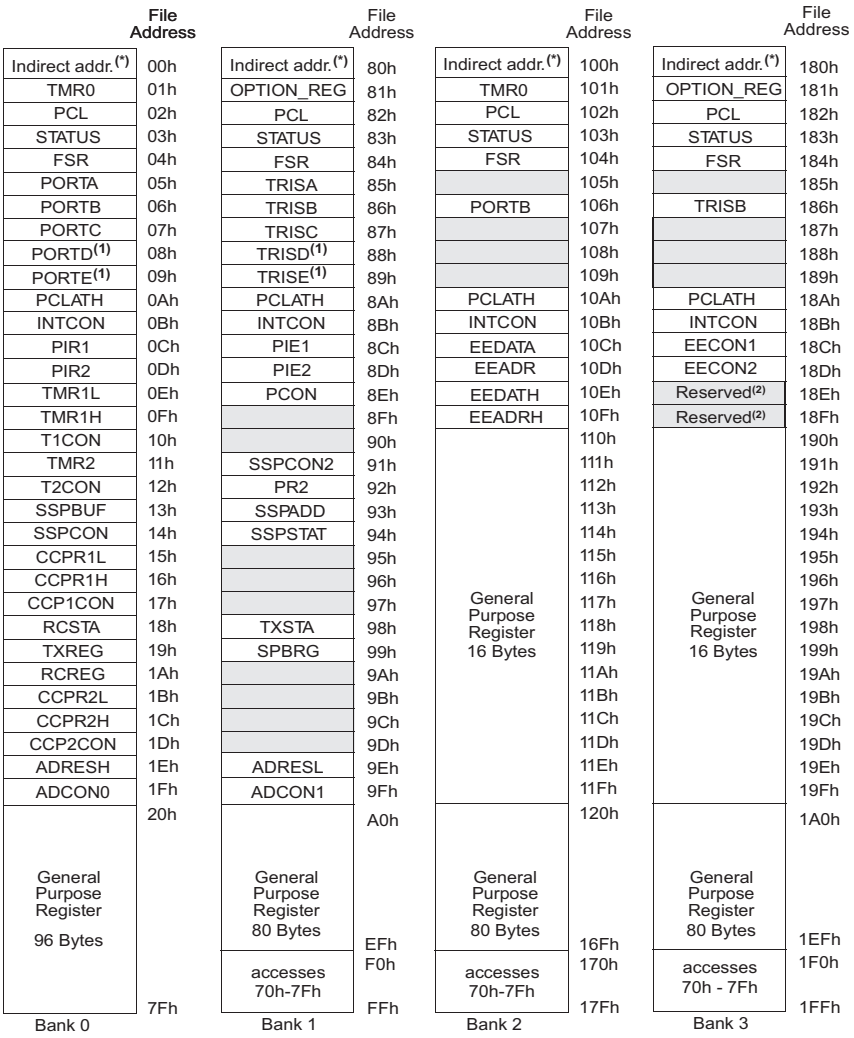



Figura 2.7: Distribución de la memoria RAM en los cuatro bancos.
*Fuente: [Microchip, 2001].*

El PIC16F877 tiene 368 bytes de capacidad y para seleccionar el banco al que se desea acceder en la RAM se emplean los bits 5 y 6 del registro de estado del PIC, denominados RP0 y RP1 respectivamente (*Ibidem*).

La Tabla 2.1 muestra el código para acceder a los diferentes bancos de la memoria RAM.

| BANCO | RP1 | RP0 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

Tabla 2.1: Bits de selección de banco en el registro de estado.

### Palabra de configuración

La palabra de configuración es una posición reservada de la memoria de programa FLASH, que ocupa la dirección 2007h y que sólo es accesible durante la programación del PIC, en el cual el valor de sus bits determina algunas características fundamentales (Cf. [Angulo, 2000]).

En el caso de los microcontroladores PIC16F877 la palabra de configuración se compone de diversos bits donde se activan o desactivan sus características como:

- Código de protección de memoria de programa (CP1:CP0).
- Modo depurador en circuito (DEBUG).
- Permiso de escritura en la memoria FLASH (WRT).
- Código de protección de la memoria EEPROM (CPD).
- Permiso para programación en bajo voltaje (LVP).
- Permiso para reset por baja tensión (BODEN).
- Permiso para el timer de conexión de alimentación (PWRTE).
- Permiso de timer del perro guardián (WDTE).
- Tipo de oscilador (FOSC1:FOSC0).

### Memoria de datos EEPROM

Esta memoria es de tipo no volátil de 256 bytes en este microcontrolador. Soportan hasta 100,000 operaciones de grabado/borrado del microcontrolador.

#### Tipos de osciladores

Los PIC permiten hasta 8 diferentes modos de implementación del oscilador. El usuario puede seleccionar alguno de estos modos programando 3 bits de configuración del dispositivo denominados: FOSC2, FOSC1 y FOSC0.

En algunos de estos modos el usuario puede indicar que se genere o no una salida del oscilador (CLKOUT) a través de una línea de entrada/salida. Los modos de operación se muestran en la siguiente lista:

- LP que involucra frecuencia baja y bajo consumo de potencia.
- XT (cristal) es un resonador cerámico externo para media frecuencia.
- HS que significa alta velocidad y alta potencia (cristal/resonador).
- RC que involucra una oscilador con una resistencia y un capacitor.

En el presente trabajo de tesis, el oscilador utilizado es un cristal resonador de 20Mhz para lo cual se utilizaron los capacitores recomendados para este oscilador de $22\rho$ f.

### 2.2.3. Registros generales del microcontrolador PIC16F877

#### Registros de estado

Para gobernar el funcionamiento de los recursos de los PIC existe un conjunto de registros específicos cuyos bits soportan el control de los mismos. Dichos registros están ubicados en las primeras posiciones de cada banco de la memoria de datos RAM (Cf. [Angulo, 2000]). En la Figura 2.7, se muestra el contenido de la memoria de datos RAM del PIC16F877 y la denominación y situación de los registros específicos junto a los de propósito general.

#### Registro de estado (STATUS)

Éste es el registro más usado de todos, dado que sus bits están destinados a controlar las funciones vitales del procesador. Por este motivo, está duplicado en las cuartas posiciones de cada banco. Los tres bits de menor peso son los señalizadores de ciertas condiciones en las operaciones lógico-aritméticas y los tres bits de más peso se emplean para seleccionar el banco de la RAM al que se desea acceder. La Tabla 2.2 muestra la estructura interna del registro (Cf. [Angulo, 2000]).

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| IRP | RP1 | RP0 | TO# | PD# | Z | DC | C |

Tabla 2.2: Estructura interna del registro de estado.

**Registro de opciones (OPTION_REG)**

El registro OPTION_REG contiene varios bits de control para configurar el divisor de frecuencia del Timer 0 y del perro guardián, las interrupciones externas INT y las resistencias del pull-up del puerto B (Cf. [Microchip, 2001]).

El registro OPTION_REG toma el valor 1111 1111 en cualquier tipo de reinicialización que se produzca.

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| RBPU | INTEDG | TOCS | TOSE | PSA | PS2 | PS1 | PS0 |

Tabla 2.3: Estructura interna del registro de opciones.

### 2.2.4. Registros para control de interrupciones

Los PIC16F877 tienen 14 causas que pueden originar una interrupción, la mayoría de los recursos o periféricos de que dispone los PIC16F877 son capaces de ocasionar éstas si se programan adecuadamente los bits de los registros. (Cf. [Angulo, 2000])

Las causas que provocan una interrupción en el PIC16F877, son las siguientes:

1. Desbordamiento del Timer 0.
2. Desbordamiento del Timer 1.
3. Desbordamiento del Timer 2.
4. Activación de RBO/INT.
5. Captura o comparación en el módulo CCP1
6. Captura o comparación del módulo CCP2.
7. Transferencia en el puerto serie síncrona.
8. Colisión de bus del puerto serie síncrona.

9. Fin de la transmisión en el USART.
10. Fin de la recepción en el USART.
11. Fin de la conversión en el convertidor A/D.
12. Transferencia en la compuerta paralela esclava.
13. Cambio de estado en los 4 bits más significativos del puerto B.
14. Finalización en la escritura de un byte en la EEPROM.

### Registro de control de interrupciones (INTCON)

Este es un registro de lectura/escritura que, para facilitar su acceso, se halla duplicado en los cuatro bancos de la memoria. Tiene la misión de controlar las interrupciones provocadas por el Timer 0, controlar las interrupciones dado un cambio de estado en los 4 bits más significativos del puerto B y por activación del bit RB0/INT (*Ibidem.*). En la Tabla 2.4 se muestra el registro interno del registro de interrupciones INTCON.

| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
|---|---|---|---|---|---|---|---|
| **GIE** | **PEIE** | **TMR0IE** | **INTE** | **RBIE** | **TMROIF** | **INTF** | **RBIF** |

Tabla 2.4: Estructura interna del registro de interrupciones.

### Registro de permiso de interrupciones 1 (PIE1)

Contiene los bits que permiten o prohíben las interrupciones provocadas por los periféricos internos del microcontrolador y que no estaban contempladas en el INTCON. Para que cumplan su función los bits del PIE1 es necesario que PEIE está activado o en 1 (*Ibidem.*).

Este registro cuenta con ocho bits, cada bit del registro PIE1 tiene un propósito en general:

1. PSPIE: permiso de interrupción para la puerta paralela
2. ADIE: permiso de interrupción para el conversor A/D al finalizar la conversión.
3. RCIE: permiso de interrupción para el receptor del USART cuando el buffer se llena.
4. TXIE: permiso de interrupción para el transmisor del USART cuando el buffer se llena.

5. SSPIE: permiso de interrupción para un puerto serie síncrona.
6. CCP1IE: permiso de interrupción para el módulo CCP1 cuando se produce una captura o comparación.
7. TMR2IE: permiso de interrupción para el Timer 2 con su desbordamiento.
8. TMR1IE: permisos de interrupción para el Timer 1 con su desbordamiento.

En la Tabla 2.5 se puede observar la estructura interna de este registro.

| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
|---|---|---|---|---|---|---|---|
| **PSPIE** | **ADIE** | **RCIE** | **TXIE** | **SSPIE** | **CCP1IE** | **TMR2IE** | **TMR1IE** |

Tabla 2.5: Estructura interna del registro PIE1.

#### Registro de señalizador de interrupciones (PIR1)

En este registro, los bits actúan de señalizadores desde el momento en el que se origina la interrupción, independientemente de si está permitida o prohibida en correspondencia con el registro PIE1 (*Ibidem.*). La Tabla 2.6 muestra la estructura interna del registro.

| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
|---|---|---|---|---|---|---|---|
| **PSPIF** | **ADIF** | **RCIF** | **TXIF** | **SSPIF** | **CCP1IF** | **TMR1IF** | **TMR1IF** |

Tabla 2.6: Estructura interna del registro PIR1.

### 2.2.5. Puertos de entrada y salida

Los microcontroladores PIC16F877 tienen disponibles cinco puertos de entrada/-salida denominados PORTA, PORTB, PORTC, PORTD y PORTE. Todas las líneas de estos puertos son totalmente programables, es decir, pueden ser configuradas para trabajar como entradas o como salidas.

#### Puerto A

Este puerto solo dispone de 6 líneas, denominadas RA0-RA5, son bidireccionales y su sentido queda configurado según la programación de los bits del registro TRISA. Si el registro TRISA se programa en uno, las líneas del puerto A quedarán configuradas como entradas, por el contrario si el registro se programa en 0, las líneas del puerto A

quedarán como salidas.

Las líneas RA0/AN0, RA1/AN1 y RA2/AN2, RA3/AN3 y RA5/AN4 (ver Figura 2.4) además de líneas de entrada/salida digitales, también pueden actuar como canales 0, 1, 2, 3 y 4 respectivamente, por los que se le puede aplicar una señal analógica al convertidor Analógico/Digital. En este proyecto estas líneas son utilizadas para el convertidor Analógico/Digital.

Por otra parte existen líneas con funciones multiplexadas como la línea RA4/T0CK1, que actúa como entrada/salida digital y como entrada de reloj para el timer 0, y la línea RA5/AN4/SS# que además de entrada/salida digital, se utiliza para seleccionar el modo esclavo cuando se trabaja en comunicación síncrona.

Para seleccionar si las líneas del puerto A serán entradas o salidas digitales o como canales de entrada para el conversor A/D, se debe escribir el valor adecuado en el registro ADCON1.

### Puerto B

Este puerto dispone de ocho líneas bidireccionales y trabaja de forma similar al puerto A, o sea, como entrada/salida digital, cuya función se elige mediante la programación del registro TRISB, al igual que el puerto A con TRISA. Tiene tres registros asociados:

1. PORTB: En este registro los ocho bits que contiene reflejan directamente el estado de las ocho líneas del puerto B: RB0,...,RB7 (ver Figura 2.4).

2. TRISB: En forma similar a TRISA, al poner un 0 en un bit de TRISB se configura la línea RB correspondiente como salida y al poner un 1 en un bit de TRISB se configura la línea RB correspondiente como entrada.

3. OPTION_REG: El bit 7 de este registro, denominado RBPU es usado para conectar/desconectar una resistencia *pull-up* conectada a cada línea RB.
   Poniendo un 0 en este bit, todas las resistencias se conectan. Para desconectar las resistencias *pull-up* se debe poner este bit en 1, también se desconectan automáticamente cuando la línea correspondiente es configurada como salida.

### Puerto C

Consta de 8 líneas bidireccionales (ver Figura 2.4), cuyo sentido se configura mediante el registro TRISC, al igual que el puerto A y el puerto B. Todas las líneas de este puerto además de ser entradas/salidas digitales, tienen multiplexadas diferentes funciones, entre las más destacadas y utilizadas en este proyecto de tesis se mencionan:

- RC0/T1OS0/T1CK1: actúan también como salida del Timer 1 o como entrada de impulsos para el Timer 1.
- RC1/T1OS1/CCP2: trabaja como entrada al oscilador del Timer 1.
- RC6/TX/CK: una de las líneas más importantes y utilizadas ya que se usa como línea de transmisión en el USART.
- RC7/RX/DT: utilizada en este proyecto para la recepción del USART.

Cuando se habilita la línea del periférico respectivo puede ser ignorada la configuración de TRISC, de hecho, algunos periféricos configuran la línea como salida mientras que otros la configuran como entrada (Cf. [Dumetri, 2002]).

#### Puerto D

Este puerto D dispone de 8 líneas bidireccionales, cuyo sentido se configura con el registro TRISD al igual que el puerto A, B y C. En éste todas las líneas disponen en la entrada de un Dispador tipo Schmitt (ver Figura 2.4). Además de usar las líneas como entradas/salidas digitales, implementan un puerto paralelo esclavo de 8 líneas (PSP). Para que funcionen como puerto de comunicación paralelo, se debe poner el bit PSPMODE=1 (Cf. [Angulo, 2000]).

#### Puerto E

Sólo posee 3 líneas configurables como entradas o salidas mediante los 3 bits menos significativos del registro TRISE. Se puede tener acceso a sus líneas mediante los 3 bits menos significativos del registro PORTE.

Estas líneas están compartidas con el convertidor analógico/digital, por ello, antes de usarlas deberán ser configuradas como entradas/salidas digitales o analógicas, según se desee en forma similar a como se hizo con el puerto A, usando el registro de configuración ADCON1 (Cf. [Dumetri, 2002]).

### 2.2.6. Registros de los temporizadores

#### Registro de temporizador del Timer 1 (TMR1)

El PIC16F877 dispone de un conjunto de tres temporizadores: TMR0, TMR1 y TMR2. El Timer 1 se caracteriza por tener un contador/temporizador de 16 bits y por ser de lectura y escritura. Se puede seleccionar el tipo de reloj a utilizar (reloj interno o externo) y cuenta con interrupción opcional por desbordamiento de FFFFh hasta 0000h (Cf. [Angulo, 2000]).

El Timer 1 es el único temporizador/contador ascendente de 16 bits, por lo que requiere de dos registros concatenados: TMR1H:TMR1L, que son los encargados de almacenar el valor del conteo en cada momento. Dicho valor evoluciona desde 0000h hasta FFFFh, instante en el cual se activa el señalizador TMR1IF y regresa al valor inicial. También si se desea se puede provocar una petición de interrupción (*Ibidem.*).

En el modo temporizador del Timer 1, el valor en TMR1H:TMR1L se incrementa con cada ciclo de instrucción (Fosc/4). En la Figura 2.8 se muestra el diagrama por bloques del Timer 1 en el que destacan las diversas señales de control y el predivisor de frecuencia.

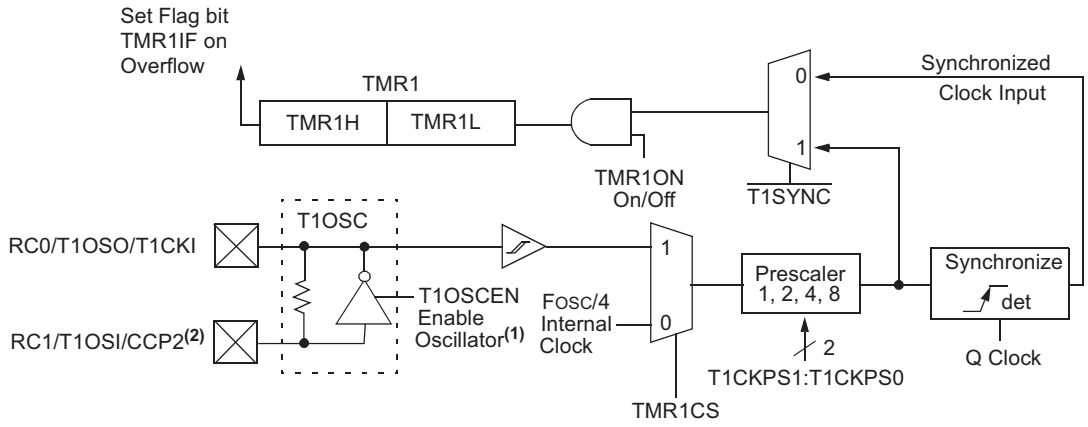



Figura 2.8: Esquema interno de los principales bloques del timer 1.
*Fuente: [Microchip, 2001].*

### Registro de control del Timer 1 (T1CON)

El registro que gobierna el funcionamiento del Timer 1 es el T1CON ubicado en el banco 1 de la memoria de datos RAM (ver Figura 2.7). La estructura interna se observa en la Tabla 2.7

| T1CKPS1 | T1CKPS2 | T1OSCEN | T1SYN | TMR1CS | TMR1ON |
|---|---|---|---|---|---|

Tabla 2.7: Estructura interna del registro T1CON.

El bit TMR1ON (bit 0) gobierna el permiso o la prohibición del funcionamiento del Timer 1, en caso de poner el bit en cero, el Timer 1 no funcionará. El bit TMR1CS selecciona la fuente de los impulsos de conteo, si vale 0 se selecciona el reloj interno

(Fosc/4), si vale 1 es el reloj externo el que se aplica en las líneas RC0 y RC1. En ambos casos el registro TRISC carece de significado, ya que una o ambas líneas RC1 y RC0 no pueden actuar al mismo tiempo como entrada de impulsos y como líneas de entrada/salida (Cf. [Angulo, 2000]).

El predivisor de frecuencia (preescaler) es un simple divisor de frecuencia de los impulsos que se aplican al TMR1 por 1, 2, 4 u 8. El rango de división lo eligen los bits T1CKPS1 y T1CKP0 según la Tabla 2.8.

| T2CKPS1 | T2CKPS0 | RANGO DEL PREDIVISOR |
|---|---|---|
| 0 | 0 | 1:1 |
| 0 | 1 | 1:2 |
| 1 | 0 | 1:4 |
| 1 | 1 | 1:8 |

Tabla 2.8: Bits para selección del rango del predivisor.

El Timer 1 puede generar una petición de interrupción cuando se produce el desbordamiento del contador, es decir cuando se pasa de FFFFh a 0000h. En esta situación automáticamente la bandera TMR1IF (bit 0 del registro PIR1) cambia su valor a 1. El permiso o prohibición de la producción de interrupción del Timer 1 está controlada por el TMR1IE del bit 0 del registro PIE1 (Cf. [Angulo, 2000]).

### 2.2.7. Registros de comunicación serial

Este microcontrolador dispone de un módulo transmisor denominado USART (Universal Synchronous Asynchronous Receiver Transmitter) capaz de soportar la comunicación serial síncrona y asíncrona. El USART puede configurarse de modo asíncrono full dúplex el cual que puede comunicar dispositivos periféricos como ordenadores personales, o como un sistema síncrono half duplex para comunicarse con otros microcontroladores, dispositivos periféricos como los convertidores A/D, los circuitos integrados D/A, etc. (Cf. [Microchip, 2001]).

La configuración asíncrona que es la que se realizó en este trabajo de tesis, se verá con mayor detalle en la Sección 4.1.1.

#### Registro de estado del transmisor (TXSTA)

Este registro tiene la función de controlar la transmisión serial y su estado, se conforma de 8 bits y sus funciones van desde elegir el tipo de transmisión hasta la velocidad de transmisión. La función de cada bit se describe a continuación:

- Bit 7: CSRC.
  Este bit tiene la función de seleccionar la fuente de reloj para usarse en transmisión síncrona, y si la transmisión elegida es transmisión asíncrona no importa el valor que se le asigne.

- Bit 6: TX9.
  Cuando este bit está en estado activo (igual a 1) la transmisión será a 9 bits, mientras que cuando se encuentre en estado inactivo (igual a 0), la trasmisión se hará a 8 bits.

- Bit 5: TXEN.
  La función de este bit es habilitar la transmisión, esto mientras el bit sea igual a 1, y cuando se encuentre igual a 0 la transmisión quedará desactivada.

- Bit 4: SYNC.
  Este bit selecciona el tipo de transmisión que ha de llevarse a cabo, si el bit es igual a 1 el tipo de transmisión será síncrona, si el bit es igual a 0 el tipo será asíncrona.

- Bit 3: no se encuentra implementado.

- Bit 2: BRGH.
  Éste bit selecciona la velocidad de la tasa de baudios a la que se ha de transmitir el byte, éste no se aplica para transmisión síncrona. En transmisión asíncrona, se puede elegir alta o baja velocidad, poniendo a 1 ó 0 respectivamente este bit.

- Bit 1: TRMT.
  Éste es el bit de estado del registro de desplazamiento del transmisor (TSR), cuando se encuentra en 1, significa que el TSR se encuentra vacío y si se encuentra igual a 0 significa que el TSR está lleno.

- Bit 0: TX9D.
  Este es el noveno bit de datos cuando se elige transmitir a 9 bits, también se puede emplear como bit de paridad.

La estructura interna del registro TXSTA se muestra en la Tabla 2.9.

| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
|---|---|---|---|---|---|---|---|
| **CSRC** | **TX9** | **TXEN** | **SYNC** | **-** | **BRGH** | **TRMT** | **TX9D** |

Tabla 2.9: Estructura interna del registro TXSTA.

**Registro de estado del receptor (RCSTA)**

Consta de 8 bits cuya función es configurar el estado y el control del receptor en la transmisión serial, su configuracion interna se muestra en la Tabla 2.10 y las funciones de cada bit se describen a continuación:

- Bit 7: SPEN.
  Éste es el bit bit de habilitación del puerto serie mientras se encuentre en 1, se habilita el puerto serie (configura los pines RC7/RX/DT y RC6/TX/CK para el puerto serie) y si se encuentra en 0 deshabilita el puerto serie.

- Bit 6: RX9.
  Este bit habilita la recepción de 9 u 8 bits poniendo a 1 ó 0 respectivamente.

- Bit 5: SREN.
  Este bit habilita la recepción única en el modo asíncrono su valor no importa, en modo síncrono maestro mientras su valor sea 1 habilita una recepción única y cuando su valor sea 0 deshabilita la recepción única.

- Bit 4: CREN.
  Es un bit de habilitación de recepción continua tanto para la transmisión síncrona como para la asíncrona.

- Bit 3:ADDEN.
  Este bit tiene la función de habilitar o deshabilitar la detección de dirección del bit. Si su valor es igual a 1 se habilita la detección de dirección, y si su valor es igual a 0, se deshabilita la detección de dirección. Todos los bits son recibidos y el noveno bit es usan como bit de paridad.

- Bit 2: FERR.
  Este bit muestra el estado del error de empaquetamiento si su valor es 1 existe error de empaquetamiento y si se encuentra en 0 significa que no existe error.

- Bit 1: OERR.
  Este bit muestra el estado del error por desbordamiento, si su valor sea igual a 0 significa que no existe error.

- Bit 0: RX9D. En este bit se aloja el noveno bit de datos de recepción ya se puede emplear como bit de paridad

### 2.2.8. Registros del módulo de conversión Analógico/Digital

El módulo de conversión Analógico/Digital dispone de 8 líneas en el microcontrolador PIC16F877. A través de las entradas analógicas se aplica la señal analógica a un

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| **SPEN** | **RX9** | **SREN** | **CREN** | **ADDEN** | **FERR** | **OERR** | **RX9D** |

Tabla 2.10: Estructura interna del registro RCSTA.

condensador de captura y retención (sample and hold), que después se introduce en el convertidor. El convertidor de aproximaciones sucesivas da como resultado una palabra de 10 bits. La resolución que tiene cada bit procedente de la conversión, tiene un valor que está en función de la tensión de referencia $V_{ref}$, de acuerdo con:

$$resolucion = (V_{ref+} - V_{ref-})/1{,}024 = V_{ref}/1{,}024 \tag{2.1}$$

Así, si la tensión de referencia positiva es 5 Volts y la negativa es tierra (0 Volts), entonces la resolución es de 4.8 mV/bit.

El funcionamiento del conversor A/D requiere de la manipulación de cuatro registros:

1. ADRESH (parte alta del resultado de la conversión).
2. ADRESL (parte baja del resultado de la conversión).
3. ADCON0 (registro de control 0).
4. ADCON1 (registro de control 1).

En la pareja de registros ADRESH:ADRESL se deposita el resultado de la conversión, que al estar compuesta por 10 bits, solo son significativos 10 de los 16 bits de esa pareja de bytes (*Ibidem.*).

El módulo de conversión Analógico/Digital tiene la posibilidad de justificar este resultado, y la selección de este formato de justificación a la izquierda o derecha se realiza mediante la configuración del registro de control 1 (ADCON1). Los bits restantes (a los 10 de la conversión) se llenan con ceros, estos dos registros cuando el convertidor Analógico/Digital está en OFF y no se utilizan, pueden utilizarse como dos registros de 8 bits de propósito general (Cf. [Microchip, 2001]).

### Registro de control 0 para módulo de conversión Analógico/Digital

El registro ADCON0 controla la operación del convertidor Analógico/Digital y se encuentra en el banco 0 de la memoria de datos RAM (*Ibidem.*). El registro ADCON0 consta de 8 bits los cuales se definen a continuación:

- Bits 7-6: ADCS1:ADCS0.
  Estos bits seleccionan el tipo de reloj para el Convertidor Analógico/Digital. En la Tabla 2.11 se muestra los tipos de reloj disponibles y el dígito que se ha de cargar en éstos bits, al igual que el dato que se ha de cargar en el bit ADCS2 del registro ADCON1.

| **<ADCS2> ADCON1** | **<ADCS1:ADCS0> ADCON0** | **Reloj de Conversión** |
|---|---|---|
| 0 | 0 0 | Fosc/2 |
| 0 | 0 1 | Fosc/8 |
| 0 | 1 0 | Fosc/32 |
| 0 | 1 1 | $F_{RC}$ |
| 1 | 0 0 | Fosc/4 |
| 1 | 0 1 | Fosc/16 |
| 1 | 1 0 | Fosc/64 |
| 1 | 1 1 | $F_{RC}$ |

Tabla 2.11: Tabla de selección de bits para elegir reloj de conversión.

- Bits 5-3: CHS2:CHS0.
  Estos tres bits tienen la función de seleccionar el canal analógico que se ha de utilizar en el proceso de conversión Analógico/Digital, respecto a la Tabla 2.12.

| **<ADCS1:ADCS0>** | **Canal de Conversión** |
|---|---|
| 0 0 0 | Canal 0 (AN0) |
| 0 0 1 | Canal 1 (AN1) |
| 0 1 0 | Canal 3 (AN2) |
| 0 1 1 | Canal 3 (AN3) |
| 1 0 0 | Canal 4 (AN4) |
| 1 0 1 | Canal 5 (AN5) |
| 1 1 0 | Canal 6 (AN6) |
| 1 1 1 | Canal 7 (AN7) |

Tabla 2.12: Tabla de selección de canal analógico en el proceso de conversión.

- Bit 2: GO/DONE. Este bit indica el estado de la conversión, cuando el bit sea igual a 1 indica que la conversión está en progreso, el cual cambiará automáticamente a 0 por hardware cuando la conversión ha terminado, esto ocurrirá siempre y cuando el bit ADON se encuentre en estado activo.

- Bit 0: ADON. Este bit tiene la función de activar o desactivar el módulo de conversión, cuando el bit se encuentre en 1, el módulo de conversión Analógico/Digital se enciende y cuando se encuentra en 0 este módulo se encuentra apagado.

El convertidor Analógico/Digital no trabajará correctamente con un TAD menor que TAD(mínimo) = 1.6 $\mu$s. El usuario deberá cuidar la elección del reloj adecuado para no violar esta limitante. La estructura interna del registro ADCON0 se observa en la Tabla 2.13 que ese muestra a continuación.

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| **ACS1** | **ADCS0** | **CHS2** | **CHS1** | **CHS0** | **GO/DONE** | **-** | **ADON** |

Tabla 2.13: Estructura interna del registro ADCON0.

**Registro de control 1 para módulo de conversión Analógico/Digital**

Este registro tiene la función de configurar las líneas del puerto A como entradas analógicas o entradas/salidas digitales y su estructura se muestra en la Tabla 2.14.

El bit de menos peso ADFM de este registro selecciona el formato del resultado de la conversión. Si vale 1, los 2 bits más significativos del resultado de la conversión serán alojados en las 2 primeras posiciones del registro ADRESH, dejando las 6 posiciones más significativas en 0, mientras que los siguientes 8 se encontrarán en el registro ADRESL. Y si ADFM es igual a 0, los 8 bits más significativos del resultado se encontrarán en el registro ADRESH y los 2 menos significativos se alojarán en las posiciones 6 y 7 del registro ADRESL, dejando las siguientes 6 igual a 0 (Cf. [Angulo, 2000]).

Los restantes 4 bits (PCFG3:PCFG0) de este registro ADCON1, se usan para configurar las líneas de los canales de entrada como entradas analógicas o como entradas / salidas digitales.

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| **ADFM** | **ADCS2** | – | – | **PCFG3** | **PCFG2** | **PCFG1** | **PCGG0** |

Tabla 2.14: Estructura interna del registro ADCON1.

## 2.3. Pantalla de LCD para visualización

Los módulos LCD están compuestos básicamente por una pantalla de cristal líquido y un circuito microcontrolador especializado el cual posee los circuitos y memorias de

control necesarias para desplegar el conjunto de caracteres ASCII, un conjunto básico de caracteres japoneses, griegos y algunos símbolos matemáticos por medio de un circuito denominado generador de caracteres. La lógica de control se encarga de mantener la información en la pantalla hasta que ella sea sobreescrita o borrada en la memoria RAM de datos. La pantalla de cristal líquido está conformada por una ó dos líneas de 8, 16, 20, 24 ó 40 caracteres de 5x7 pixels c/u.

Estas pantallas alfanuméricas de cristal líquido, denominadas abreviadamente LCD, constituyen uno de los visualizaciones de mensajes más económicos, prácticos y eficaces. El control de esta pantalla, se realiza con el microcontrolador, en el puerto B disponible en éste.

## 2.4. Puerto de comunicación serial RS-232

Las transmisiones serie se caracterizan por utilizar una única línea para transmitir la información. Esto obliga en el transmisor a convertir los bytes de paralelo a serie para enviar los bits de forma secuencial, por su parte en el receptor la serie de bits recibidos son convertidos de serie a paralelo para reconstruir el byte. Inicialmente el puerto serie se utilizó para conectar la PC y un módem. En esta conexión, a la PC se le denomina DTE (Data Terminal Equipment) y al Módem DCE (Data Communication Equipment).

Existen dos tipos posibles de transmisiones serie: síncrona y asíncrona. En las transmisiones asíncronas a cada byte de información a transmitir se le añaden una serie de bits fijos de señalización para marcar el comienzo (bit de arranque o start) y el final de cada byte (bits de parada o stop), esto se explicará con más detalle en la Sección 4.1.1.

El DTE puede ser un dispositivo inteligente como una PC o un autómata, mientras que el DCE suele ser un dispositivo no inteligente como un módem, una impresora, etc. Por lo que se estableció la norma RS-232, la cual define una velocidad de transmisión serial de hasta 20 Kbps y longitud máxima de 15 metros; y las señales eléctricas se definen entre +3 y +15 volts para el estado activo (uno lógico) y de -3 y -15 para el estado inactivo (cero lógico), además de que sólo permite conectar dos dispositivos para establecer comunicación punto a punto.

Actualmente la mayoría de las PC poseen dos puertos serie definidos como COM1 y COM2 a nivel de sistema operativo y utilizan conectores DB9 macho. La ventaja de esta técnica es que es más fácil de implementar, pero una de sus desventajas es que además de cada byte de información se envían tres bits más que no son de información.

En este proyecto de tesis se utilizó el registro USART en su modo comúnmente usado asíncrono para comunicar al puerto serial usando el protocolo RS-232 por lo que

se requirió una interfaz a RS-232 debido a los distintos niveles de voltaje, ya que estos no deben ser conectados directamente a las señales RS-232.

### 2.4.1. Convertidor de interfaz RS-232 a RS-485

Como ya se describió en la sección anterior la norma RS-232 define una comunicación punto a punto serial, pero en el presente proyecto de tesis, se utilizó una comunicación multipunto serial y para ello se debió implantar la norma RS-485.

Mientras que la norma RS-232 solo permite enlaces punto a punto entre dos nodos, la norma RS-485 permite enlaces multipunto o multinodo mediante la conexión de un bus de dos hilos entre todos los nodos, para formar una topología física en bus y la longitud máxima sin atenuaciones es de aproximadamente 1200 metros a una velocidad de 90Kbps a velocidad máxima de enlace de 10Mbps.

El sistema mínimo tiene disponible el puerto de comunicación serial del microcontrolador (USART) basado en la norma RS-232 al cual se le interconecta un módulo de conversión de norma RS-232 a RS-485, mediante circuitos integrados MAXIM 233. Con esta conversión, se obtienen todas las ventajas que la norma RS-485 ofrece, las cuales se verán con mayor detalle en la Sección 3.1.

Como ya se mencionó, los sistemas mínimos deben estar comunicándose con la Unidad Central o PC, para lo cual se utilizó un convertidor de norma RS-485 a RS-232, norma definida para comunicación serial en computadoras, utilizando el circuito integrado MAXIM 485. La figura 2.9 muestra la configuración del MAXIM 485.

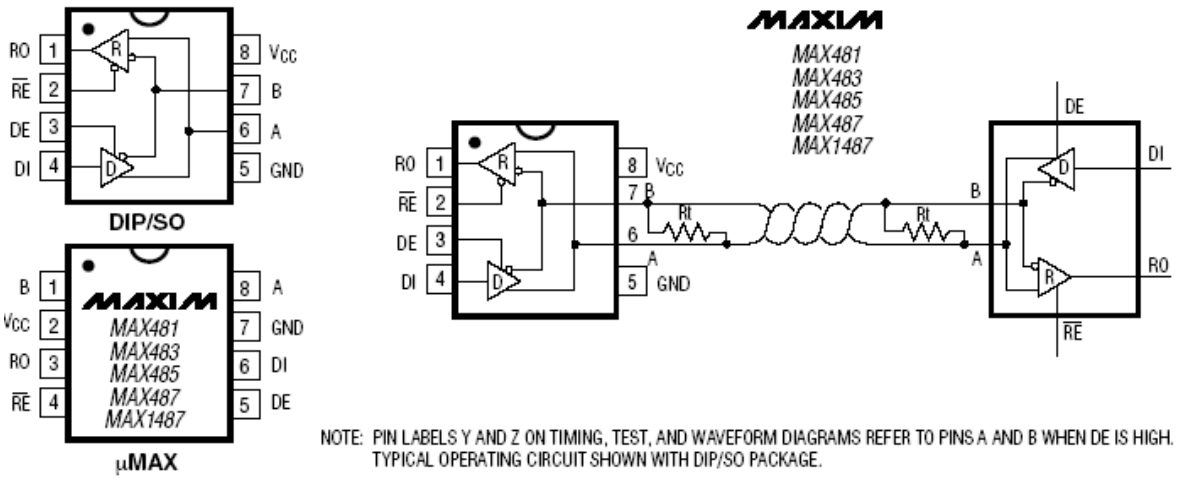



Figura 2.9: Configuración del MAXIM 485.

De esta manera, en el presente proyecto de tesis, mediante dispositivos que se interconectan a los sistemas mínimos propuestos y mediante la configuración de cada uno de los registros antes mencionados, fue posible la realización de la comunicación serial distribuida, cuya configuración se explicará con mayor detalle en los capítulos siguientes.

# Capítulo 3

# Diseño del protocolo de comunicación serial distribuido

**Introducción**

Cuando varios dispositivos se comunican a distancia por medio de una red de comunicación de datos, lo que se hace realmente es intercambiar datos de un dispositivo a otro. Para poder realizar esta comunicación de datos se hace necesario la utilización de un protocolo, el cual es un conjunto de reglas establecidas para asegurar el intercambio ordenado de información entre los participantes de una comunicación o entre los niveles en los que se articula la misma. Los participantes de cualquier comunicación se entenderán mucho mejor si siguen estas reglas (formatos de información, turnos de espera, etc.).

## 3.1. Norma RS-485

La norma RS-485 permite enlaces multipunto o multinodo (más de 2 nodos) mediante la conexión de bus de 2 o 4 hilos entre todos los nodos para formar una red con topología física en bus. Todos lo nodos pueden escuchar el medio, pero sólo uno de ellos puede transmitir en un mismo instante, se trata por tanto de comunicaciones half-duplex con 2 hilos o full duplex con 4 hilos.

El mayor beneficio de la norma RS-485 es que esta puede operar una línea en 3 estados llamados estados de operación:

1. 1 lógico,
2. 0 lógico,
3. Alta impedancia.

En el estado de alta impedancia no existe una corriente presente en la línea. Esto es conocido como un estado deshabilitado y puede ser iniciado por una señal a través de un pin de control sobre la línea del driver del circuito integrado. Los 3 estados de operación permiten un conexión multipunto de hasta 32 transmisores que pueden ser conectados en la misma línea, sin embargo solo uno puede ser activado al mismo tiempo. Cada terminal es un sistema multipunto que debe contener una única dirección para evitar conflictos con otros sistemas multipunto. La norma RS-485 limita la corriente en casos donde ocurra una colisión (Cf. [Inc, 2002]).

De acuerdo con la norma RS-485, la señal está balanceada, al contrario que en la norma RS-232 donde no lo está. Esto quiere decir que se utiliza la diferencia de potencial entre los 2 hilos para establecer el nivel lógico que hay en la línea.

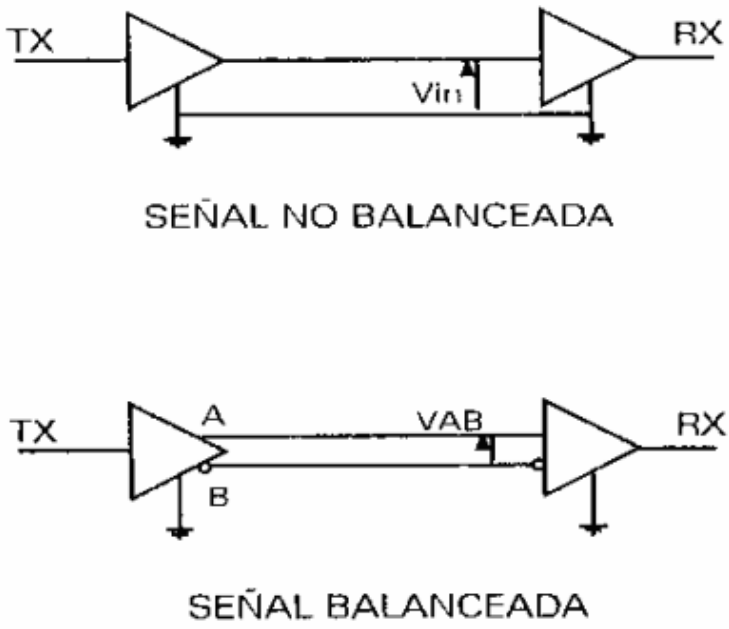



Figura 3.1: Transmisión balanceada /no balanceada.
*Fuente: [Morcillo, 2000].*

Las señales balanceadas son mucho más inmunes al ruido eléctrico, debido a que las interferencias se acoplan por igual a los dos hilos y, por tanto, su diferencia es cero. Ésta inmunidad al ruido eléctrico las hace ideales para entornos industriales. Si utilizamos un cable en par trenzado se mejora aún más la inmunidad al ruido.

El módulo de la tensión diferencial VA - VB debe de encontrarse entre 1.5 y 6 volts y la sensibilidad es de 0.2 volts. Se considera uno lógico cuando la tensión VA es 0.2

volts mayor que la tensión VB, y cero lógico cuando la tensión de VB es 0.2 volts mayor que VA. La Figura 3.2 muestra la señal típica RS-485; se ha dibujado la señal en la línea asumiendo una tensión base de 3 volts. Obsérvese que la tensión VAB tiene un valor pico a pico de 6 volts.

La longitud máxima del enlace es de aproximadamente 1 200 metros a una velocidad de 90 Kbps, y la velocidad máxima 10 Mbps. La velocidad y longitud del enlace son inversamente proporcionales, por ejemplo, si deseamos obtener la máxima velocidad, el cable deberá ser de unos pocos metros.

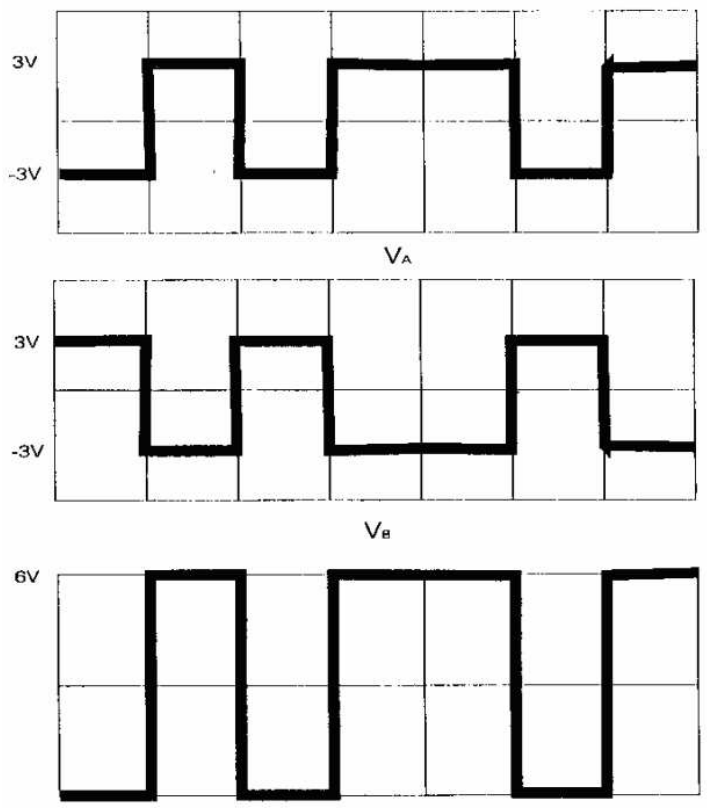



Figura 3.2: Tensión en RS-485.
*Fuente: [Morcillo, 2000].*

Cuando se utilizan adaptadores de doble circuito es posible sustituir un enlace RS-232 por otro RS-485, sin modificar el protocolo de comunicaciones. Esta es otra de las ventajas de esta norma, dado que es transparente para el programador. Para que el software desarrollado para RS-232 funcione en un enlace RS-485, sólo es necesario instalar un adaptador RS-232 a RS-485 en cada nodo y los cables adecuados. También es posible utilizar los adaptadores de doble circuito para formar un enlace multinodo

full-duplex como el de la Figura 3.3 (Cf. [Morcillo, 2000]).

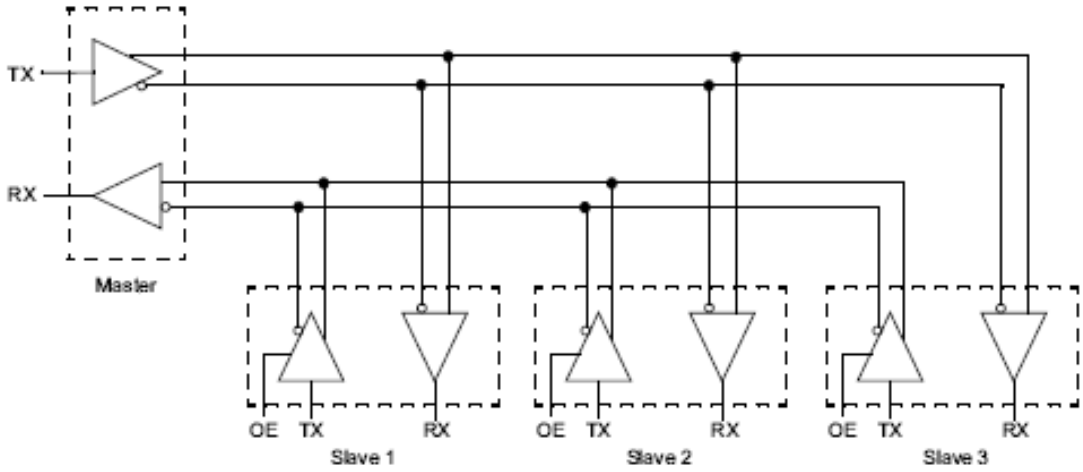



Figura 3.3: Enlace full-duplex multipunto.
*Fuente: [Inc, 2002].*

Un enlace a través de la norma RS-485 puede ser realizado con una configuración de 4 hilos. En este tipo de conexión, un nodo puede ser el maestro y todos los otros nodos serán los esclavos. El maestro puede comunicarse con todos los esclavos, pero un esclavo solamente puede comunicarse con el maestro. Por lo tanto el esclavo nunca escucha la respuesta que le da otro esclavo al maestro, un esclavo no puede contestarle a otro esclavo. Esta es una ventaja de esta arquitectura propuesta dentro del estándar RS-485 (Cf. [Inc, 2002]).

La norma RS-485 está frecuentemente implementada con un maestro y varios esclavos. El maestro es el único dispositivo que inicializa la comunicación en el bus y esto evita que existan problemas de colisión de información. Comúnmente el maestro transmite la dirección y datos, donde los datos son recibidos por un esclavo en particular que previamente ha recibido una dirección. Cada esclavo debe tener una dirección única y debe saber que tipo de dato espera y retornará al maestro. Además cada dispositivo debe controlar la salida de una señal de habilitación de un circuito integrado para habilitar la salida solamente mientras está transmitiendo.

Por lo tanto todos los dispositivos usualmente usan la misma conexión física, dado que esto es esencial para evitar el manejo de señales en el bus excepto cuando se está transmitiendo. El usuario debe asegurarse que los circuitos integrados (receivers) tienen un estado válido cuando el dispositivo no se está manejando en el bus. Regularmente esto se realiza colocando una resistencia en cada extremo del bus (Cf. [Garbutt, 2003]).

## 3.2. Uso de 9 y 8 bits

El USART puede manejar datos de 8 y 9 bits. En la mayoría de la aplicaciones con el RS-232 se usa 8 bits, pero existen diferentes razones para usar 9 bits como lo son:

1. Datos de 9 bits,
2. Datos de 8 bits con 2 bits de parada,
3. Datos de 8 bits con paridad,
4. Datos de 8 y 9 bits para detectar direcciones
5. El bit TX9 en el registro TXSTA y el bit RX9 en el registro RCSTA deben ser puestos en 1 para habilitar la transmisión y recepción en 9 bits.

### 3.2.1. Datos de 8 bits con 2 bits de parada

El noveno bit puede ser usado como un bit extra de parada, si el formato de los datos requiere 2 bits de parada.

Para transmitir los datos con 2 bits de parada, el noveno bit debe ser puesto en el bit TX9D del registro TXSTA antes de escribir los otros datos en el TXREG. El noveno bit puede ponerse a 1 al inicializar el programa, y éste nunca se necesitará cambiarlo. Cuando el dato es transmitido, el noveno bit es usado como la primera parada y la segunda parada es generada por el USART. Al recibir el dato con los 2 bits de parada, el noveno bit se almacena en el bit RX9D del registro RCSTA para ser checado después de cada recepción, antes de leer el RCREG. Si el noveno bit no está puesto en 1, indica que existe un error en el bit de parada. Esto es parecido a un error de trama, el cual ocurre si la segunda parada no se encuentra en 1, y la rutina de error puede ser usada para ambos errores. Es posible usar 8 bits para recibir un dato con 2 bits de parada. En este caso, el error no será detectado en el segundo bit de parada. Esto no es recomendable porque si el segundo bit de parada es cero, el USART lo interpretará como un bit de inicio para la nueva recepción.

### 3.2.2. Datos de 8 bits con bit de paridad

El noveno dato puede ser usado como un bit de paridad, si el formato del dato lo requiere 8 bits de datos y 1 bit de paridad.

Para transmitir el dato con un bit de paridad, el noveno bit debe ponerse en el bit TX9D del registro TXSTA, el cual debe estar en 1 ó 0 según la paridad antes de escribir los datos en el TXREG. Para recibir los datos con el bit de paridad, el noveno bit se almacena en el bit RX9D del registro RCSTA, para probar la correcta paridad antes de

leer el dato en el RCREG, asegurándose que el dato del noveno bit no se pierda si otra recepción ocurre. Algunas veces el formato del dato debe ser de 7 bits de datos y 1 bit de paridad. En este caso el USART puede ser usado en modo de 8 bits y el octavo bit de datos es usado como bit de paridad en lugar del noveno bit (Cf. [Garbutt, 2003]).

### 3.2.3. Datos de 9 bits para detectar direcciones

El noveno bit puede ser usado como un bit indicador de dirección si el formato del dato requiere 8 bits de datos con un noveno bit como dirección. Este es usado frecuentemente en la comunicación RS-485. Se transmite una dirección usando la dirección del noveno bit, el noveno bit debe ser puesto a 1 en el bit TX9D del registro TXSTA del PIC antes de escribir los otros 8 bits en el registro de trabajo. Para transmitir datos de 8 bits el noveno bit debe de ponerse a cero antes de escribir el dato en el registro de trabajo.

Para recibir una dirección usando el noveno bit de direccionamiento, el noveno bit debe ponerse en 1 el bit ADDEN del registro RCSTA para habilitar la detección de una dirección. Esto causa que el USART ignore todas las recepciones con el noveno bit en cero. Cuando ocurre una recepción, está será por una transmisión que tenga el noveno bit en 1, por lo tanto no es necesario mandar a probar el noveno bit. Regularmente la recepción de una dirección es leída del registro de recepción (RCREG) y comprobada. Sí la dirección es la correcta, el bit ADDEN debe ser puesto a 0 y el dato podrá ser recibido. Si la dirección no es la correcta la dirección es ignorada. El modo de 8 y 9 bits de datos y el noveno bit de direccionamiento provee una flexibilidad adicional. Con esta aplicación se puede tener una guía para implementar códigos que usen diferentes maneras de usar la comunicación asíncrona del USART (Cf. [Inc, 2001]).

En el presente trabajo de tesis, se utiliza el USART en el PIC 16F877 que contiene un noveno bit para detectar una señal como una dirección. Este es usado en la implementación de la norma RS-485, donde el noveno bit indica que el maestro está transmitiendo a los esclavos una dirección. Esto permite que el esclavo ignore automáticamente todas las direcciones hasta que la dirección le sea transmitida. El esclavo entonces puede comparar el identificador con su propia dirección, y si ésta es la misma, el esclavo se prepara para recibir los datos siguientes los cuales son enviados en 8 bits. Esto reduce el gasto de software de los esclavos y hace el software de los esclavos sea más fácil de implementar y que sea más eficiente (Cf. [Garbutt, 2003]).

Una ventaja de tener 9 datos de bits es que este puede ser usado como un indicador de dirección. Cada dispositivo conectado al bus serial tiene asignado una dirección específica y monitorea los datos por transmitir con el noveno bit a 1. Cuando el noveno bit esta puesto en 1, el software del PIC compara el dato recibido con su propia dirección.

Si la dirección es la correcta el software puede habilitar la recepción de los datos seguidos de la dirección, pero si la dirección no es la correcta, los datos siguientes son ignorados. Algunos PIC's comprueban la dirección del bit y pueden ignorar la transmisión que no tenga el noveno bit puesto a 1.

El método usado para la carga del dato del noveno bit depende de la aplicación en particular. Por ejemplo si este bit es usado con otro bit de parada, este puede ser puesto en 1 cuando se inicializa el USART y nunca se necesitará ser cambiado (Cf. [Inc, 2001]).

## 3.3. Convertidor Analógico-Digital

Los microcontroladores son muy eficientes al procesar los números digitales, pero estos no pueden manejar señales analógicas directamente. Un convertidor analógico digital, convierte un nivel de voltaje a un número digital. El microcontrolador puede entonces procesar el número digital eficientemente sobre el valor original del voltaje analógico. Por definición los números digitales no son números fraccionarios

El módulo de conversión Analógico/Digital dispone de cinco entradas para los dispositivos de 28 pines y ocho para los otros dispositivos de la familia. A través de la entrada analógica se aplica la señal analógica a un condensador de captura y retención (sample and hold) que después se introduce en el convertidor. El convertidor de aproximaciones sucesivas da como resultado una palabra de 10 bits. El convertidor Analógico/Digital tiene como característica especial de ser capaz de seguir trabajando mientras el dispositivo esté en el modo SLEEP (Cf. [Microchip, 2001]).

El convertidor Analógico/Digital es capaz de realizar una precisa conversión si la entrada de voltaje analógico se encuentra dentro del rango del convertidor. Si el voltaje está fuera del rango, el valor de conversión será incorrecto. La entrada del rango es puesta a 1 por una referencia de voltaje mínimo y máximo. En muchos casos, el voltaje de referencia mínimo y máximo son seleccionados de acuerdo al voltaje con el que se alimente el microcontrolador, y otras veces se utilizan referencias externas. Además, algunos dispositivos tienen un voltaje de referencia interno que puede ser usado. La fuente para ese voltaje de referencia se encuentra como una opción de configuración en el convertidor del microcontrolador. En esta parte existen restricciones en los niveles de referencia de voltaje, por ejemplo: la referencia de voltaje generalmente no debe ser menor que $V_{SS}$ (-0.3 volts) o más grande que $V_{DD}$ (4 a 7.5 volts). Debe existir por lo tanto una mínima diferencia que es requerida entre la referencia de voltaje mayor con la referencia de voltaje menor.

### 3.3.1. Cuantización

La salida del convertidor analógico digital es una representación cuantificada de la señal analógica original. El término de cuantización se refiere la subdivisión de rangos pequeños pero incrementos cuantificables. El rango de entrada total aceptable es dividido entre un número finito de regiones con un incremento determinado. El convertidor analógico digital determina la región apropiada que le asignará al voltaje de entrada. Por ejemplo, si tenemos un voltaje de entrada de 2.343 volts, éste sufre un redondeo. El resultado apropiado será asignado una valor digital de 87, porque 2.343 volts se encuentra asignado entre los límites de cuantización de 2.3 volts a 2.4 volts. Un voltaje de entrada entre 2.3 y 2.4 volts, le será asignado un valor digital de 87. El proceso de cuantización tiene el potencial de introducir una imprecisión conocida como error de cuantización, la cual puede ser vista como un error de redondeo.

Sobre las décimas de volt, el máximo error de cuantización en este caso será de 5 por ciento de volt, se debe de tomar en cuenta, que en consecuencia el mínimo error de cuantización para el convertidor analógico digital en el dispositivo del microcontrolador será de 500 micro volts. Esto se refiere, que el valor más pequeño no puede ser menor a un mili volt.

### 3.3.2. Resolución

La resolución se define como el número posibles de estados de salidas del convertidor analógico digital. Como anteriormente se mencionó el resultado es digital o un número entero, por lo tanto para un convertidor de 8 bits las posibilidades de salida serán: 0,1,2,3 hasta el número 255 como el máximo valor de salida. Para un convertidor de 10 bits tendremos 1024 diferentes posibilidades de salida, y para convertidor de 12 bits tendremos 4096 posibilidades de salida. Si tenemos una resolución alta el convertidor tendrá menos error de cuantización por que el rango es dividido en secciones más pequeñas. Este concepto es similar al proceso de redondeo de los números cercanos a las centésimas, teniendo un potencial de error menor que el redondeo cercano a las décimas (Cf. [Technology, 2001]).

#### Voltajes de referencia

Todo convertidor Analógico/Digital requiere voltajes de referencia que determinan el valor de mínima escala ($V_{ref-}$) y el de plena escala ($V_{ref+}$), de manera que la conversión de un valor de voltaje analógico $V_{in}$ en el rango de $V_{ref-}$ a $V_{ref+}$ producirá un valor equivalente binario D en el rango de 0 a 2n, sonde n es la resolución del convertidor (n=10).

Como la relación entre escalas es lineal, una regla de tres nos da la relación entre el voltaje analógico de entrada ($V_{in}$) y el valor digital (D) obtenido por el convertidor.

$$D/2n - 1 = (V_{in} - V_{ref-})/(V_{ref+} - V_{ref-})$$

Con la elección más común: $V_{ref+} = V_{DD} = 5$v, $V_{ref-} = V_{SS} = 0$v, y como n=10, obtenemos:

$$D = (1023/5)V_{in} = 204{,}6Vin$$

De donde se observa que cuando $V_{in}$ varía en todo su rango, desde 0 hasta 5v, el valor obtenido D varía también en todo su rango, de 0 a 1023. Si a la inversa, obtenemos un valor D y deseamos saber que voltaje representa, basta con despejar:

$$V_{in} = (5/1023)D = (0{,}004887585533)D$$

Como puede verse, la conversión del dato D al voltaje correspondiente requiere una multiplicación por un número fraccionario, para lo cual el PIC no posee instrucciones, si deseamos realizar esta multiplicación en el PIC debemos hacer un programa que multiplique números de punto fijo o de punto flotante (Cf. [Dumetri, 2002]).

La Figura 3.4 muestra el diagrama simplificado de un módulo del convertidor Analógico/Digital. Los pines de la entrada analógica están conectadas a las entradas de los multiplexores analógicos los cuales conectan a la selección del canal. El multiplexor analógico permite que las múltiples entradas se encuentren disponibles para la conversión. Este importante hacer notar que solo hay un convertidor analógico digital en el microcontrolador, y que solamente un canal puede ser seleccionado, y ser convertido al mismo tiempo. Normalmente el capacitor de retención esta conectado a la salida del multiplexor analógico. Cuando una conversión es iniciada, el multiplexor analógico desconecta todas las entradas del capacitor de retención, y aproxima sucesivamente al convertidor para llevar a cabo la conversión sobre el voltaje guardado por el capacitor de retención.

### 3.3.3. Tiempo de adquisición, conversión y espera

#### Tiempo de adquisición

Es el tiempo requerido para cargar y descargar el capacitor de retención hasta el fin de la conversión del convertidor analógico digital. El capacitor de retención debe tener el tiempo suficiente de establecer el voltaje de entrada analógica antes de ser iniciada la conversión actual Si el tiempo permitido no es el suficiente para la adquisición, la conversión resultará errónea.

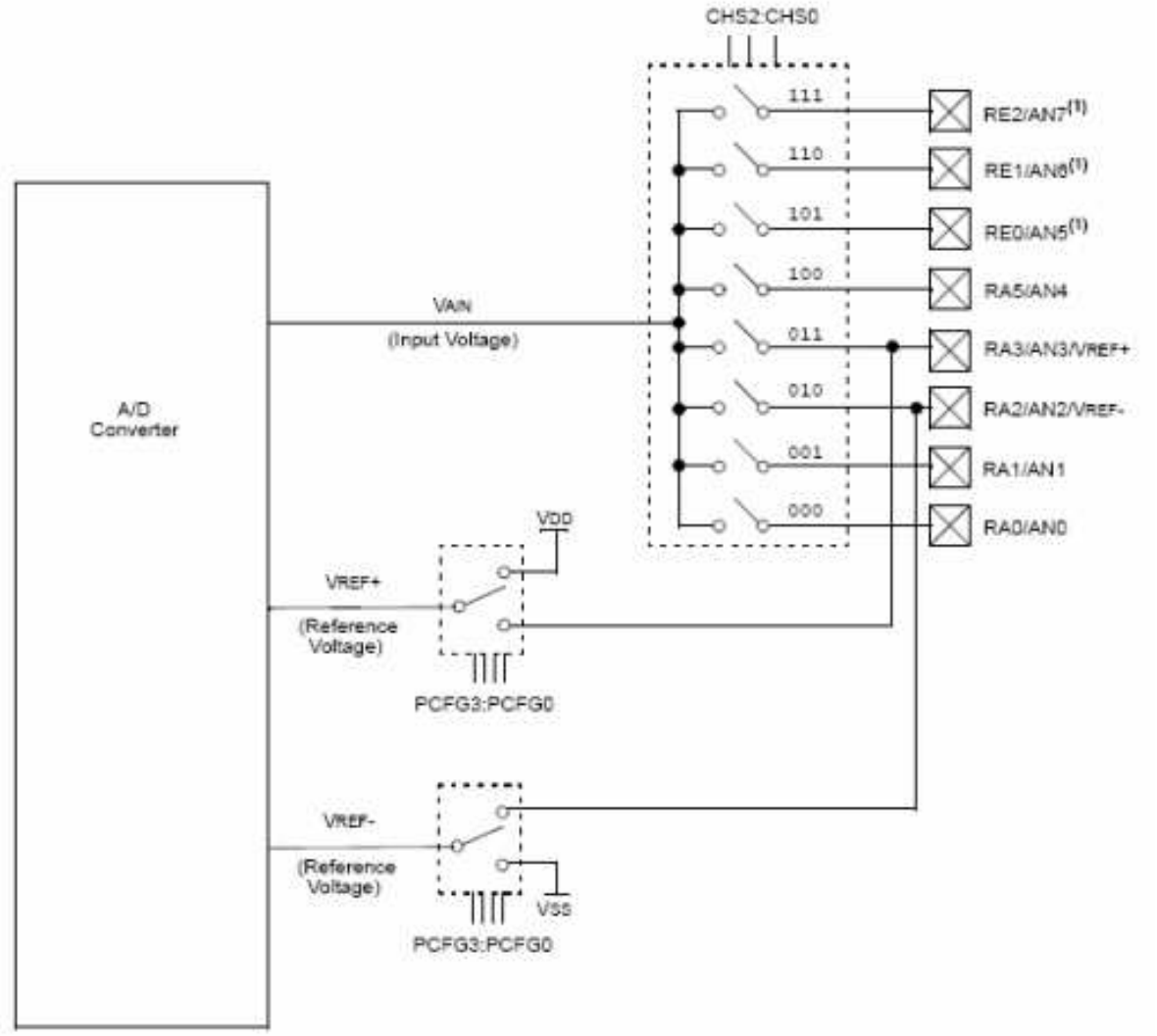



Figura 3.4: Diagrama del convertidor A/D.
*Fuente: [Microchip, 2001].*

El tiempo de adquisición requerido está basado sobre un número de factores, dos de los cuales son la impedancia del multiplexor analógico y la salida de la impedancia de la fuente analógica. Esta es generalmente de 10 k ohms para convertidores de 8 y 10 bits y 2.5 K ohms para convertidores de 12 bits.

**Tiempo de conversión**

Al tiempo requerido para la aproximación sucesiva de la conversión y la escritura del valor final en el registro de resultado se le conoce como tiempo de conversión. Este tiempo será el periodo del reloj analógico digital multiplicado por el número de bits de resolución del convertidor analógico-digital, además se requieren de 2 a 3 periodos de reloj para especificar el tiempo en el convertidor analógico-digital.

**Tiempo de espera de la adquisición**

Después de seleccionar el canal en el registro ADCON0, se debe permitir el suficiente tiempo de adquisición antes de iniciar la conversión. Esto se puede hacer colocando un tiempo de retardo en el programa.

### 3.3.4. Aproximación sucesiva

El convertidor de aproximaciones sucesivas ejecuta un bit en un tiempo, empezando con el bit más significativo y terminando con el bit menos significativo. El valor del bit más significativo es determinado por la señal de entrada en donde el valor de entrada esté por encima o por debajo de la mitad del rango de entrada válido. El próximo bit más significativo se determina, ya sea que la entrada esté por arriba o por debajo de la mitad del rango faltante, y así sucesivamente hasta determinar el bit menos significativo. Cada bit requiere un periodo de reloj del convertidor analógico digital para la conversión.

### 3.3.5. Configuración del convertidor Analógico/Digital

La entrada al canal y el reloj del convertidor analógico-digital son seleccionados, y el convertidor es habilitado, por las opciones de los registros ADCON0. La conversión no será iniciada en la misma instrucción en donde se habilitó el convertidor, ya que esto violaría los requerimientos del tiempo de adquisición.

**Inicia la conversión A/D**

Después de que ha pasado el tiempo de adquisición, la conversión puede empezar. Esto es indicado poniendo a 1 el bit GO en el registro ADCON0.

**Espera para la conversión de A/D finalizada**

Una vez que la conversión ha sido iniciada, se debe considerar el tiempo de conversión antes de leer el resultado. El estado de la conversión analógica-digital es indicada por el bit GO/DONE en el registro del ADCON0, el cual es automáticamente puesto en 0 cuando la conversión es realizada.

**Lectura del resultado**

Una vez que la conversión analógica-digital se ha completado, el resultado puede ser leído. El resultado de la conversión es automáticamente puesto en el registro del ADRES.

Para utilizar el convertidor se deberán seguir los siguientes pasos:

1. Configurar el módulo A/D:
   - Configurar los pines analógicos y los Voltajes de referencia $V_{ref-}$ y $V_{ref+}$, mediante el registro ADCON1 (9Fh) (y los correspondientes bits TRIS como entradas),
   - Seleccionar el canal de entrada a convertir mediante los bits CHS2:CHS0 del registro ADCON0 (1Fh),
   - Seleccionar el reloj de conversión mediante los bits ADCS1:ADCS2 (ADCON0<7:6>),
   - Energizar el convertidor mediante el bit ADON (ADCON0<0>).
2. Configurar interrupciones para el convertidor A/D (si se desea), para ello: limpiar ADIF y poner a 1 los registros ADIE, PEIE y GIE.
3. Esperar mientras transcurre el tiempo de adquisición (unos $20\mu$ seg).
4. Iniciar la conversión poniendo el segundo bit del ADCONO.
5. Esperar a que termine la conversión:
   - Por *poleo* (Polling): Consultando continuamente el bit GO/DONE (el cual es limpiado por el convertidor cuando la conversión se completa).
   - Por interrupciones: Cuando la conversión termina, la bandera ADIF se activa y esto genera una solicitud de interrupción, la cual deberá ser atendida por una rutina de atención a la interrupción diseñada para ello.
6. Leer el dato convertido D de los registros (ADRESH:ADRESL).

7. Para la siguiente conversión, esperar al menos 2 veces TAD (tiempo de conversión por bit).

En la Figura 3.5 se muestra el diagrama de tiempo, donde se observan los eventos que tienen lugar durante el proceso de una conversión Analógico/Digital.

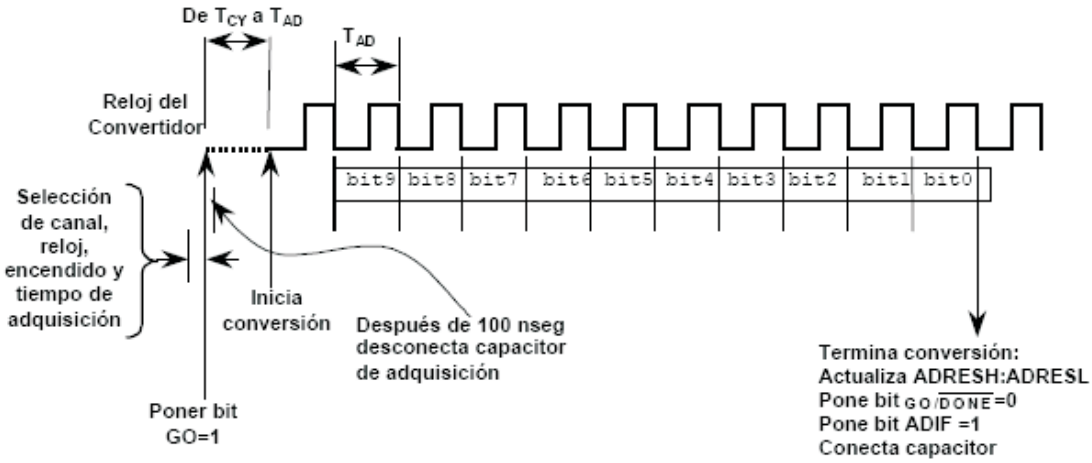



Figura 3.5: Proceso de conversión.
*Fuente: [Dumetri, 2002].*

### 3.3.6. Características eléctricas del convertidor

La Tabla 3.3.6 muestra algunas de las especificaciones más importantes, y son válidas para los PIC16F87X-04, PIC16F87X- 10, PIC16F87X-20, PIC16LF87X-04 (Cf. [Dumetri, 2002])

| Característica | Mínimo | Típico | Máximo |
|---|---|---|---|
| Vref+ - Vref- | 2 v | – | VDD + 0.3 v |
| Vref+ | $VDD - 2{,}5v$ | – | VDD+0.3 |
| Vref- | VSS-0.3 v | – | Vref+ - 2 v |
| voltaje analógico VAIN | VSS-0.3 v | – | Vref+ + 0.3 v |
| Imp. de la fuente ZAIN | – | – | 10 K ohms |
| I consumida por el A/D | – | 220 micro A | – |

Tabla 3.1: Especificaciones del convertidor

De los recursos contenidos en el PIC 16F877, quizá el convertidor Analógico/Digital sea el módulo que mas aplicaciones se le han dado. Sensores de temperatura, luz, humedad, humo, etc. son valores que se desean monitorear pero que necesitan ser traducidos para ser entendidos por el sistema de control, labor que realizan los convertidores Analógico/Digital. La tarea de conversión es siempre igual, sea cual sea el sensor utilizado. Por ello, teniendo en cuenta que lo que varía en un programa al cambiar de sensor será el tratamiento que se haga del valor una vez convertido (Cf. [Morcillo, 2000]).

## 3.4. Protocolo de enlace de datos

A continuación se analizará la estructura lógica que se realizó para poder lograr la comunicación entre los diferentes módulos esclavos y el coordinador general llamado maestro. Para realizar la comunicación entre los diferentes módulos, es preciso que se definan algunos términos:

- Una interfaz: periférico de comunicación, que incluye aspectos físicos (valores de tensión, cable o línea de datos, código de señales, tipo de cableado, etc.).
- Flujo o coordinación de la transmisión: identificación de los datos, solicitud de más datos o envío de señales. Ésta coordinación es independiente de la red por la que se ejecuta la transmisión (nivel OSI de la red) y precisa de un soporte físico (nivel OSI físico) para realizarse.

Según el modelo abierto OSI

- El **Nivel físico** se encarga de definir las tareas de procedimiento, características de las señales que se emplean para transmitir sobre el medio y todos los aspectos que están relacionados con la interfaz, con el medios de transmisión y con la aplicación que se esté ejecutando y precise de los servicios de comunicación. Por ejemplo: conectores, tipo de señales, codificación, etc.
- El **Nivel de enlace** de datos tiene como objetivo hacer que la comunicación sea fiable y eficiente.

### 3.4.1. Coordinación de la comunicación

La coordinación de la comunicación, desde el punto de vista de la iniciativa en la comunicación, se puede tener de 2 diferentes formas:

- **Centralizado:** en la cual un módulo actúa como módulo principal **maestro**, y el resto son pasivas, hasta que se les pregunta **secundarios, esclavos**. Además

el módulo principal hace la función de moderador, repartiendo los turnos a los módulos secundarios. Lo realiza en dos tipos de estrategias distintas, que son **poleo (polling)** y **selección**. La estrategia de poleo se utiliza por el módulo principal para descubrir si las estaciones tienen información que transmitir. El poleo se puede programar en una lista, o bien por una prueba desde el principal a las secundarias, cada cierto tiempo. La estrategia de selección se utiliza para escoger uno de los módulos secundarios, y realizar el intercambio de información.

- **Contienda:** las estaciones o módulos compiten entre sí por adueñarse de la iniciativa de la comunicación. La idea es escuchar el medio y si ninguna transmite, entonces tomar posesión del medio. Con un esquema para resolver las colisiones (más de una estación intenta transmitir en el mismo instante) y de prioridades, se soluciona los problemas habituales de este método de coordinación de la comunicación (Cf. [Morcillo, 2000]).

en el presente proyecto se utilizó el método de comunicación centralizado de selección, el cual nos permite tener una comunicación privada entre un maestro y un esclavo en específico, sin que otro esclavo interrumpa esta comunicación.

### 3.4.2. Diferentes principios de arbitraje

Cuando uno hace el estudio completo de un sistema de comunicación, a fin de tomar en cuenta todos los parámetros, el tiempo de latencia se define generalmente como la duración que existe entre el instante que indica el principio de la solicitud de una transmisión y el inicio real de la acción generada por ésta (Cf. [Dominique, 1996]).

Por el momento, en una forma más simple, definiremos el tiempo de latencia $t_{lat}$ de un mensaje como la duración que existe entre el instante que indica el inicio de la solicitud de una transmisión y el inicio real de ésta (Cf. [Dominique, 1996]).

Esta noción es difusa y muy cercana a las estadísticas, principalmente en los sistemas de tiempo real. La razón de esto es simple: solamente algunos mensajes específicos tienen realmente el tiempo de latencia garantizado y esto solamente en los picos de tráfico. Es por tanto necesario considerar dos tipos de mensajes:

- $R$, el número de mensajes de los cuales la latencia debe ser garantizada,
- $S$, los otros,

Y claro $M = R + S$, la totalidad de entre ellos (Cf. [Dominique, 1996]).

### Primeras consecuencias sobre la capacidad y longitud de una red

Sabiendo que la velocidad de propagación de las ondas electromagnéticas $v_{prop}$ es del orden de 200000 Km/s. en las líneas eléctricas y las fibras ópticas (ó bien que las ondas hacen alrededor de 5$\eta$s para recorrer 1 m ó bien recorren 200 m/$\mu$s) (Cf. [Etschberger, 2000]), en un sistema funcionando por contención de bits, un bit puede viajar de un extremo de la red al otro antes de ser detectado a su llegada. Ahora bien, puede suceder que algunos micros instantes antes de su llegada la otra estación, no habiéndolo todavía detectado, es decir no habiéndolo *visto* llegar a sus terminales, decide iniciar una emisión en la otra dirección. Esto resultaría en un choque frontal (Cf. [Etschberger, 2000]).

Si se designa a $t\_bus$ como el tiempo que toma la señal en recorrer la distancia máxima de la red, la suma global de los tiempos de ida y regreso debidos a la propagación de la señal sobre el bus es de:

$$2t\_bus = (2l)(v\_prop) \tag{3.1}$$

**Ejemplo:** Con l = 40 m, se obtiene $t_bus = 200\eta$s

Donde: l= longitud del bus, $v\_prop$= Velocidad de propagación.

Así que para que la estación que ha emitido el bit inicial esté apta a administrar los conflictos, los tiempos que debe durar el bit $t\_bit$ (bit time) deben ser más grande que el $t\_bus$. Además, para que esté completo es necesario, incluso obligatorio, tener en cuenta los tiempos necesarios, el periodo de muestreo y tratamiento del bit en la estación a la que llega (Cf. [Etschberger, 2000]).

### Control de desbordamiento de datos

Para el análisis inicial de control de la transmisión, entenderemos en principio que no existen errores en la transmisión. Esto nos permitirá entender los mecanismos de establecimiento, de solicitud y permiso de transmisión de datos.

La comunicación consiste en el intercambio de datos entre el emisor y el receptor. Además es necesario que el transmisor entregue los datos al receptor a una velocidad que permita al segundo procesarlos, sin desbordar su funcionamiento. El objetivo es conseguir una comunicación eficiente, con una recepción de los datos, en principio sin errores, y la mayor cantidad de datos que permita el medio de transmisión y el receptor Cf. [Morcillo, 2000]).

**Control de la transmisión: control de errores**

Cuando nos encontramos en el entorno de las comunicaciones, en el nivel de enlace de datos, los errores se entienden como sigue:

- **Error en la recepción:** durante la transmisión se han modificado uno o más bits de una trama, aunque ésta la haya recibido entera. Analizando las tramas, utilizando el campo de verificación, se detecta ese error.
- **Pérdida de trama:** de información o de reconocimiento, el receptor no detecta que ha llegado una trama, por tanto, hay un error. Por ejemplo, se ha perdido la secuencia de las tramas, sean datos o de reconocimiento.

Un criterio que se aplica, en ambos casos, es el de fijar un tiempo de espera para la recepción del reconocimiento de las tramas transmitidas. Transcurrido éste se vuelve a iniciar la cuenta del mismo y se solicita el reenvío al transmisor (Cf. [Morcillo, 2000]).

### 3.4.3. Conflictos en la transmisión

Al transmitir a través de un medio, las señales sufren una serie de deformaciones en sus diferentes características entre ellas:

1. **Atenuación:** Consiste en la disminución de la amplitud de la señal transmitida. Además tiene un efecto proporcional a la distancia de la comunicación. Si ésta es muy elevada llega un punto en el que la amplitud de la misma es irrelevante y no es recogida por los receptores.

   Para recuperar las características de la señal se utilizan elementos activos, que se insertan en el medio, cada cierta distancia. Cuando la señal que se va a transmitir es digital se utilizan los repetidores. Si la señal es analógica se utilizan los amplificadores.

2. **Distorsión:** Aunque la atenuación afecta por igual a todos los componentes frecuenciales de la señal que es transmitida, también se sabe que afecta más a las componentes de mayor frecuencia. Esto provoca el cambio de forma de la señal original, dado que sus componentes ya no aportan la misma amplitud (en términos de Fourier, han cambiado los valores de las componentes del desarrollo). Esta deformación se denomina distorsión por atenuación.

   Puede que no todas las señales lleguen al receptor en el mismo instante, lo que también origina una distorsión por retardo de grupo. Los niveles eléctricos correspondientes a los unos pueden verse afectados, introduciendo errores en la transmisión.

3. **Ruido:** Los tipos de ruido que vamos a comentar son:

   - Ruido térmico: Es un problema de todos los sistemas electrónicos. Afecta a todo el espectro de frecuencia, y es proporcional a la temperatura.

   - Ruido en una línea de transmisión: Es bastante inesperado, y se debe a interferencias ajenas a la propia línea. Para expresar la calidad de la señal respecto al ruido en un sistema de transmisión se emplea una expresión, desarrollada por el matemático Claude Shannon. Cuando está presente un nivel de ruido eléctrico en una señal, cabe esperar que el aumento del nivel de la señal permita la correcta recepción de los datos.

   - Diafonía: Es debida al acoplamiento no deseado entre líneas que transportan señales eléctricas que, se encuentran muy próximas.

### 3.4.4. Direccionamiento

En lo que respecta al direccionamiento se decidió con el fín de asegurar una buena elasticidad del sistema, de utilizar otro principio de direccionamiento, ya no tanto basado sobre las direcciones fuente y destino, si no más bien basado sobre el contenido del mensaje, esto implica dos cosas:

1. En principio, que un mensaje sea transmitido a todas las (otras) estaciones de la red. El término consagrado para tal principio es el de **difusión** (broadcast diffusion).
2. Enseguida, que el tratamiento de selección del mensaje transmitido sea entonces efectuado por un filtrado que se llama de aceptación, abordo de cada estación.

Para poder realizar lo anterior, el mensaje es etiquetado (posee un **label**) por un identificador -ID (i)-, que será entonces comparado en la lista de mensajes recibidos (o que se desea recibir) en cada estación (Cf. [Kiencke, 1994]).

De este hecho, todos los mensajes son recibidos sobre toda la extensión de la red y la consistencia de los datos es entonces garantizada en los sistemas de control distribuidos (Cf. [Kiencke, 1994]).

Para poder trabajar con la estrategia de coordinación de selección y direccionamiento, como se puede adivinar, los módulos precisan identificar a los módulos destinos de los mensajes. Se realiza asignándole una dirección a cada módulo. Habrá una dirección o identificador en cada módulo, sea el principal o secundario.

### 3.4.5. Buses de campo

Los buses de campo, están en el nivel más cercano a los dispositivos que integran las máquinas, el bus de campo no es sino una red industrial. El acceso a la información de los dispositivos de campo se realiza por medio de dispositivos que traducen o capturan la información. Buscan la reducción del cableado y la elevada velocidad en el plano bit. Es decir, en el plano del dispositivo (Cf. [Morcillo, 2000]).

Ante la necesidad de automatizar un proceso de media escala, pronto encontramos la necesidad de integrar un sistema de bus de campo. Además, sobre este proceso se requiere tener rápidamente parámetros de calidad y cantidad de producción. Esto conlleva la proliferación de sensores en distintos puntos y diferentes grados de control, todos coordinados.

El funcionamiento del ciclo de las comunicaciones en un sistema de control que incorpora un bus de campo debe ser interpretado como un programa que se ejecuta en segundo plano con respecto al sistema de control global de automatización. Es decir, el bus de campo surge como opción inmediata al plantearse tres consideraciones:

- Control integrado en diferentes niveles de información y diferentes procesos en una planta.
- Cantidad elevada de sensores y actuadores.
- Distancia de las órdenes de mandos superiores a varias decenas de metros.

#### Cableado

Desde la conexión de las interfaces o módulos de expansión, se conectan los sensores (entradas) y actuadores (salidas), bien analógicos y digitales. Ante este sistema, cada punto digital que hay que controlar conlleva:

- Dos cables de conexión de la señal de mando.
- Alimentación eléctrica

Es decir, cada sensor, cada señal analógica, requiere su cableado específico. Ante esto, los armarios de control se convierte en una maraña de cables que va y viene de las máquinas. En cambio, el bus de campo reduce ampliamente el cableado utilizado para las señales. El bus de campo introduce una sustancial variación. Con un par de cables (trenzado por ejemplo) se puede transmitir la señal digital. De este cable se realiza las conexiones necesarias. Cada dispositivo se conecta mediante una T: conector y derivador. Pero no cada elemento final, que puede ser conectados a terminales de grupos de dispositivos de entradas y salidas.

La alimentación eléctrica puede, incluso, incorporarse por el mismo cable, o precisar de otros dos hilos, separando las fuentes de señal y alimentación. La expansión del sistema consiste en ampliar las terminales e incorporar los dispositivos. Ya no hay que realizar el cableado desde el extremo de la máquina hasta la interfaz del sistema (Cf. [Morcillo, 2000]).

### 3.4.6. Tipos de dispositivos

Para poder realizar el protocolo de enlace es necesario definir el número de dispositivos o módulos a utilizar, la asignación de direcciones, el formato de los datos. En este caso la implementación de esta comunicación será para un invernadero de 1000 metros cuadrados, donde se implementarán 4 módulos secundarios, un principal, sensores y actuadores. La logística de configuración de sensores y actuadores se muestra en la Figura 3.6

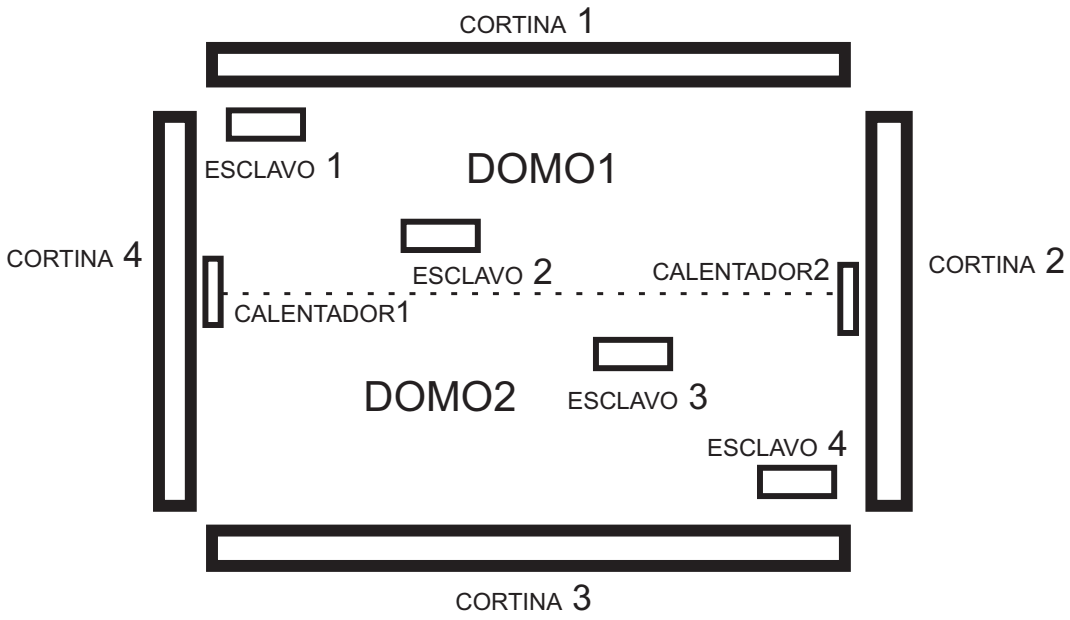



Figura 3.6: Configuración de los dispositivos.

Las variables analógicas que van a adquirir los sensores son:

- V1=Temperatura
- V2= Humedad
- V3= Radiación Solar

- V4= Dirección del viento
- V5= Velocidad del viento

Las acciones a realizar por los actuadores son:

- A1= Apertura de Cortinas (4 Cortinas)
- A2= Apertura de domos (2 domos)
- A3= Acción del calentador (2 calentadores)
- A4= Acción del riego.

Para controlar las variables y acciones se proponen 4 módulos llamados esclavos los cuales van a controlar:

- Esclavo 1: cortina1, cortina 4 y riego.
- Esclavo 2: calentador 2 y domo 1.
- Esclavo 3: calentador 1 y domo 2.
- Esclavo 4: Cortina 2 y Cortina 3.
- Además de que cada módulo está midiendo las variables de humedad y temperatura.

### 3.4.7. Logística del protocolo

Las direcciones que van a ser asignadas a los módulos esclavos son de 9 bits ya que se utiliza la transmisión de 9 bits para detectar direcciones en el PIC (véase el apartado 3.2.3), en la Tabla 3.2 son mostradas los identificadores usados para los esclavos.

| No. Esclavo | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| **Esclavo 1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| **Esclavo 2** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **Esclavo 3** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **1** |
| **Esclavo 4** | **1** | **0** | **0** | **0** | **0** | **1** | **0** | **0** |

Tabla 3.2: Asignación de Direcciones

A continuación se presenta el formato de petición de datos de 8 bits que hace el módulo maestro hacia los esclavos. Para poder tener un control al momento de programar

los identificadores de 8 bits que hace el módulo maestro hacia los esclavos se siguió el siguiente criterio para determinar el ID de las acciones y peticiones.

**Cortinas**

Las cortinas pueden tener 4 posiciones:

1. Que van desde 0 que indica que la cortina está cerrada, 33 , 66 y finalmente 100 por ciento que equivale a la apertura total de la cortina.
2. El bit 0 y el bit 1 se utilizan para saber la posición en que se encuentra la cortina, el bit 2 y el bit 3, indican el numero de esclavo, el bit 4 y 5 determinan que cortina se va a accionar, y el bit 6 y el bit 7 se programan para tener en total los 8 bits, sin repetición con las demás estructuras.

Esto se muestra en la Tabla 3.3.

| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
|---|---|---|---|---|---|---|---|
| **1** | **0** | **No. de** | **Cortina** | **No. de** | **Esclavo** | **Posición** | **Cortina** |

Tabla 3.3: Estructura del ID de las cortinas.

**Calentadores**

Con los calentadores el número de ID va a depender del tiempo de encendido del calentador. Para ello se ocupa los bits 0, 1, 2, 3 y 4. El bit 5 determina cual de los 2 calentador es encendido, en este caso si es 0 indica que está encendido el calentador número 1 o si es 1 indica que esta encendido el calentador número 2.
Esto se muestra en Tabla 3.4.

| **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
|---|---|---|---|---|---|---|---|
| **1** | **1** | **No. Calent.** | **Tiempo** | **de** | **Encendido** | **del** | **Calent.** |

Tabla 3.4: Estructura del ID de los calentadores.

**Domos**

Para los domos solo existen 2 condiciones o están abiertos o se encuentran cerrados. Para lo cual se ocupa el bit 0 y el bit 1 para determinar la apertura ó cierre de los domos, el bit 2 y el bit 3 determina el número de esclavo, el bit 4 y 5 indican si está abierto ó cerrado el domo 1 ó el domo 2. Esto se muestra en la Tabla 3.5.

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | No. de | Domo | No. | Esclavo | Abierto | ó Cerrado |

Tabla 3.5: Estructura del ID de los Domos.

**Temperatura**

Para determinar el ID de la temperatura se utiliza 2 ID diferentes, ya que en los 4 esclavos se esta midiendo la temperatura, para determinar el número de esclavo al que se refiere, se utilizan los bits 3, 4 y 5. Y se ocupan 2 ID para evitar que otra petición ó alguna acción pudiera coincidir con el ID de la temperatura si se tomara solo una ID. El primer ID es ocupado por el esclavo 1 y el esclavo 4, y el segundo ID lo ocupa el esclavo 2 y el esclavo 3. Esto se muestra en la Tabla 3.6

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Número | de | Esclavos | 0 | 0 | 1 |
| 0 | 1 | Número | de | Esclavos | 0 | 0 | 1 |

Tabla 3.6: Estructura del ID de la Temperatura.

**Humedad**

Para determinar el ID de la humedad se utilizó 2 identificadores, al igual como ocurre con la temperatura, el primer ID es para los esclavos 1 y 4, y el segundo ID es para los esclavos 2 y 3. Esto se muestra en la Tabla 3.7

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Número | de | Esclavos | 0 | 1 | 1 |
| 0 | 1 | Número | de | Esclavos | 0 | 1 | 1 |

Tabla 3.7: Estructura del ID de la Humedad.

Se realizó toda esta estructura de bits para prever, que al realizar este protocolo, si se tenía algún error en la transmisión o recepción del dato, se facilitara encontrar y resolver el problema de forma más rápida y fácil. Ya que esto al comparar el código, se puede definir donde ocurrió el error, si fue en la transmisión o en la recepción de la comunicación del dato, al igual que al comprobar efectivamente la petición o acción que envió el maestro a un esclavo en específico, sea recibida exclusivamente por el esclavo

solicitado, sin que otro esclavo reciba este dato, el cual no le corresponde.

A continuación se presentan los identificadores de las acciones y peticiones ocupados para cada esclavo (Tabla 3.8).

| Acción | Apertura | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| **Riego** | – | **0** | **1** | **X** | **X** | **X** | **X** | **X** | **X** |
| **Temp** | – | **1** | **1** | **0** | **0** | **1** | **0** | **0** | **1** |
| **Hum** | – | **1** | **1** | **0** | **0** | **1** | **0** | **1** | **1** |
| **Cort 1** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| – | **33** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |
| – | **66** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| – | **100** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **1** |
| **Cort 4** | **0** | **1** | **0** | **1** | **1** | **0** | **0** | **0** | **0** |
| – | **33** | **1** | **0** | **1** | **1** | **0** | **0** | **0** | **1** |
| – | **66** | **1** | **0** | **1** | **1** | **0** | **0** | **1** | **0** |
| – | **100** | **1** | **0** | **1** | **1** | **0** | **0** | **1** | **1** |

Tabla 3.8: Esclavo 1

Para determinar los valores de Bit 0 al Bit 5 para accionar el riego, va a depender del tiempo en que se tenga prendida la bomba del riego, con los 6 bits se puede definir 63 intervalos de tiempo diferentes para nuestro riego (Tabla 3.9- 3.10 ).

| Acción | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| **Temp** | **0** | **1** | **0** | **1** | **0** | **0** | **0** | **1** |
| **Hum** | **0** | **1** | **0** | **1** | **0** | **0** | **1** | **1** |
| **Calent 2** | **1** | **1** | **1** | **X** | **X** | **X** | **X** | **X** |
| **Domo 1** | **0** | **0** | **0** | **1** | **0** | **1** | **0** | **X** |

Tabla 3.9: Esclavo 2

Para determinar los Bits 4, 3, 2, 1, 0 de los calentadores 1 y 2, va a depender del tiempo que esté encendido, este puede tenere 15 intervalos de tiempo diferentes. Para el domo 1 y 2 el Bit 0 nos indica si está abierto o cerrado el domo, si es 1 se encuentra abierto y si es 0 estará cerrado (Tabla 3.11).

Todos estos dígitos binarios de dirección, confirmación y acción fueron designados para el desarrollo del protocolo bajo una logística de trabajo que sobre todo evita la recepción de datos que no corresponden a cada esclavo y por la tanto la transmisión de datos que no le son pedidos por el maestro.

| **Acción** | **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
|---|---|---|---|---|---|---|---|---|
| **Temp** | **0** | **1** | **0** | **1** | **1** | **0** | **0** | **1** |
| **Hum** | **0** | **1** | **0** | **1** | **1** | **0** | **1** | **1** |
| **Calent 1** | **1** | **1** | **0** | **X** | **X** | **X** | **X** | **X** |
| **Domo 2** | **0** | **0** | **1** | **0** | **0** | **1** | **1** | **X** |

Tabla 3.10: Esclavo 3

| **Acción** | **Apertura** | **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
|---|---|---|---|---|---|---|---|---|---|
| **Temp** | – | **1** | **1** | **1** | **0** | **0** | **0** | **0** | **1** |
| **Hum** | – | **1** | **1** | **1** | **0** | **0** | **0** | **1** | **1** |
| **Cort 2** | **0** | **1** | **0** | **0** | **1** | **0** | **1** | **0** | **0** |
| – | **33** | **1** | **0** | **0** | **1** | **0** | **1** | **0** | **1** |
| – | **66** | **1** | **0** | **0** | **1** | **0** | **1** | **1** | **0** |
| – | **100** | **1** | **0** | **0** | **1** | **0** | **1** | **1** | **1** |
| **Cort 3** | **0** | **1** | **0** | **1** | **0** | **0** | **1** | **0** | **0** |
| – | **33** | **1** | **0** | **1** | **0** | **0** | **1** | **0** | **1** |
| – | **66** | **1** | **0** | **1** | **0** | **0** | **1** | **1** | **0** |
| – | **100** | **1** | **0** | **1** | **0** | **0** | **1** | **1** | **1** |

Tabla 3.11: Esclavo 4

# Capítulo 4

# Esquema de solución

**Introducción**

En este Capítulo se analiza la programación realizada para implementar el protocolo de comunicación serial distribuida de forma más detallada. Así como también los conceptos básicos (más detalladamente que en los capítulos anteriores) que sirven para entender dicho protocolo.

El esquema de solución consiste principalmente de 5 sistemas mínimos y una computadora personal (PC). Dentro de los sistemas mínimos tenemos al sistema mínimo principal denominado Maestro y a los cuatro restantes denominados Esclavos. El Maestro es el encargado de enviar las señales provenientes de la PC al esclavo indicado. De igual manera, recibe las señales provenientes de los Esclavos y las envía a la PC. Dicha comunicación se realiza bajo la norma RS485, usando el módulo USART que contiene el microcontrolador.

## 4.1. Comunicación serie asíncrona

En la comunicación serial se hace necesario establecer métodos de sincronización para evitar la interpretación errónea de los datos transmitidos. En el caso específico de la transmisión serial asíncrona, que es empleada en el protocolo, se encuentra una serie de bits que siempre están presentes (bit de inicio, bit de parada), y otros que solamente aparecen en función de la información (bits de datos). Esto se puede observar en la Figura 4.1.

Tales bits que conforman a las tramas en la transmisión serial son:

- El bit de inicio (START) es el que marca el inicio de la transmisión y está siempre presente. Siempre es cero.

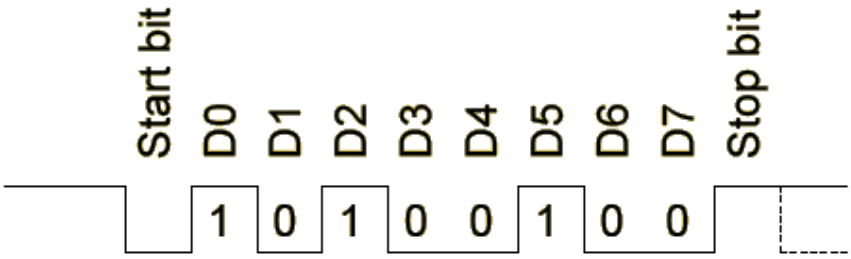



Figura 4.1: Transmisión asíncrona de 8 bits.
*Fuente: [Garbutt, 2003].*

- Los bits de Datos (DATOS) pueden variar y son los que transportan la información.
- Los bits de Parada (STOP) indican que se ha terminado la comunicación. Están siempre a uno y como mínimo debe de haber uno.
- El bit de de Multiprocesador (MULTI) sólo está presente en sistemas con la extensión de multiprocesador, donde hay mensajes destinados a un conjunto de receptores o bien direcciones.
- El bit de Paridad ( PARIDAD) es un bit de detección de error, no está siempre presente y admite las configuración de paridad par e impar.

### 4.1.1. Puerto Serie USART

EL USART es uno de los puertos de comunicación serial que contiene el PIC16F877. El Puerto USART puede configurarse de los siguientes modos:

- Modo Asíncrono (full-duplex)
- Modo Síncrono Maestro (Half Duplex)
- Modo Síncrono Esclavo (Half Duplex).

De igual modo, el puerto USART tiene la capacidad multiprocesador, por lo tanto puede transmitir 9 bits para detectar direcciones. Para conocer los registros que deben modificarse para utilizar cualquiera de estos modos consultar la Sección 2.2.7.

**Modo Asíncrono.**

En este modo el USART usa un formato estándar NRZ (No retorno a Z) asíncrono, el cual para la sincronización usa: 1 bit de inicio (I), 8 o 9 bits de datos y 1 bit de paro (P). Mientras no se están transmitiendo datos el USART envía continuamente un bit de marca. Cada dato es transmitido y recibido comenzando por el bit menos significativo (LSB). El hardware no maneja bit de Paridad, pero el noveno bit puede ser usado para este fin y manejado por software.

El módulo de transmisión asíncrono consiste de los siguientes elementos:

- Generador de frecuencia de transmisión (BRG)

  El BGR es usado tanto en el modo síncrono como en el modo asíncrono. Consiste de un contador/divisor de frecuencia de 8 bits controlado por el registro SPBRG. Dicha frecuencia se calcula aplicando una fórmula de las dos disponibles, dependiendo del bit que se cargue al BRGH. Si el BRGH=1, indica que se utilizará una alta velocidad:

  $$BaudRate = F_{osc}/(16(X+1)) \tag{4.1}$$

  En cambio, si el BRGH=0, se selecciona una baja velocidad de transmisión:

  $$BaudRate = F_{osc}/(64(X+1)) \tag{4.2}$$

  Debido a que el divisor es de 8 bits, no se puede tener cualquier velocidad de transmisión deseada, ya que X se deberá redondear al entero más cercano.

- Circuito de Muestreo

  El dato en el pin RC7 (pin de recepción) es muestreado tres veces por un circuito de mayoría, con la finalidad de determinar con seguridad si el dato recibido se trata de un nivel bajo o un nivel alto.

- Transmisor Asíncrono

  El diagrama del transmisor asíncrono se muestra en la Figura 4.2. El transmisor asíncrono tiene como componente principal al registro de corrimiento TSR (Transmit shift register). El TSR obtiene el dato del Buffer de lectura/escritura TXREG, quien a su vez lo obtiene mediante software. Una vez que el TXREG se queda vacío se activa la bandera TXIF (PIR1,4), es decir TXIF=1, siempre y

cuando se hayan habilitado las interrupciones globales y se encuentre habilitado el TXIE (PIR4).

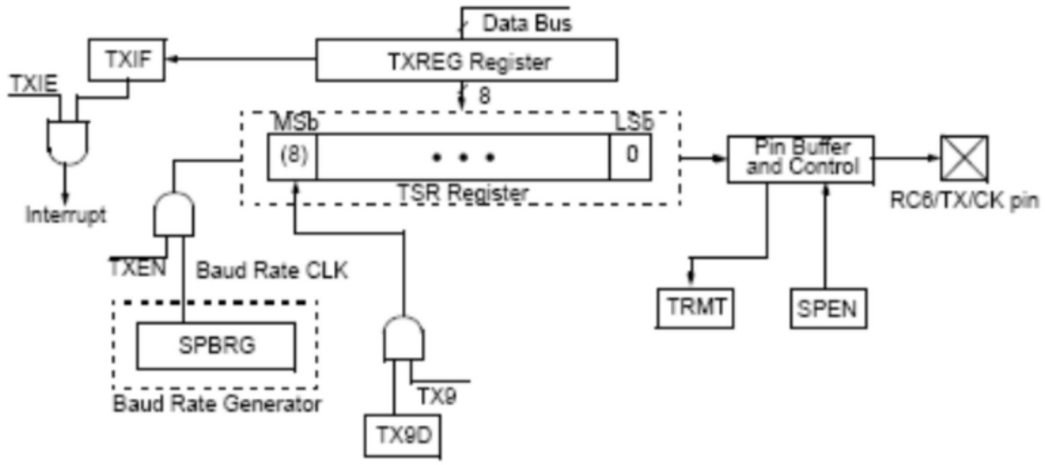



Figura 4.2: Diagrama a bloques del transmisor USART.
*Fuente: [Microchip, 2001].*

La ventaja de tener dos registros el TXREG y el TSR, es que se permite que un nuevo dato pueda ser escrito en el TXREG mientras que el dato anterior está siendo enviado.

La transmisión se habilita cargando a 1 el TXEN, mientras ésto no ocurra el pin TX se mantiene en alta impedancia. Si el TXEN se deshabilita durante la transmisión ésta sera abortada y se reinicializará el transmisor.

- Receptor Asíncrono

El diagrama del receptor asíncrono se muestra en la Figura 4.3 Al igual que en la transmisión, es necesario habilitar la recepción. Esto se realiza cargando a 1 el bit CREN del registro RCSTA.

El componente principal del receptor es el registro RSR (Receive Shift Register). Este registro no es accesible por software, pero cuando se recibe el bit de Stop (dato completo) automáticamente el dato se transfiere al registro RCREG. Dicho registro puede contener hasta dos datos, debido a que es un buffer doble. Si las dos posiciones están llenas y se detecta el bit de parada de un tercer dato, se destruirá el primer dato, activándose así la bandera de error de sobreescritura OERR. La

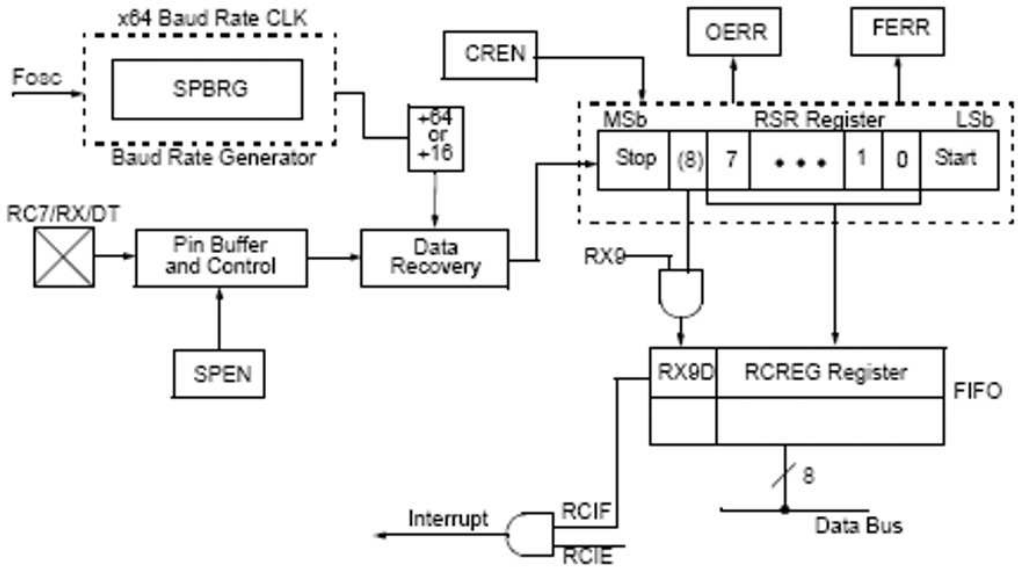



Figura 4.3: Diagrama a bloques del receptor USART.
*Fuente: [Microchip, 2001].*

única manera de limpiar este error es reinicializar el módulo de recepción, es decir, el CREN se deberá poner en cero. Para evitar dicho error, es necesario hacer dos lecturas consecutivas al RCREG.

Al transferirse el dato al RCREG, se cargará automáticamente un 1 a la bandera de recepción RCIF (PIR1, 5), siempre y cuando se halle habilitado el bit RCIE (PIE1, 5). La bandera RCIF sólo puede ser de lectura debido a que es inicializada mediante hardware.

## 4.2. Comunicación 8-9 bits

En el caso de la transmisión asíncrona de 9 bits de datos, el noveno bit es colocado después de los 8 bits de datos y es seguido por el bit de parada (ver Figura 4.4), lo que facilita la implementación del formato de datos RS-232 que requieren bit de paridad o dos bits de parada.

La ventaja de contar con 9 bits de datos es que puede ser usado como indicador de direcciones. Esto es comúnmente utilizado en el protocolo RS-485. A cada dispositivo colocado en un bus serial se le asigna una dirección específica. Cuando el noveno bit es activado, el software del PIC compara el dato recibido con su propia dirección. Si el

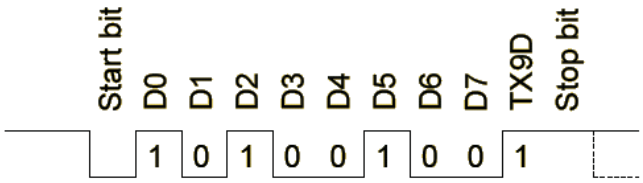



Figura 4.4: Transmisión asíncrona de 9 bits.
*Fuente: [Garbutt, 2003].*

dato recibido y la dirección son iguales, se habilita la recepción de los siguientes bits de datos. Si no son iguales los siguientes bits de datos son ignorados por el PIC.

## 4.3. LabVIEW

LabVIEW es un ambiente de desarrollo gráfico con funciones integradas para realizar adquisición de datos, control de instrumentos, análisis de mediciones y presentaciones de datos (Cf. [VISA, 2000]).

Los programas hechos en LabVIEW son denominados VIs (Virtual Instruments) debido a que en apariencia y operación imitan a instrumentos físicos, como multímetros u osciloscopios.

En LabVIEW, se pueden construir interfaces de usuario, paneles con controles (botones,*dials*, *knops*) e indicadores (gráficas, LEDs, displays, etc). Igualmente se puede usar LabVIEW para comunicar distintos dispositivos como tarjetas de adquisición, dispositivos de control, así como también instrumentos GPIB, PXI, VXI, RS-232, RS-485, etc.

### 4.3.1. LabVIEW VISA

El módulo NI-VISA implementado en LabVIEW es un lenguaje de entrada/salida de datos que se utiliza en la programación de instrumentación. VISA es es un API*de alto nivel que hace llamadas a los controladores de bajo nivel. La Figura 4.5 muestra la jerarquía con la que opera NI-VISA:

VISA tiene la capacidad de controlar instrumentos GPIB,VXI y seriales haciendo las llamadas apropiadas dependiendo del tipo de instrumento que se va a usar. Una de sus ventajas es que utiliza las mismas operaciones para la comunicación con los instrumentos sin importar el tipo de interfaz. Es decir, si se quiere escribir una cadena

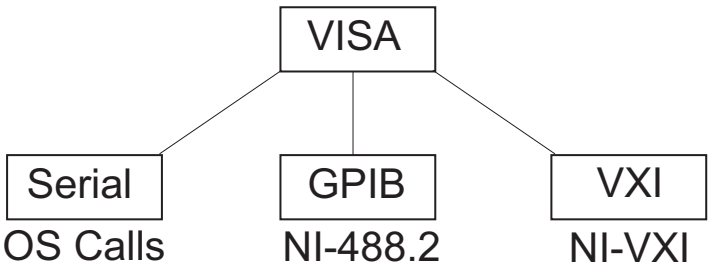



Figura 4.5: Jerarquía NI-VISA.
*Fuente: [VISA, 2000].*

de caracteres en ASCII, VISA lo hace de la misma forma si se trata de instrumentos seriales, GPIB o VXI. Por lo tanto, VISA provee dependencia de la interfaz.

### Programación de NI-VISA

VISA es un lenguaje orientado a objetos, por tanto, los objetos más importantes en el lenguaje VISA son conocidos como recursos. En la terminología de la programación orientada a objetos, las funciones que pueden ser usadas con objetos se les conoce como operaciones. Igualmente los objetos tienen variables asociadas a él, que contienen información del objeto mismo. En el lenguaje VISA estas variables son conocidas como atributos. En la Figura 4.6 se muestra el diagrama de la estructura interna del lenguaje VISA:

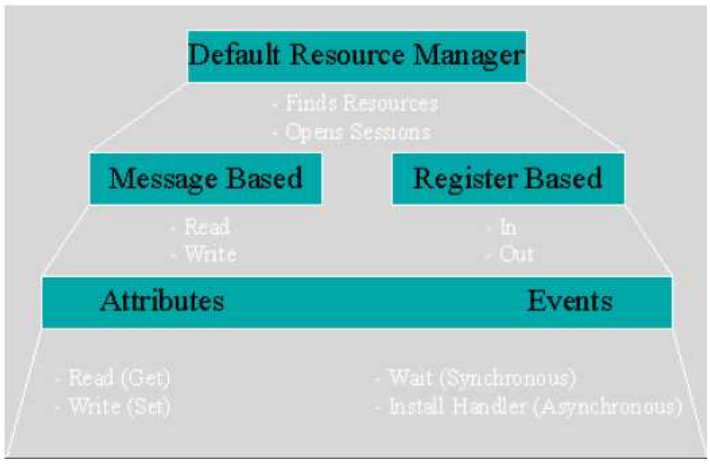



Figura 4.6: Estructura interna NI-VISA.
*Fuente: [VISA, 2000].*

Como se puede observar lo que tiene mayor jerarquía en el lenguaje VISA es el administrador de recursos (default resource manager) ya que es el que se encarga de encontrar recursos y abrir sesiones. Las operaciones más comunes para la comunicación de instrumentos basados en cadenas de caracteres son las de lectura y escritura.

El VI *VISA Find Resources* es el encargado de llevar a cabo la función de buscar los recursos disponibles en el sistema, además de ser el punto de partida de cualquier programa que utilice VISA (Figura 4.7). Es decir, se utiliza para determinar si todos los recursos necesarios para que una aplicación funcione correctamente están disponibles en el sistema.

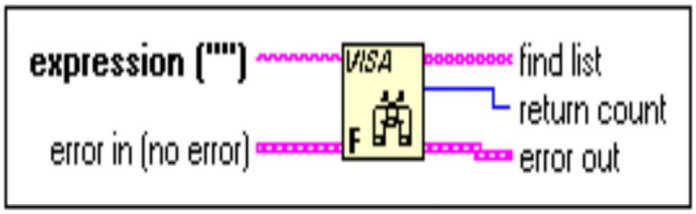



Figura 4.7: VISA find resources.
*Fuente: [VISA, 2000].*

La única entrada necesaria para que funcione el VI es la entrada llamada *expression*. Ésta determina el tipo de recurso que el VI va a regresar. Los resultados obtenidos se muestran en *find list*.

La otra función del administrador de recursos, consiste en abrir la sesión necesaria para que la comunicación se pueda llevar a cabo. Esto se realiza mediante el VI OPEN (ver Figura 4.8).

Los parámetros más importantes para el VI VISA OPEN son:

- VISA Open timeout: especifica el periodo máximo en milisegundos que el VISA OPEN espera antes de mostrar un error.
- VISA resource name: indica el nombre del recurso que se va a emplear.
- error in: describe los errores que ocurren antes de que corra el VI.

Una sesión abierta por VISA utiliza recursos del sistema. Así para finalizar correctamente una sesión hecha por VISA es necesario cerrar todos los recursos utilizados. Para ello existe el Instrumento Virtual VISA close (Figura 4.9).

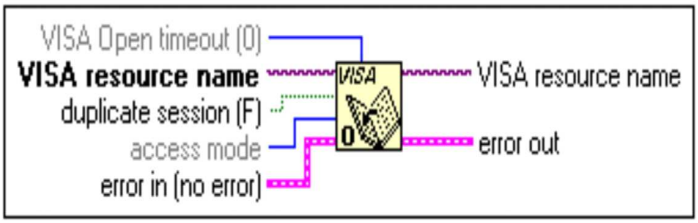



Figura 4.8: VISA open.
*Fuente: [VISA, 2000].*

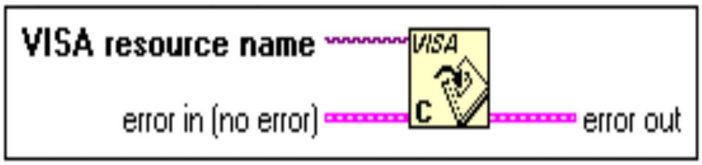



Figura 4.9: VISA close.
*Fuente: [VISA, 2000].*

En este VI los parámetros importantes son:

- VISA resource name: indica el nombre del recurso que se quiere cerrar.
- error in: describe los errores que ocurren antes de que corra el VI.

Para el desarrollo del protocolo además de utilizar los dos instrumentos virtuales mencionados anteriormente, fue necesario utilizar tres instrumentos virtuales (VIs) de VISA para la escritura, lectura y configuración del puerto serial (ver Figura 4.10). Tales VI's son:

**VISA WRITE:** Escribe un dato en el dispositivo o interfaz especificada.

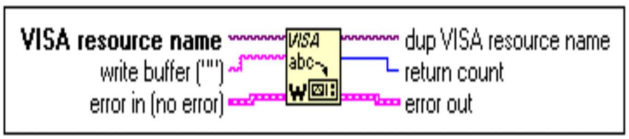



Figura 4.10: VISA write.
*Fuente: [VISA, 2000].*

- VISA resource name: indica el nombre del recurso donde se va a escribir el dato.
- write buffer: contiene el dato que se va a escribir en el dispositivo o interfaz.
- error in: describe los errores que ocurren antes de que corra el VI.

**VISA READ:** Lee el dato del dispositivo o interfaz especificada (Figura 4.11).

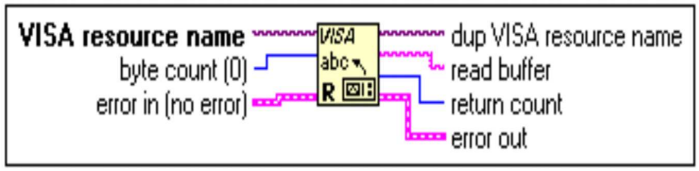



Figura 4.11: VISA read.
*Fuente: [VISA, 2000].*

- VISA resource name: indica el nombre del recurso de donde se va a leer el dato.
- read buffer: contiene el dato que fue leído del dispositivo o interfaz.
- error in: describe los errores que ocurren antes de que corra el VI.

**VISA CONFIGURE SERIAL PORT:** inicializa el puerto serial especificado con los parámetros especificados (Figura 4.12).

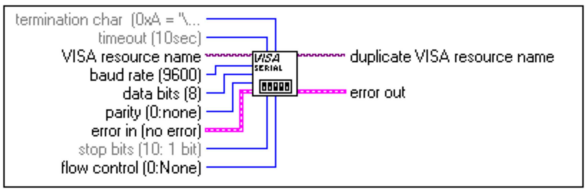



Figura 4.12: VISA configure serial port.
*Fuente: [VISA, 2000].*

- VISA resource name: especifica el puerto serial a configurar.
- baud rate: indica la tasa de transmisión.
- data bits: número de bits del dato entrante. Está entre 5 y 8.
- parity: indica la paridad.
- stop bits: número de bits de parada para indicar el fin de la trama.
- error in: describe los errores que ocurren antes de que corra el VI.

## 4.4. Esquema general de solución

La solución planteada en éste trabajo de tesis, consiste básicamente de los siguientes elementos (véase Figura 4.13):

- 4 Sistemas Mínimos Esclavos
- 1 Sistema Mínimo Maestro
- 1 Computadora Personal

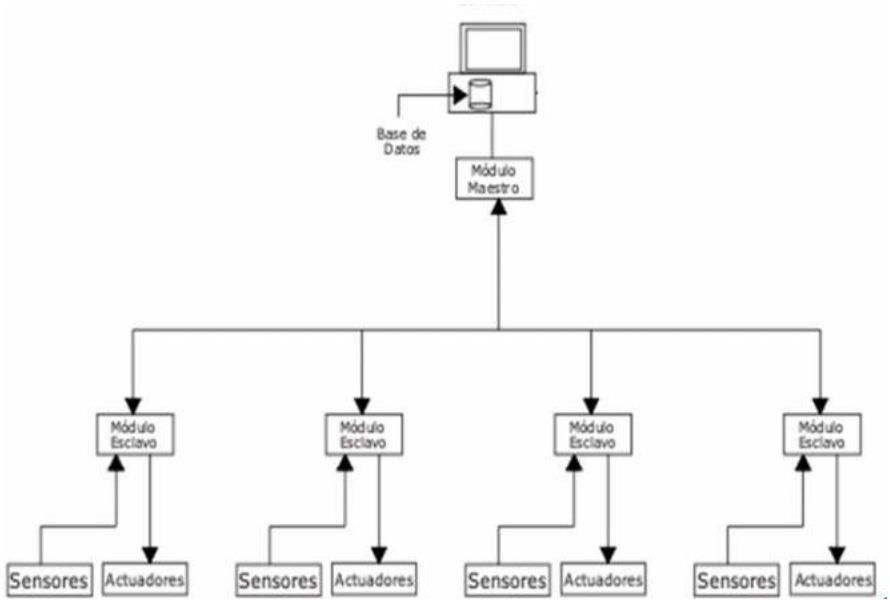



Figura 4.13: Arquitectura del sistema.

Los Sistemas Mínimos Esclavos tienen a su cargo tres funciones principales. La primera consiste en adquirir datos provenientes de los sensores conectados a él. Tales sensores son los de humedad, temperatura, radiación solar y dirección de viento. La segunda función consiste en enviar las señales adquiridas al Sistema Mínimo Maestro. Y la tercera consiste en enviar señales de control a los actuadores, tales como lo motores, bombas, etc.

Por su parte el Sistema Mínimo Maestro tiene encomendada la tarea principal dentro del protocolo, ya que es el encargado de enviar las señales de acciones y peticiones al esclavo correspondiente dependiendo de las diferentes señales que le mande la PC. Es decir, si el usuario quiere conocer la temperatura que está midiendo el esclavo 1, la PC le envía un byte al maestro con dicha indicación. El Maestro por su parte, identifica a que esclavo corresponde el byte recibido e inmediatamente después envía la dirección (ID) del esclavo en 9 bits, en este caso al esclavo 1. Todos los Esclavos reciben dicha dirección, pero solamente el esclavo 1 la toma, para posteriormente enviarle el último valor que adquirió de la temperatura.

La computadora personal (PC) tiene la función principal de servir como interfaz (desarrollada en LabVIEW) para el usuario. En esta interfaz se cuenta con los instrumentos virtuales y algoritmos adecuados para la supervisión y control de todo el sistema (véase Figura 4.14).

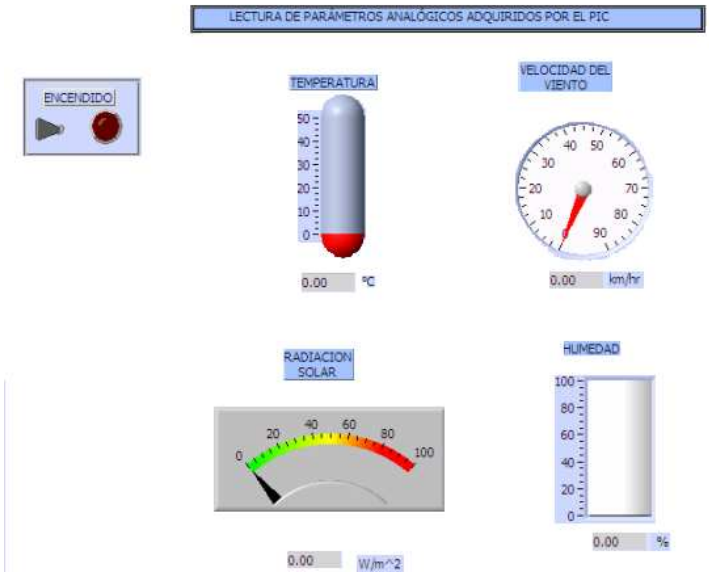



Figura 4.14: Instrumentos virtuales.

Igualmente la PC sirve como base de datos, en donde se guardan los datos obtenidos de las mediciones de los sensores y la acción de los actuadores en una hoja de cálculo de Excel.

En la siguiente sección se explica con detalle cómo se configuró el PIC16F877 para realizar las tareas asignadas a cada elemento del protocolo.

## 4.5. Protocolo de comunicación

Como se mencionó en la Sección 4.4 el protocolo está divido en dos partes: la comunicación Maestro-Esclavos, realizada bajo la norma de comunicación RS-485 y la comunicación Maestro-PC, realizada bajo la norma de comunicación RS-232.

En el protocolo, lo primero que se realiza es la inicialización de los Esclavos, en la cual se verifica si no existe error en la comunicación. Consiste en enviar del Maestro a cada uno de los Esclavos conectados, su ID de dirección en 9 bits y recibir un número de 8 bits de confirmación. Si por algún motivo un Esclavo no enviara el número de confirmación al Maestro, éste detecta el error y en adelante no lo toma en cuenta para realizar las acciones o peticiones hasta que sea corregido el error. Las rutinas de configuración del PIC se muestran en la Figura 4.15 y la Figura 4.16. En ellas podemos observar la forma en que el transmisor y el receptor asíncrono del USART fueron configurados para recibir y enviar tanto 8 como 9 bits.

```
#include pic.h
#include comserial.h

void envia9Bits() // en esta parte del protocolo se configura el transmisor asíncrono del puerto USART para la transmisión de 9 bits.

// Configuración del registro TXSTA para la transmisión

TX9=1; // Habilita transmisión de 9 bits
TXEN=1; // Habilita transmisión
SYNC=0; // Selecciona transmisión asíncrona
BRGH=1; // Alta velocidad
SPBRG=0x81; // 9600 baudios de velocidad de transmisión
TX9D=1;// Coloca un 1 para que sea transmitido como el noveno bit
SPEN=1; // Habilita al puerto serial.
TXIE=0; // Habilita interrupción por transmisión
RCIE=0; // Habilita interrupción por recepción
TXIF=1; // Limpia la bandera de interrupción por transmisión
```

```
RCIF=0; // Limpia la bandera de interrupción por recepción


//Configuración para la transmisión de 8 bits.

void envia8Bits()

TX9=0; // Habilita transmisión de 8 BITS
TXEN=1; // Habilita transmisión
SYNC=0; // Selecciona transmisión asíncrona
BRGH=1; // Selecciona Alta velocidad
SPBRG=0x81; //
SPEN=1; // Habilita al puerto serial
TXIE=0; // Habilita interrupción por transmisión
RCIE=0; // Habilita interrupción por recepción
TXIF=1; //Limpia la bandera de interrupción por transmisión
RCIF=0; //Limpia la bandera de interrupción por recepción


// Configuración Recepción de 8 bits
    void recibe8Bits()

SYNC=0; //Selecciona transmisión asíncrona
BRGH=1; // Selecciona Alta velocidad
SPEN=1; // Habilita al puerto serial
RX9=0; // Habilita recepción de 8 BITS
CREN=1; // Habilita recepción continua
ADDEN=0; // Deshabilita detección de dirección
RCIE=1; // Habilita interrupción por recepción
RCIF=0;// Limpia la bandera de interrupción por recepción
// Configuración de Recepción de 9 bits
    void recibe9Bits()
SYNC=0; //Selecciona transmisión asíncrona
SPEN=1; // Habilita al puerto serial
RX9=1; // Habilita recepción de 9 Bits
CREN=1; // Habilita recepción continua
ADDEN=1; // Habilita detección de dirección
RX9D=1; // Noveno bit del dato recibido.
RCIE=1; // Habilita interrupción por recepción
RCIF=0;// Limpia la bandera de interrupción por recepción
    #include pic.h
#include comserialmaestro.h
```

Figura 4.15: Rutina de configuración para la transmisión serial.

```
#include inicializa.h
#include delay.h

    void inicializa2()
int esclavo1,esclavo2,esclavo3,esclavo4;
envia9Bits();
recibe9Bits();
Transmite1Byte(65); //Inicializa al Esclavo 1
Transmite1Byte(66); //Inicializa al Esclavo 2
Transmite1Byte(67); //Inicializa al Esclavo 3
Transmite1Byte(68); // Inicializa al Esclavo 4
```

Figura 4.16: Rutina de configuración del maestro para la inicialización.

Una vez concluida la etapa de inicialización, los Esclavos se encuentran configurados para recibir 9 bits e igualmente para detectar interrupciones por recepción. La ventaja de manejar interrupciones es que el PIC puede estar ejecutando otras rutinas mientras espera que la interrupción ocurra.

Por lo tanto mientras no ocurra una interrupción por recepción los Esclavos se encuentran adquiriendo datos provenientes de los sensores. Si en algún momento los Esclavos reciben un dato en 9 bits, éstos generan una interrupción y verifican dicho dato con su ID de dirección, si es igual, cambia la configuración de recepción de 9 bits a 8 bits de recepción e igualmente espera a que ocurra otra interrupción. En ésta ocasión, el Esclavo estará esperando ya sea una Petición o una ejecución de una acción según el ID de dato que reciba. La configuración de los IDs de datos se mostró en la Sección 3.4.

Una vez completada la acción o la petición, el Esclavo vuelve a su rutina principal de adquirir señales y espera un dato de 9 bits para generar una interrupción.
En el siguiente código se observa la forma en que el Esclavo 1 fue configurado para realizar las acciones y peticiones que el Maestro le envía.

```
    void escogertabla()

int codigo; float temp,humedad; codigo=RCREG;
```

```
switch(codigo)

// Recibe 201 en número binario y adquiere la temperatura del sensor del canal 1. Envía la confirmación y posteriormente la temperatura medida.
case 201:/* temperatura:*/
temp=canal_adquisicion(canal1);
envia9Bits();
Transmite1Byte(Direccionesclavo);
envia8Bits();
Transmite1Byte(temp);
break;
case 203: // Mide y envía Canal 2, La humedad
humedad=canal_adquisicion(canal2);
envia9Bits();
Transmite1Byte(Direccionesclavo);
envia8Bits();
Transmite1Byte(humedad);
break;
case 128: //Realiza una Acción. Cierra la Cortina 1
PORTD=0x00;
break;
case 129: //Abre la Cortina 1 un 33 por ciento
PORTD=0x01;
break;
case 130://Abre la Cortina 1 un 66 por ciento
PORTD=0x02;
break;
case 131: //Abre la Cortina 1 un 100 por ciento
PORTD=0x03;
break;
case 176: //Cierra la Cortina 4
PORTD=0x00;
break;
case 177: //Abre la Cortina 4 un 33 por ciento
PORTD=0x04;
break;
case 178: //Abre la Cortina 4 un 66 por ciento
PORTD=0x08;
break;
case 179:// Abre la Cortina 4 un 100 por ciento
PORTD=0x12;
break;
```

```
// Rutina de Adquisición de datos.

    unsigned char canal_adquisicion(int channel)

    ADCON0=ADIF=0; // Limpia la bandera
ADGO=1; //Comienza la adquisición
while(ADGO)
continue;
return(ADRESH); // Regresa los 8 bits más significativos
```

El Maestro por su parte, una vez que termina la etapa de inicialización, comienza una rutina de enviar peticiones a cada uno de los esclavos. Para ello, primeramente envía el ID de dirección del Esclavo, posteriormente envía el ID del dato que requiera en 8 bits y espera que el Esclavo le responda. Si el tiempo de espera excede un tiempo límite que se tiene programado, el Maestro vuelve a enviar el ID del dato. Si vuelve excederse el tiempo, el Maestro detecta un error de comunicación entre ambos Sistemas Mínimos.

La ventaja de configurar de esta manera al PIC, es que nos da cierta autonomía, es decir, si fallara la comunicación con la PC, el Maestro no deja de estar recibiendo los datos de los sensores y por lo tanto puede tener una pequeña ley de control en él para mandar las señales adecuadas a los actuadores. Para hacer la comunicación Maestro-PC se utilizan los Instrumentos Virtuales de LabVIEW descritos en la sección 4.3.1 para poder enviar y recibir datos del puerto serial. La configuración fue la siguiente:

- VISA resource name: COM1
- baud rate: 9600
- data bits: 8
- parity: Sin paridad
- stop bits: 1.0

Para la configuración del VISA write y del VISA read fue necesario especificar el nombre del puerto, en éste caso COM1, y la escritura del buffer cambia dependiendo de la acción o petición a realizar (ver Figura 4.17).

Para el caso de la lectura del buffer del VISA Read, la cadena de caracteres que lee (código ASCII) del puerto, es convertida a tipo de dato numérico para que pueda ser visualizado en los instrumentos virtuales, es decir, para el caso de la temperatura el dato leído en ASCII se convierte al valor numérico equivalente y así puede ser visualizado en un termómetro virtual. En la figura 4.18 se muestra la tabla del código ASCII.

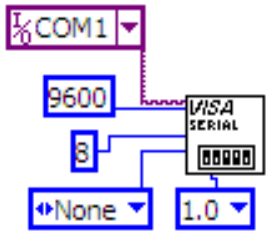



Figura 4.17: Configuración del módulo VISA.
*Fuente: [VISA, 2000].*

| Least Significant Character | Most Significant Character | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Hex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | 0 | NUL | DLE | Space | 0 | @ | P | ` | p |
| | 1 | SOH | DC1 | ! | 1 | A | Q | a | q |
| | 2 | STX | DC2 | " | 2 | B | R | b | r |
| | 3 | ETX | DC3 | # | 3 | C | S | c | s |
| | 4 | EOT | DC4 | $ | 4 | D | T | d | t |
| | 5 | ENQ | NAK | % | 5 | E | U | e | u |
| | 6 | ACK | SYN | & | 6 | F | V | f | v |
| | 7 | Bell | ETB | ' | 7 | G | W | g | w |
| | 8 | BS | CAN | ( | 8 | H | X | h | x |
| | 9 | HT | EM | ) | 9 | I | Y | i | y |
| | A | LF | SUB | * | : | J | Z | j | z |
| | B | VT | ESC | + | ; | K | [ | k | { |
| | C | FF | FS | , | < | L | \ | l | \| |
| | D | CR | GS | - | = | M | ] | m | } |
| | E | SO | RS | . | > | N | ^ | n | ~ |
| | F | SI | US | / | ? | O | _ | o | DEL |

Figura 4.18: Tabla ASCII.
*Fuente: [Campos, 1998].*

## 4.6. Pruebas realizadas al protocolo

Para comprobar que los esclavos realmente están recibiendo el ID correcto que les manda el maestro, la pruebas realizadas consistieron en utilizar 2 esclavos y un maestro.

Se realizaron tres pruebas significativas:

1. Inicialización de esclavos.

   En esta prueba el maestro envía una dirección en 9 bits que inicializa a un solo esclavos, los cuales se encuentran en la posición de recibir 9 bits. Una vez que detectan esta dirección, se comprobó que el esclavo al que le pertenece esta dirección de inicialización, retorna una confirmación de inicialización al maestro indicando que está presente y en comunicación; mientras que el otro esclavo solo ignoró esta dirección.

2. Petición de señales analógicas.

   Una vez comprobado que cuando el maestro envía una dirección a un esclavo solo éste retorna una respuesta, se procedió a que el maestro solicitara las señales analógicas que el esclavo adquiere en sus canales.
   Así mientras el maestro envía una dirección de 8 bits a un esclavo en el que solicita una señal determinada en un canal, el esclavo indicado responde con un byte que contiene la señal adquirida en el canal que el maestro solicitó. En los canales analógicos de cada esclavo se colocaron potenciómetros para regular el voltaje de entrada y simular la variación de una señal.

3. Comunicación PC-Maestro

   Esta prueba con el maestro se realizó por separado de los esclavos, solo utilizando un PIC y una computadora. Este PIC adquiere 4 señales analógicas en diferentes canales, para ser enviadas a la computadora mediante comunicación serial bajo la norma RS-232 y ser visualizadas en instrumentos virtuales de LABVIEW.

Así cuando el programa en LABVIEW solicita la adquisición de un canal determinado, éste envía un byte al microcontrolador en código ASCII, el cual es interpretado por el PIC respondiendo con la señal solicitada a la computadora mediante un byte. Esta señal puede ser visualizada en los instrumentos virtuales desplegados en pantalla casi en tiempo real.

**Conclusiones generales**

El presente proyecto de tesis está basado en la implementación de un sistema de comunicación serial distribuido utilizando el microcontrolador PIC16F877.

Éste fue hecho para supervisar y controlar las variables en un invernadero, involucrando el diseño y construcción de un sistema mínimo completo de bajo costo, que permita adquirir y procesar las mediciones de las variables físicas (internas y externas), además de las de control, alarmas y de estado de los diferentes dispositivos y actuadores del sistema distribuido, que controlarán el invernadero. Este sistema mínimo usa el microcontrolador PIC 16F877 como cerebro de los procesos.

El principal objetivo de estos sistemas mínimos a los que llamamos esclavos, es la adquisición y conversión digital de las señales analógicas, para después ser enviadas a otro sistema mínimo llamado maestro mediante comunicación serial y así el maestro envía esas señales a la PC, para la supervisión y control del invernadero.

Con este proyecto se busca tener un sistema mínimo completamente desarrollado para lograr un sistema propio y un cierto grado de independencia tecnológica. Basándonos en una arquitectura abierta, se realizó un sistema que se acople a nuestras necesidades, y a diferencia de lo que actualmente existe lograr acoplarse a los diseños de los sistemas comerciales, los cuales son pensados principalmente para la industria automotriz, por lo que tienen un elevado costo.

El presente trabajo se podrá implementar en distintos procesos que demanden la adquisición de datos y la acción de actuadores. Sin embargo debido a las características del microcontrolador utilizado en los sistemas mínimos, se tiene un número limitado de entradas analógicas usadas en la adquisición de datos provenientes de los sensores y un número limitado de salidas digitales empleadas para accionar los actuadores. No obstante la arquitectura planteada en este trabajo de tesis, facilita tener un número suficiente de entradas o salidas de señales para controlar distintos procesos.

# Bibliografía


[Angulo, 1999] Angulo, U. J., A. M. I. (1999). *Microcontroladores PIC. Diseño práctico de aplicaciones. Primera Parte.* Mc Graw Hill.

[Angulo, 2000] Angulo, U. J., e. a. (2000). *Microcontroladores PIC. Diseño práctico de aplicaciones. Segunda Parte.* Mc Graw Hill.

[Campos, 1998] Campos, C.M y Castañeda, P. (1998). Implementación de un sistema de desarrollo utilizando los microcontroladores pic de microchip. Internet site, Universidad de Guadalajara.

[Dominique, 1996] Dominique, P. (1996). *Réseaux Locaux, Le bus CAN.* DUNOD.

[Dumetri, 2002] Dumetri, P. (2002). Descripcion general del pic16f877. Internet site, foro.

[Etschberger, 2000] Etschberger, K. (2000). *CAN Controller Area Network.* Éditeur Hanser.

[Garbutt, 2003] Garbutt, M. (2003). *Asynchronous Communications with the PICmicro USART.* Microchip Technology Inc No. AN774.

[Inc, 2001] Inc, M. T. (2001). *USART. Using the USART in asynchronous mode.* Microchip Technology Inc Tutorial.

[Inc, 2002] Inc, M. T. (2002). *Industrial Data Comunicactions RS-232/RS485.* Microchip Technology Inc Tutorial No. 2.

[Kiencke, 1994] Kiencke, U. (1994). *Controller Area Network - from concept to reality -Universidad de Karlsruhe (IIIT).* CiA publications.

[Microchip, 2001] Microchip (2001). *PIC16F87X 28/40-Pin 8-Bit CMOS FLASH Microcontrollers.* Microchip Technology Data Sheets No. DS30292C.

[Microchip, 2005] Microchip (2005). *MPLAB IDE Quick Start Guide*, pages 1–11. Microchip Technology Inc.

[Morcillo, 2000] Morcillo, R.P y Cócera, R. (2000). *Comunicaciones Industriales*. Paraninfo Thompson Learning.

[PICC, 2005] PICC, H.-T. (2005). The picc-compiler for the microchip picmicro family. Internet site, HI-TECH SOFTWARE.

[Technology, 2001] Technology, M. (2001). *Analog to Digital Converters*. Microchip Technology Tutorial.

[VISA, 2000] VISA, N. (2000). Labview basics ii. course manual. Manual, National Instruments.

# Apéndice A

# Especificaciones del microcontrolador PIC16F877

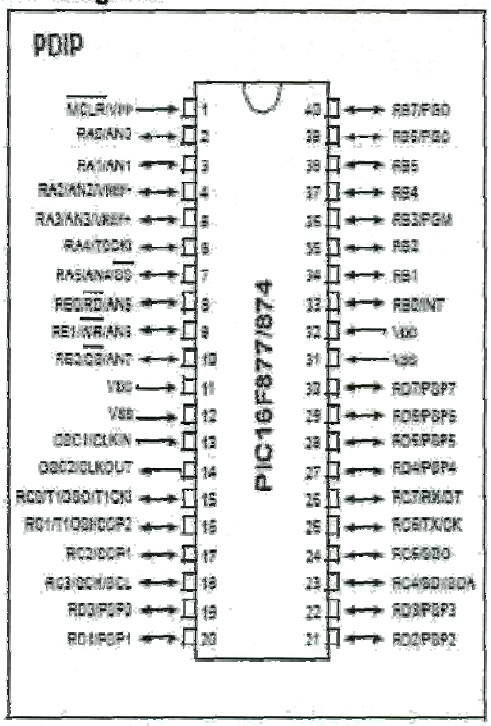
## Devices Included in this Data Sheet:

PIC16F873 • PIC16F874
• PIC16F876 • PIC16F877

## Microcontroller Core Features:

• High performance RISC CPU
• Only 35 single word instructions to learn
• All single cycle instructions except for program branches which are two cycle
• Operating speed: DC - 20 MHz clock input
DC - 200 ns instruction cycle
• Up to 8K x 14 words of FLASH Program Memory,
Up to 368 x 8 bytes of Data Memory (RAM)
Up to 256 x 8 bytes of EEPROM Data Memory
• Pinout compatible to the PIC16C73B/74B/76/77
• Interrupt capability (up to 14 sources)
• Eight level deep hardware stack
• Direct, indirect and relative addressing modes
• Power-on Reset (POR)
• Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
• Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
• Programmable code protection
• Power saving SLEEP mode
• Selectable oscillator options
• Low power, high speed CMOS FLASH/EEPROM technology
• Fully static design
• In-Circuit Serial Programming (ICSP) via two pins
• Single 5V In-Circuit Serial Programming capability
• In-Circuit Debugging via two pins
• Processor read/write access to program memory
• Wide operating voltage range: 2.0V to 5.5V
• High Sink/Source Current: 25 mA
• Commercial, Industrial and Extended temperatureranges
• Low-power consumption:
- < 0.6 mA typical @ 3V, 4 MHz
- 20 ìA typical @ 3V, 32 kHz
- < 1 ìA typical standby current

## Pin Diagram




## Peripheral Features:

• Timer0: 8-bit timer/counter with 8-bit prescaler
• Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
• Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
• Two Capture, Compare, PWM modules
- Capture is 16-bit, max. resolution is 12.5 ns
- Compare is 16-bit, max. resolution is 200 ns
- PWM max. resolution is 10-bit
• 10-bit multi-channel Analog-to-Digital converter
• Synchronous Serial Port (SSP) with SPI (Master mode) and I2C (Master/Slave)
• Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
• Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls (40/44-pin only)
• Brown-out detection circuitry for Brown-out Reset (BOR)

# PINOUT DESCRIPTION

| Pin Name | DIP Pin# | PLCC Pin# | CFP Pin# | I/O/P Type | Buffer Type | Description |
|---|---|---|---|---|---|---|
| | | | | | | PORTC is a bi-directional I/O port. |
| RC0/T1OSO/T1CKI | 15 | 16 | 32 | I/O | ST | RC0 can also be the Timer1 oscillator output or a Timer1 clock input. |
| RC1/T1OSI/CCP2 | 16 | 18 | 35 | I/O | ST | RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output. |
| RC2/CCP1 | 17 | 19 | 36 | I/O | ST | RC2 can also be the Capture1 input/Compare1 output/PWM1 output. |
| RC3/SCK/SCL | 18 | 20 | 37 | I/O | ST | RC3 can also be the synchronous serial clock input/output for both SPI and $I^2C$ modes. |
| RC4/SDI/SDA | 23 | 25 | 42 | I/O | ST | RC4 can also be the SPI Data In (SPI mode) or data I/O ($I^2C$ mode). |
| RC5/SDO | 24 | 26 | 43 | I/O | ST | RC5 can also be the SPI Data Out (SPI mode). |
| RC6/TX/CK | 25 | 27 | 44 | I/O | ST | RC6 can also be the USART Asynchronous Transmit or Synchronous Clock. |
| RC7/RX/DT | 26 | 29 | 1 | I/O | ST | RC7 can also be the USART Asynchronous Receive or Synchronous Data. |
| | | | | | | PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus. |
| RD0/PSP0 | 19 | 21 | 38 | I/O | ST/TTL[3] | |
| RD1/PSP1 | 20 | 22 | 39 | I/O | ST/TTL[3] | |
| RD2/PSP2 | 21 | 23 | 40 | I/O | ST/TTL[3] | |
| RD3/PSP3 | 22 | 24 | 41 | I/O | ST/TTL[3] | |
| RD4/PSP4 | 27 | 30 | 2 | I/O | ST/TTL[3] | |
| RD5/PSP5 | 28 | 31 | 3 | I/O | ST/TTL[3] | |
| RD6/PSP6 | 29 | 32 | 4 | I/O | ST/TTL[3] | |
| RD7/PSP7 | 30 | 33 | 5 | I/O | ST/TTL[3] | |
| | | | | | | PORTE is a bi-directional I/O port. |
| RE0/$\overline{RD}$/AN5 | 8 | 9 | 25 | I/O | ST/TTL[3] | RE0 can also be read control for the parallel slave port, or analog input5. |
| RE1/$\overline{WR}$/AN6 | 9 | 10 | 26 | I/O | ST/TTL[3] | RE1 can also be write control for the parallel slave port, or analog input6. |
| RE2/$\overline{CS}$/AN7 | 10 | 11 | 27 | I/O | ST/TTL[3] | RE2 can also be select control for the parallel slave port, or analog input7. |
| Vss | 12,31 | 13,34 | 6,29 | P | — | Ground reference for logic and I/O pins. |
| VDD | 11,32 | 12,35 | 7,28 | P | — | Positive supply for logic and I/O pins. |
| NC | — | 1,17,28, 40 | 12,13, 33,34 | | — | These pins are not internally connected. These pins should be left unconnected. |

## SPECIAL FUNCTION REGISTER SUMARY

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on: POR, BOR | Details on page: |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bank 0 | | | | | | | | | | | |
| 00h(3) | INDF | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | 0000 0000 | 27 |
| 01h | TMR0 | Timer0 Module Register | | | | | | | | xxxx xxxx | 47 |
| 02h(3) | PCL | Program Counter (PC) Least Significant Byte | | | | | | | | 0000 0000 | 26 |
| 03h(3) | STATUS | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 18 |
| 04h(3) | FSR | Indirect Data Memory Address Pointer | | | | | | | | xxxx xxxx | 27 |
| 05h | PORTA | — | — | PORTA Data Latch when written: PORTA pins when read | | | | | | --0x 0000 | 29 |
| 06h | PORTB | PORTB Data Latch when written: PORTB pins when read | | | | | | | | xxxx xxxx | 31 |
| 07h | PORTC | PORTC Data Latch when written: PORTC pins when read | | | | | | | | xxxx xxxx | 33 |
| 08h(4) | PORTD | PORTD Data Latch when written: PORTD pins when read | | | | | | | | xxxx xxxx | 35 |
| 09h(4) | PORTE | — | — | — | — | — | RE2 | RE1 | RE0 | ---- -xxx | 36 |
| 0Ah(1,3) | PCLATH | — | — | — | Write Buffer for the upper 5 bits of the Program Counter | | | | | ---0 0000 | 26 |
| 0Bh(3) | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 20 |
| 0Ch | PIR1 | PSPIF(3) | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF | 0000 0000 | 22 |
| 0Dh | PIR2 | — | (5) | — | EEIF | BCLIF | — | — | CCP2IF | -r-0 0--0 | 24 |
| 0Eh | TMR1L | Holding register for the Least Significant Byte of the 16-bit TMR1 Register | | | | | | | | xxxx xxxx | 52 |
| 0Fh | TMR1H | Holding register for the Most Significant Byte of the 16-bit TMR1 Register | | | | | | | | xxxx xxxx | 52 |
| 10h | T1CON | — | — | T1CKPS1 | T1CKPS0 | T1OSCEN | T1SYNC | TMR1CS | TMR1ON | --00 0000 | 51 |
| 11h | TMR2 | Timer2 Module Register | | | | | | | | 0000 0000 | 55 |
| 12h | T2CON | — | TOUTPS3 | TOUTPS2 | TOUTPS1 | TOUTPS0 | TMR2ON | T2CKPS1 | T2CKPS0 | -000 0000 | 55 |
| 13h | SSPBUF | Synchronous Serial Port Receive Buffer/Transmit Register | | | | | | | | xxxx xxxx | 70, 73 |
| 14h | SSPCON | WCOL | SSPOV | SSPEN | CKP | SSPM3 | SSPM2 | SSPM1 | SSPM0 | 0000 0000 | 67 |
| 15h | CCPR1L | Capture/Compare/PWM Register1 (LSB) | | | | | | | | xxxx xxxx | 57 |
| 16h | CCPR1H | Capture/Compare/PWM Register1 (MSB) | | | | | | | | xxxx xxxx | 57 |
| 17h | CCP1CON | — | — | CCP1X | CCP1Y | CCP1M3 | CCP1M2 | CCP1M1 | CCP1M0 | --00 0000 | 58 |
| 18h | RCSTA | SPEN | RX9 | SREN | CREN | ADDEN | FERR | OERR | RX9D | 0000 000x | 96 |
| 19h | TXREG | USART Transmit Data Register | | | | | | | | 0000 0000 | 99 |
| 1Ah | RCREG | USART Receive Data Register | | | | | | | | 0000 0000 | 101 |
| 1Bh | CCPR2L | Capture/Compare/PWM Register2 (LSB) | | | | | | | | xxxx xxxx | 57 |
| 1Ch | CCPR2H | Capture/Compare/PWM Register2 (MSB) | | | | | | | | xxxx xxxx | 57 |
| 1Dh | CCP2CON | — | — | CCP2X | CCP2Y | CCP2M3 | CCP2M2 | CCP2M1 | CCP2M0 | --00 0000 | 58 |
| 1Eh | ADRESH | A/D Result Register High Byte | | | | | | | | xxxx xxxx | 116 |
| 1Fh | ADCON0 | ADCS1 | ADCS0 | CHS2 | CHS1 | CHS0 | GO/$\overline{DONE}$ | — | ADON | 0000 00-0 | 111 |

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on: POR, BOR | Details on page: |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bank 1 | | | | | | | | | | | |
| 80h(3) | INDF | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | 0000 0000 | 27 |
| 81h | OPTION_REG | $\overline{RBPU}$ | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 | 1111 1111 | 19 |
| 82h(3) | PCL | Program Counter (PC) Least Significant Byte | | | | | | | | 0000 0000 | 26 |
| 83h(3) | STATUS | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 18 |
| 84h(3) | FSR | Indirect Data Memory Address Pointer | | | | | | | | xxxx xxxx | 27 |
| 85h | TRISA | — | — | PORTA Data Direction Register | | | | | | --11 1111 | 29 |
| 86h | TRISB | PORTB Data Direction Register | | | | | | | | 1111 1111 | 31 |
| 87h | TRISC | PORTC Data Direction Register | | | | | | | | 1111 1111 | 33 |
| 88h(4) | TRISD | PORTD Data Direction Register | | | | | | | | 1111 1111 | 35 |
| 89h(4) | TRISE | IBF | OBF | IBOV | PSPMODE | — | PORTE Data Direction Bits | | | 0000 -111 | 37 |
| 8Ah(1,3) | PCLATH | — | — | — | Write Buffer for the upper 5 bits of the Program Counter | | | | | ---0 0000 | 26 |
| 8Bh(3) | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 20 |
| 8Ch | PIE1 | PSPIE(2) | ADIE | RCIE | TXIE | SSPIE | CCP1IE | TMR2IE | TMR1IE | 0000 0000 | 21 |
| 8Dh | PIE2 | — | (5) | — | EEIE | BCLIE | — | — | CCP2IE | -r-0 0--0 | 23 |
| 8Eh | PCON | — | — | — | — | — | — | $\overline{POR}$ | $\overline{BOR}$ | ---- --qq | 25 |
| 8Fh | — | Unimplemented | | | | | | | | — | — |
| 90h | — | Unimplemented | | | | | | | | — | — |
| 91h | SSPCON2 | GCEN | ACKSTAT | ACKDT | ACKEN | RCEN | PEN | RSEN | SEN | 0000 0000 | 68 |
| 92h | PR2 | Timer2 Period Register | | | | | | | | 1111 1111 | 55 |
| 93h | SSPADD | Synchronous Serial Port ($I^2C$ mode) Address Register | | | | | | | | 0000 0000 | 73, 74 |
| 94h | SSPSTAT | SMP | CKE | D/$\overline{A}$ | P | S | R/$\overline{W}$ | UA | BF | 0000 0000 | 66 |
| 95h | — | Unimplemented | | | | | | | | — | — |
| 96h | — | Unimplemented | | | | | | | | — | — |
| 97h | — | Unimplemented | | | | | | | | — | — |
| 98h | TXSTA | CSRC | TX9 | TXEN | SYNC | — | BRGH | TRMT | TX9D | 0000 -010 | 95 |
| 99h | SPBRG | Baud Rate Generator Register | | | | | | | | 0000 0000 | 97 |
| 9Ah | — | Unimplemented | | | | | | | | — | — |
| 9Bh | — | Unimplemented | | | | | | | | — | — |
| 9Ch | — | Unimplemented | | | | | | | | — | — |
| 9Dh | — | Unimplemented | | | | | | | | — | — |
| 9Eh | ADRESL | A/D Result Register Low Byte | | | | | | | | xxxx xxxx | 116 |
| 9Fh | ADCON1 | ADFM | — | — | — | PCFG3 | PCFG2 | PCFG1 | PCFG0 | 0--- 0000 | 112 |

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on: POR, BOR | Details on page: |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bank 2 | | | | | | | | | | | |
| 100h(3) | INDF | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | 0000 0000 | 27 |
| 101h | TMR0 | Timer0 Module Register | | | | | | | | xxxx xxxx | 47 |
| 102h(3) | PCL | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 | 26 |
| 103h(3) | STATUS | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 18 |
| 104h(3) | FSR | Indirect Data Memory Address Pointer | | | | | | | | xxxx xxxx | 27 |
| 105h | — | Unimplemented | | | | | | | | — | — |
| 106h | PORTB | PORTB Data Latch when written: PORTB pins when read | | | | | | | | xxxx xxxx | 31 |
| 107h | — | Unimplemented | | | | | | | | — | — |
| 108h | — | Unimplemented | | | | | | | | — | — |
| 109h | — | Unimplemented | | | | | | | | — | — |
| 10Ah(1,3) | PCLATH | — | — | — | Write Buffer for the upper 5 bits of the Program Counter | | | | | ---0 0000 | 26 |
| 10Bh(3) | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 20 |
| 10Ch | EEDATA | EEPROM Data Register Low Byte | | | | | | | | xxxx xxxx | 41 |
| 10Dh | EEADR | EEPROM Address Register Low Byte | | | | | | | | xxxx xxxx | 41 |
| 10Eh | EEDATH | — | — | EEPROM Data Register High Byte | | | | | | xxxx xxxx | 41 |
| 10Fh | EEADRH | — | — | — | EEPROM Address Register High Byte | | | | | xxxx xxxx | 41 |
| Bank 3 | | | | | | | | | | | |
| 180h(3) | INDF | Addressing this location uses contents of FSR to address data memory (not a physical register) | | | | | | | | 0000 0000 | 27 |
| 181h | OPTION_REG | RBPU | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 | 1111 1111 | 19 |
| 182h(3) | PCL | Program Counter (PC) Least Significant Byte | | | | | | | | 0000 0000 | 26 |
| 183h(3) | STATUS | IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | 0001 1xxx | 18 |
| 184h(3) | FSR | Indirect Data Memory Address Pointer | | | | | | | | xxxx xxxx | 27 |
| 185h | — | Unimplemented | | | | | | | | — | — |
| 186h | TRISB | PORTB Data Direction Register | | | | | | | | 1111 1111 | 31 |
| 187h | — | Unimplemented | | | | | | | | — | — |
| 188h | — | Unimplemented | | | | | | | | — | — |
| 189h | — | Unimplemented | | | | | | | | — | — |
| 18Ah(1,3) | PCLATH | — | — | — | Write Buffer for the upper 5 bits of the Program Counter | | | | | ---0 0000 | 26 |
| 18Bh(3) | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 20 |
| 18Ch | EECON1 | EEPGD | — | — | — | WRERR | WREN | WR | RD | x--- x000 | 41, 42 |
| 18Dh | EECON2 | EEPROM Control Register2 (not a physical register) | | | | | | | | ---- ---- | 41 |
| 18Eh | — | Reserved maintain clear | | | | | | | | 0000 0000 | — |
| 18Fh | — | Reserved maintain clear | | | | | | | | 0000 0000 | — |

# USART Baud Rate Generator (BRG)

The BRG supports both the Asynchronous and Synchronous modes of the USART. It is a dedicated 8-bit baud rate generator. The SPBRG register controls the period of a free running 8-bit timer. In Asynchronous mode, bit BRGH (TXSTA<2>) also controls the baud rate. In Synchronous mode, bit BRGH is ignored. Table 1 shows the formula for computation of the baud rate for different USART modes which only apply in Master mode (internal clock). Given the desired baud rate and FOSC, the nearest integer value for the SPBRG register can be calculated using the formula in Table 1. From this, the error in baud rate can be determined. It may be advantageous to use the high baud rate (BRGH = 1), even for slower baud clocks. This is because the FOSC/(16(X + 1)) equation can reduce the baud rate error in some cases. Writing a new value to the SPBRG register causes the BRG timer to be reset (or cleared). This ensures the BRG does not wait for a timer overflow before outputting the new baud rate.

## SAMPLING

The data on the RC7/RX/DT pin is sampled three times by a majority detect circuit to determine if a high or a low level is present at the RX pin.

TABLE 1: BAUD RATE FORMULA

| SYNC | BRGH = 0 (Low Speed) | BRGH = 1 (High Speed) |
|---|---|---|
| 0 | (Asynchronous) Baud Rate = FOSC/(64(X+1)) | Baud Rate = FOSC/(16(X+1)) |
| 1 | (Synchronous) Baud Rate = FOSC/(4(X+1)) | N/A |

X = value in SPBRG (0 to 255)

TABLE 2: REGISTER ASSOCIATED WITH BAUD RATE GENERATOR

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on: POR, BOR | Value on all other RESETS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 98h | TXSTA | CSRC | TX9 | TXEN | SYNC | — | BRGH | TRMT | TX9D | 0000 -010 | 0000 -010 |
| 18h | RCSTA | SPEN | RX9 | SREN | CREN | ADDEN | FERR | OERR | RX9D | 0000 000x | 0000 000x |
| 99h | SPBRG | Baud Rate Generator Register | | | | | | | | 0000 0000 | 0000 0000 |

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used by the BRG.

TABLE 3: BAUD RATES FOR ASYNCHRONOUS MODE (BRG=0)

| BAUD RATE (K) | Fosc = 20 MHz | | | Fosc = 16 MHz | | | Fosc = 10 MHz | | |
|---|---|---|---|---|---|---|---|---|---|
| | KBAUD | % ERROR | SPBRG value (decimal) | KBAUD | % ERROR | SPBRG value (decimal) | KBAUD | % ERROR | SPBRG value (decimal) |
| 0.3 | - | - | - | - | - | - | - | - | - |
| 1.2 | 1.221 | 1.75 | 255 | 1.202 | 0.17 | 207 | 1.202 | 0.17 | 129 |
| 2.4 | 2.404 | 0.17 | 129 | 2.404 | 0.17 | 103 | 2.404 | 0.17 | 64 |
| 9.6 | 9.766 | 1.73 | 31 | 9.615 | 0.16 | 25 | 9.766 | 1.73 | 15 |
| 19.2 | 19.531 | 1.72 | 15 | 19.231 | 0.16 | 12 | 19.531 | 1.72 | 7 |
| 28.8 | 31.250 | 8.51 | 9 | 27.778 | 3.55 | 8 | 31.250 | 8.51 | 4 |
| 33.6 | 34.722 | 3.34 | 8 | 35.714 | 6.29 | 6 | 31.250 | 6.99 | 4 |
| 57.6 | 62.500 | 8.51 | 4 | 62.500 | 8.51 | 3 | 52.083 | 9.58 | 2 |
| HIGH | 1.221 | - | 255 | 0.977 | - | 255 | 0.610 | - | 255 |
| LOW | 312.500 | - | 0 | 250.000 | - | 0 | 156.250 | - | 0 |

| BAUD RATE (K) | Fosc = 4 MHz | | | Fosc = 3.6864 MHz | | |
|---|---|---|---|---|---|---|
| | KBAUD | % ERROR | SPBRG value (decimal) | KBAUD | % ERROR | SPBRG value (decimal) |
| 0.3 | 0.300 | 0 | 207 | 0.3 | 0 | 191 |
| 1.2 | 1.202 | 0.17 | 51 | 1.2 | 0 | 47 |
| 2.4 | 2.404 | 0.17 | 25 | 2.4 | 0 | 23 |
| 9.6 | 8.929 | 6.99 | 6 | 9.6 | 0 | 5 |
| 19.2 | 20.833 | 8.51 | 2 | 19.2 | 0 | 2 |
| 28.8 | 31.250 | 8.51 | 1 | 28.8 | 0 | 1 |
| 33.6 | - | - | - | - | - | - |
| 57.6 | 62.500 | 8.51 | 0 | 57.6 | 0 | 0 |
| HIGH | 0.244 | - | 255 | 0.225 | - | 255 |
| LOW | 62.500 | - | 0 | 57.6 | - | 0 |

TABLE 4: BAUD RATES FOR ASYNCHRONOUS MODE (BRG=1)

| BAUD RATE (K) | Fosc = 20 MHz | | | Fosc = 16 MHz | | | Fosc = 10 MHz | | |
|---|---|---|---|---|---|---|---|---|---|
| | KBAUD | % ERROR | SPBRG value (decimal) | KBAUD | % ERROR | SPBRG value (decimal) | KBAUD | % ERROR | SPBRG value (decimal) |
| 0.3 | - | - | - | - | - | - | - | - | - |
| 1.2 | - | - | - | - | - | - | - | - | - |
| 2.4 | - | - | - | - | - | - | 2.441 | 1.71 | 255 |
| 9.6 | 9.615 | 0.16 | 129 | 9.615 | 0.16 | 103 | 9.615 | 0.16 | 64 |
| 19.2 | 19.231 | 0.16 | 64 | 19.231 | 0.16 | 51 | 19.531 | 1.72 | 31 |
| 28.8 | 29.070 | 0.94 | 42 | 29.412 | 2.13 | 33 | 28.409 | 1.36 | 21 |
| 33.6 | 33.784 | 0.55 | 36 | 33.333 | 0.79 | 29 | 32.895 | 2.10 | 18 |
| 57.6 | 59.524 | 3.34 | 20 | 58.824 | 2.13 | 16 | 56.818 | 1.36 | 10 |
| HIGH | 4.883 | - | 255 | 3.906 | - | 255 | 2.441 | - | 255 |
| LOW | 1250.000 | - | 0 | 1000.000 | - | 0 | 625.000 | - | 0 |

| BAUD RATE (K) | Fosc = 4 MHz | | | Fosc = 3.6864 MHz | | |
|---|---|---|---|---|---|---|
| | KBAUD | % ERROR | SPBRG value (decimal) | KBAUD | % ERROR | SPBRG value (decimal) |
| 0.3 | - | - | - | - | - | - |
| 1.2 | 1.202 | 0.17 | 207 | 1.2 | 0 | 191 |
| 2.4 | 2.404 | 0.17 | 103 | 2.4 | 0 | 95 |
| 9.6 | 9.615 | 0.16 | 25 | 9.6 | 0 | 23 |
| 19.2 | 19.231 | 0.16 | 12 | 19.2 | 0 | 11 |
| 28.8 | 27.798 | 3.55 | 8 | 28.8 | 0 | 7 |
| 33.6 | 35.714 | 6.29 | 6 | 32.9 | 2.04 | 6 |
| 57.6 | 62.500 | 8.51 | 3 | 57.6 | 0 | 3 |
| HIGH | 0.977 | - | 255 | 0.9 | - | 255 |
| LOW | 250.000 | - | 0 | 230.4 | - | 0 |

# USART Asynchronous Mode

In this mode, the USART uses standard non-return-tozero(NRZ) format (one START bit, eight or nine databits, and one STOP bit). The most common data formatis 8-bits. An on-chip, dedicated, 8-bit baud rate generator can be used to derive standard baud rate frequencies from the oscillator. The USART transmits and receives the LSb first. The transmitter and receiver are functionally independent, but use the same data format and baud rate. The baud rate generator produces a clock, either x16 or x64 of the bit shift rate, depending on bit BRGH (TXSTA<2>). Parity is not supported by the hardware, but can be implemented in software (and stored as the ninth data bit). Asynchronous mode is stopped during SLEEP.

Asynchronous mode is selected by clearing bit SYNC (TXSTA<4>).

The USART Asynchronous module consists of the following important elements:

- Baud Rate Generator
- Sampling Circuit
- Asynchronous Transmitter
- Asynchronous Receiver

## USART ASYNCHRONOUS TRANSMITTER

The heart of the transmitter is the transmit (serial) shift register (TSR). The shift register obtains its data from the read/write transmit buffer, TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the STOP bit has been transmitted from the previous load. As soon as the STOP bit is transmitted, the TSR is loaded with new data from the TXREG register (if available). Once the TXREG register transfers the data to the TSR register (occurs in one TCY), the TXREG register is empty and flag bit TXIF (PIR1<4>) is set. This interrupt can be enabled/disabled by setting/clearing enable bit TXIE ( PIE1<4>). Flag bit TXIF will be set, regardless of the state of enable bit TXIE and cannot be cleared in software.

It will reset only when new data is loaded into the TXREG register. While flag bit TXIF indicates the status of the TXREG register, another bit TRMT (TXSTA<1>) shows the status of the TSR register. Status bit TRMT is a read only bit, which is set when the TSR register is empty. No interrupt logic is tied to this bit, so the user has to poll this bit in order to determine if the TSR register is empty.

Transmission is enabled by setting enable bit TXEN (TXSTA<5>). The actual transmission will not occur until the TXREG register has been loaded with data and the baud rate generator (BRG) has produced a shift clock . The transmission can also be started by first loading the TXREG register and then setting enable bit TXEN. Normally, when transmission is first started, the TSR register is empty. At that point, transfer to the TXREG register will result in an immediate transfer to TSR, resulting in an empty TXREG.

Clearing enable bit TXEN during a transmission will cause the transmission to be aborted and will reset the transmitter. As a result, the RC6/TX/CK pin will revert to hi-impedance.

In order to select 9-bit transmission, transmit bit TX9 (TXSTA<6>) should be set and the ninth bit should be written to TX9D (TXSTA<0>). The ninth bit must be written before writing the 8-bit data to the TXREG register.

This is because a data write to the TXREG register

can result in an immediate transfer of the data to the TSR register (if the TSR is empty). In such a case, an incorrect ninth data bit may be loaded in the TSR register.

TABLE 5: REGISTER ASSOCIATED WITH ASYNCHRONOUS TRANSMISSION

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on: POR, BOR | Value on all other RESETS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0Bh, 8Bh, 10Bh, 18Bh | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | R0IF | 0000 000x | 0000 000u |
| 0Ch | PIR1 | PSPIF(1) | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF | 0000 0000 | 0000 0000 |
| 18h | RCSTA | SPEN | RX9 | SREN | CREN | — | FERR | OERR | RX9D | 0000 -00x | 0000 -00x |
| 19h | TXREG | USART Transmit Register | | | | | | | | 0000 0000 | 0000 0000 |
| 8Ch | PIE1 | PSPIE(1) | ADIE | RCIE | TXIE | SSPIE | CCP1IE | TMR2IE | TMR1IE | 0000 0000 | 0000 0000 |
| 98h | TXSTA | CSRC | TX9 | TXEN | SYNC | — | BRGH | TRMT | TX9D | 0000 -010 | 0000 -010 |
| 99h | SPBRG | Baud Rate Generator Register | | | | | | | | 0000 0000 | 0000 0000 |

# USART ASYNCHRONOUS RECEIVER

The receiver block diagram is shown in Figure 10-4. The data is received on the RC7/RX/DT pin and drives the data recovery block. The data recovery block is actually a high speed shifter, operating at x16 times the baud rate; whereas, the main receive serial shifter operates at the bit rate or at FOSC. Once Asynchronous mode is selected, reception is enabled by setting bit CREN (RCSTA<4>). The heart of the receiver is the receive (serial) shift register (RSR). After sampling the STOP bit, the received data in the RSR is transferred to the RCREG register (if it is empty). If the transfer is complete, flag bit RCIF (PIR1<5>) is set. The actual interrupt can be enabled/ disabled by setting/clearing enable bit RCIE (PIE1<5>). Flag bit RCIF is a read only bit, which is cleared by the hardware. It is cleared when the RCREG

register has been read and is empty.

The RCREG is a double buffered register (i.e., it is a two deep FIFO). It is possible for two bytes of data to be received and transferred to the RCREG FIFO and a third byte to begin shifting to the RSR register. On the detection of the STOP bit of the third byte, if the RCREG register is still full, the overrun error bit OERR (RCSTA<1>) will be set. The word in the RSR will be lost. The RCREG register can be read twice to retrieve the two bytes in the FIFO. Overrun bit OERR has to be cleared in software.

This is done by resetting the receive logic (CREN is cleared and then set). If bit OERR is set, transfers from the RSR register to the RCREG register are inhibited, and no further data will be received. It is therefore, essential to clear error bit OERR if it is set. Framing error bit FERR (RCSTA<2>) is set if a STOP bit is detected as clear. Bit FERR and the 9th receive bit are

buffered the same way as the receive data. Reading the RCREG will load bits RX9D and FERR with new values, therefore, it is essential for the user to read the RCSTA register before reading the RCREG register in order not to lose the old FERR and RX9D information.

TABLE 6: REGISTER ASSOCIATED WITH ASYNCHRONOUS RECEPTION

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on: POR, BOR | Value on all other RESETS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0Bh, 8Bh, 10Bh,18Bh | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | R0IF | 0000 000x | 0000 000u |
| 0Ch | PIR1 | PSPIF(1) | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF | 0000 0000 | 0000 0000 |
| 18h | RCSTA | SPEN | RX9 | SREN | CREN | — | FERR | OERR | RX9D | 0000 -00x | 0000 -00x |
| 1Ah | RCREG | USART Receive Register | | | | | | | | 0000 0000 | 0000 0000 |
| 8Ch | PIE1 | PSPIE(1) | ADIE | RCIE | TXIE | SSPIE | CCP1IE | TMR2IE | TMR1IE | 0000 0000 | 0000 0000 |
| 98h | TXSTA | CSRC | TX9 | TXEN | SYNC | — | BRGH | TRMT | TX9D | 0000 -010 | 0000 -010 |
| 99h | SPBRG | Baud Rate Generator Register | | | | | | | | 0000 0000 | 0000 0000 |

# CONVERTER (A/D) MODULE

The Analog-to-Digital (A/D) Converter module has five inputs for the 28-pin devices and eight for the other devices. The analog input charges a sample and hold capacitor. The output of the sample and hold capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation. The A/D conversion of the analog input signal results in a corresponding 10-bit digital number. The A/D module has high and low voltage reference input that is software selectable to some combination of VDD, VSS, RA2, or RA3. The A/D converter has a unique feature of being able to operate while the device is in SLEEP mode. To operate in SLEEP, the A/D clock must be derived from the A/D's internal RC oscillator.
The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)

The ADCON0 register, shown in Register 11-1, controls the operation of the A/D module. The ADCON1 register, shown in Register 11-2, configures the functions of the port pins. The port pins can be configured as analog inputs (RA3 can also be the voltage reference), or as digital I/O.

TABLE 7: REGISTER ASSOCIATED WITH A/D

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on POR, BOR | Value on $\overline{MCLR}$, WDT |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0Bh,8Bh, 10Bh,18Bh | INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF | 0000 000x | 0000 000u |
| 0Ch | PIR1 | PSPIF(1) | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF | 0000 0000 | 0000 0000 |
| 8Ch | PIE1 | PSPIE(1) | ADIE | RCIE | TXIE | SSPIE | CCP1IE | TMR2IE | TMR1IE | 0000 0000 | 0000 0000 |
| 1Eh | ADRESH | A/D Result Register High Byte | | | | | | | | xxxx xxxx | uuuu uuuu |
| 9Eh | ADRESL | A/D Result Register Low Byte | | | | | | | | xxxx xxxx | uuuu uuuu |
| 1Fh | ADCON0 | ADCS1 | ADCS0 | CHS2 | CHS1 | CHS0 | GO/$\overline{DONE}$ | — | ADON | 0000 00-0 | 0000 00-0 |
| 9Fh | ADCON1 | ADFM | — | — | — | PCFG3 | PCFG2 | PCFG1 | PCFG0 | --0- 0000 | --0- 0000 |
| 85h | TRISA | — | — | PORTA Data Direction Register | | | | | | --11 1111 | --11 1111 |
| 05h | PORTA | — | — | PORTA Data Latch when written: PORTA pins when read | | | | | | --0x 0000 | --0u 0000 |
| 89h(1) | TRISE | IBF | OBF | IBOV | PSPMODE | — | PORTE Data Direction bits | | | 0000 -111 | 0000 -111 |
| 09h(1) | PORTE | — | — | — | — | — | RE2 | RE1 | RE0 | ---- -xxx | ---- -uuu |

# Oscillator Configurations

## OSCILLATOR TYPES

The PIC16F87X can be operated in four different oscillator modes. The user can program two configuration bits (FOSC1 and FOSC0) to select one of these four modes:

- LP Low Power Crystal
- XT Crystal/Resonator
- HS High Speed Crystal/Resonator
- RC Resistor/Capacitor

## CRYSTAL OSCILLATOR/CERAMIC RESONATORS

In XT, LP or HS modes, a crystal or ceramic resonator is connected to the OSC1/CLKIN and OSC2/CLKOUT pins to establish oscillation (Figure 1). The PIC16F87X oscillator design requires the use of a parallel cut crystal. Use of a series cut crystal may give a frequency out of the crystal manufacturers specifications.

When in XT, LP or HS modes, the device can have an external clock source to drive the OSC1/ CLKIN pin (Figure 2).

FIGURE 1: CRYSTAL/CERAMIC RESONATOR OPERATION (HS, XT OR LP OSC CONFIGURATION)

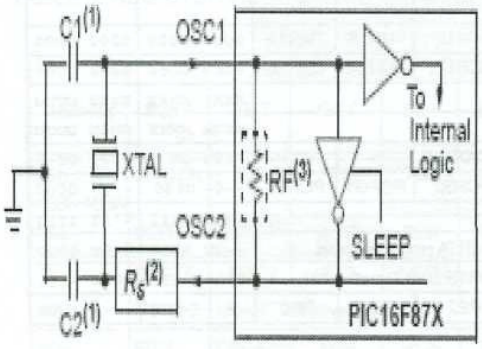
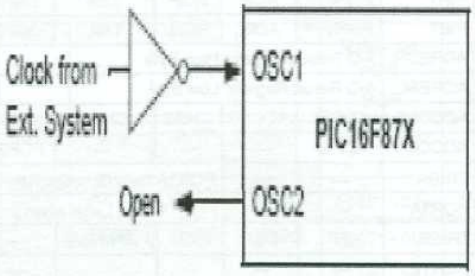



FIGURE 2: EXTERNAL CLOCK INPUT OPERATION (HS, XT OR LP OSC CONFIGURATION)



# Apéndice B

# Divergencias de ANSI C estándar con PICCLITE

# ANEXO B *Divergence from the ANSI C Standard*

PICC Lite diverges from the ANSI C standard in one area: function recursion.
Due to the PIC's hardware limitations of no stack and limited memory, function recursion is unsupported.

## Supported Data Types

The PICC Lite compiler supports basic data types of 1, 2 and 4 byte size. All multi-byte types follow *least significant byte first* format, also known as *little-endian*. Word size values thus have the least significant byte at the lower address, and double word size values have the least significant byte and least significant word at the lowest address.
Table 1.1 shows the data types and their corresponding size and arithmetic type.
TABLE 1.1 DATA TYPES

| Type | Size (in bits) | Arithmetic Type |
|---|---|---|
| bit | 1 | boolean |
| char | 8 | signed or unsigned integer[a] |
| unsigned char | 8 | unsigned integer |
| short | 16 | signed integer |
| unsigned short | 16 | unsigned integer |
| int | 16 | signed integer |
| unsigned int | 16 | unsigned integer |
| long | 32 | signed integer |
| unsigned long | 32 | unsigned integer |
| float | 24 | real |
| double | 24 or 32[b] | real |

## Structures and Unions

PICC Lite supports **struct** and **union** types of any size from one byte upwards. Structures and unions may be passed freely as function arguments and return values. Pointers to structures and unions are fully supported.

### Structure Qualifiers

PICC Lite supports the use of type qualifiers on structures. When a qualifier is applied to a structure, all of its members will attain this qualification

## Pointers

The format and use of pointers depend upon the range of processor.

### Midrange Pointers

All pointers for the Midrange are the same as for the Baseline processors with the following exceptions:

- *RAM Pointers*
  Because an 8-bit pointer can only access 256 bytes, RAM pointers can only access objects in Bank 0 and Bank 1.

- *Bank2 Pointers* and *Bank3 Pointers*
  These pointers are RAM pointers which are used to access Bank 2 and Bank 3 of RAM respectively.
  Note that at present it is not possible to have a Midrange RAM pointer which can access objects in three or more banks, or which can access objects in bank pairs other than those mentioned above.
- *Const Pointers*
  **Const** pointers for the Midrange processors are 16-bit wide. They can be used to access either ROM or RAM.
  If the upper bit of the **const** pointer is non-zero, it is a pointer into RAM in nay bank. A **const** pointer may be used to read from RAM locations, but writing to such locations is not permitted.
  If the upper bit is zero, it is a pointer able to access the entire ROM space.
  The ability of this pointer to access both ROM and RAM is very useful in string-related functions where a pointer passed to the function may point to a string in ROM or RAM.

- *Function Pointers*
  These pointers reference functions. A function is called using the address assigned to the pointer.

## Mixing C and Assembler Code

**#asm, #endasm and asm()**

PIC instructions may also be directly embedded in C code using the directives **#asm**, **#endasm** and **asm()**. The **#asm** and **#endasm** directives are used to start and end a block of assembler instructions which are to be embedded inside C code. The **asm()** directive is used to embed a single assembler instruction in the code generated by the C compiler. To continue our example from above, you could directly code a rotate left on a memory byte using either technique as the following example shows:

```
#include <stdio.h>
unsigned char var;
void main(void)
{
var = 1;
#asm
rlf _var,f
#endasm
asm("rlf _var,f");
}
```

When using inline assembler code, great care must be taken to avoid interacting with compiler generated code. If in doubt, compile your program with the PICL -S option and examine the assembler code generated by the compiler.

IMPORTANT NOTE: the **#asm** and **#endasm** construct is not syntactically part of the C program, and thus it does *not* obey normal C flow-of-control rules. For example, you cannot use a **#asm** block with an **if** statement and expect it to work correctly. If you use in-line assembler around any C constructs such as **if**, **while**, **do** etc. you should use only the **asm("")** form, which is interpreted as a C statement and will correctly interact with all C flow-of-control structures.

## Linking Programs

The compiler will automatically invoke the linker unless requested to stop after producing assembler code (PICL -S option) or object code (PICL -C option).
PICL and HTLPIC by default generate *Intel* HEX files and *Bytecraft* COD. If you use the -BIN option or specify an output file with a .bin filetype using the PICL -O option the compiler will generate a binary image instead. After linking, the compiler will automatically generate a memory usage map which shows the address used by, and the total sizes of, all the memory areas which are used by the compiled code. Note that bit objects are shown separately. For example:

```
Memory Usage Map:
Program ROM $0000 - $001A$001B (27) words
Program ROM $07EE - $07FF$0012 (18) words
$002D ( 45) words total Program ROM
Bank 0 RAM $0020 - $0022$0003 ( 3) bytes total Bank 0 RAM
Bank 1 RAM $00A0 - $00A2$0003 ( 3) bytes total Bank 1 RAM
Bank 0 Bits $0118 - $0119$0002 ( 2) bits total Bank 0 Bits
```

## PIC Assembly Language

The source language accepted by the HI-TECH Software PICC Lite Macro Assembler is described below. All opcode mnemonics and operand syntax are strictly PIC assembly language.

### Additional Mnemonics

Apart from the PIC assembly language mnemonics, the PIC assembler includes the *fcall* and *ljmp* mnemonics. These instructions implement *call* and *goto* instructions but with the added job of setting the necessary bits in PCLATH. These additional mnemonics should be used where possible as they make assembler code independent of the final position of the routines that are to be executed.

## Assembler Format Deviations

The HI-TECH PICC Lite assembler uses a slightly modified form of assembly language to that specified by Microchip. Certain PIC instructions used by Microchip assembler use the operands ",0" or ",1" to specify the destination for the result of that operation. The HI-TECH PICC Lite assembler uses the more-readable operands ",w" and ",f" to specify the destination register. The W register is selected as the destination when using the ",w" operand, and the file register is selected when using the ",f" operand or if no destination operand is specified. The case of the letter in the destination operand in not important. The Microchip numerical operands cannot be used with the HI-TECH PICC Lite assembler.

## Character Set

The character set used is standard 7 bit ASCII. Alphabetic case is significant for identifiers, but not opcodes and reserved words. Tabs are treated as equivalent to spaces.

## Constants

### Numeric Constants

The assembler performs all arithmetic as signed 32 bit. Errors will be caused if a quantity is too large to fit in a memory location. The default radix for all numbers is 10. Other radices may be specified by a trailing base specifier as given in Table 1.2.

TABLE 1.2: ASPIC NUMBER AND BASES

| Radix | Format |
|---|---|
| Binary | digits 0 and 1 followed by *B* |
| Octal | digits 0 to 7 followed by *O*, *Q*, *o* or *q* |
| Decimal | digits 0 to 9 followed by *D*, *d* or nothing |
| Hexadecimal | digits 0 to 9, A to F preceded by *0x* or followed by *H or h* |

## Pointers

The format and use of pointers depend upon the range of processor.

### Midrange Pointers

All pointers for the Midrange are the same as for the Baseline processors with the following exceptions:

- *RAM Pointers*
  Because an 8-bit pointer can only access 256 bytes, RAM pointers can only access objects in Bank 0 and Bank 1.

- *Bank2 Pointers* and *Bank3 Pointers*
  These pointers are RAM pointers which are used to access Bank 2 and Bank 3 of RAM respectively.
  Note that at present it is not possible to have a Midrange RAM pointer which can access objects in three or more banks, or which can access objects in bank pairs other than those mentioned above.
- *Const Pointers*
  **Const** pointers for the Midrange processors are 16-bit wide. They can be used to access either ROM or RAM.
  If the upper bit of the **const** pointer is non-zero, it is a pointer into RAM in nay bank. A **const** pointer may be used to read from RAM locations, but writing to such locations is not permitted.
  If the upper bit is zero, it is a pointer able to access the entire ROM space.
  The ability of this pointer to access both ROM and RAM is very useful in string-related functions where a pointer passed to the function may point to a string in ROM or RAM.

- *Function Pointers*
  These pointers reference functions. A function is called using the address assigned to the pointer.

## Character Constants

A character constant is a single character enclosed in single quotes ( ' ). Multi character constants may be specified using double quotes.

## Delimiters

All numbers and identifiers must be delimited by white space, non-alphanumeric characters or the end of a line.

## Special Characters

There are a few characters that are special in certain contexts. Within a macro body, the character & is used for token concatenation. To use the bitwise & operator within a macro body, escape it by using && instead. In a macro argument list, the angle brackets < and > are used to quote macro arguments.

## Identifiers

Identifiers are user-defined symbols representing memory locations or numbers. A symbol may contain any number of characters drawn from the alphabetics, numerics and the special characters dollar ($), question mark (?) and underscore(_). The first character of an identifier may not be numeric. The case of alpahabetics is significant, e.g. *Fred* is not the same symbol as *fred*. Some examples of identifiers are shown here:

```
An_identifier
an_identifier
an_identifier1
$$$
?$_12345
```

## Strings

A string is a sequence of characters not including carriage return or newline, enclosed within matching quotes. Either single (') or double (") quotes may be used, but the opening and closing quotes must be the same. A string used as an operand to a DB directive may be any length, but a string used as operand to an instruction must not exceed 1 or 2 characters, depending on the size of the operand required.

## Expressions

Expressions are made up of numbers, symbols, strings and operators. Operators can be unary (one operand, e.g. *not*) or binary (two operands, e.g. +). The operators allowable in expressions are listed in Table 1.3. The usual rules governing the syntax of expressions apply.

The operators listed may all be freely combined in both constant and relocatable expressions. The HI-TECH linker permits relocation of complex expressions, so the results of expressions involving relocatable identifiers may not be resolved until link time.

TABLE 1.3 OPERATORS

| Operator | Purpose |
|---|---|
| * | Multiplication |
| + | Addition |
| - | Subtraction |
| / | Division |
| = *or* eq | Equality |
| > *or* gt | Signed greater than |
| >= *or* ge | Signed greater than or equal to |
| < *or* lt | Signed less than |
| <= *or* le | Signed less than or equal to |
| <> *or* ne | Signed not equal to |
| low | Low byte of operand |
| high | High byte of operand |
| highword | High 16 bits of operand |
| mod | Modulus |
| & | Bitwise AND |
| ^ | Bitwise XOR (exclusive or) |
| \| | Bitwise OR |
| not | Bitwise complement |
| << *or* shl | Shift left |
| >> *or* shr | Shift right |
| rol | Rotate left |
| ror | Rotate right |
| seg | Segment (bank number) of address |
| float24 | 24-bit version of real operand |
| nul | Tests if macro argument is null |

# Apéndice C

# Artículos publicados

En la realización del presente trabajo de tesis se envió un articulo al XIV Congreso Internacional de Computación CIC 2005, del Instituto Politécnico Nacional, en Septiembre del 2005, el cual fue seleccionado para exponerlo en este congreso, además de ser publicado en el Research On Computing Science, que es una publicación trimestral, de circulación internacional, editada por el Centro de Investigación en Computación del IPN.

A continuación se anexa éste artículo.

# Microcontroller-Based Distributed Serial Communication and PC-Based Supervision and Control for an Automated Greenhouse

Laura E. Muñoz Hernández [1], Luis A. Paredes Salinas [2], Julio Romero González[3], America Saldaña Sánchez[4], J. Antonio Sánchez Hernández[5], Julio C. Ramos Fernández[6], Virgilio López Morales[7]

Centro de Investigación en Tecnologías de Información y Sistemas, ICBI-UAEH,
Carr. Pachuca Tulancingo Km. 4.5, C.U., C.P. 42084, Pachuca Hidalgo México

[1]lauris3112@gmail.com , [2]albert8314@hotmail.com, [3]jediknight_fisto@yahoo.com.mx, [4]ameri_kas@hotmail.com, [5]his_eternal_majesty@yahoo.com.mx, [6]jramos@utlt.edu.mx, [7]virgilio@uaeh.edu.mx

**Abstract**

In order to have a better and bigger plants production, it is necessary to have a regulated micro-environment, in a greenhouse. Inside a hydroponic greenhouse it is possible to regulate the main physical variables (temperature, humidity irrigation cycles, etc.) as well as the variables involved in the rates of plants growing (nutrients, sun radiation, water, etc.). Then, a greenhouse must be constantly supervised in order to regulate the main physical variables which are involved in the production process. For instance, temperature and humidity can be regulated by opening or drawing the ventilation system or domes or by heating or circulating the air inside the greenhouse. These tasks should be supervised all days, and it demands constantly, a supervised distributed communication system. In this paper we proposed a distributed serial communication system based on microcontrollers to monitoring and supervising the process via Internet. With this system, it can be achieved the on-line supervision and control of the greenhouse operation and conditions meteorological, as well as the main variables (temperature, humidity irrigation cycles, etc.).

## 1. Introduction

Greenhouses can provide an excellent controlled environment for plant production. The greenhouse should provide uniform lighting, heating, and water to all plants. For this reason is important to have a system for supervising all this variables in order to control them, thus it can cultivate plants in excellent conditions. The advantages that offer us the greenhouses production are enough to consider automating it. These advantages are: low-costs productions, better control of pests, better quality, save of water and more than one crop per year.

Furthermore, the fast evolution of the Personal Computer (PC) in the last two decades generated a revolution in virtual instrumentation for test and measurement. Virtual instrumentation offers several benefits to engineers and scientists who require increased productivity, accuracy, and performance (National Instruments, 2005).

A virtual instrument consists of an industry-standard computer or workstation equipped with powerful application software, cost-effective hardware such as plug-in boards, and driver software, which together perform the functions of traditional instruments (National Instruments, 2005).

Our objective in this paper is to design a minimum system using the PIC16F877 microcontroller in order to create a distributed serial communication system and a resource network to control and supervise via a website. The main goal of the minimum system called Slave is the acquisition and digital conversion of analog signals, to be sent to another minimum system called Master. The Master sends these signals to the PC that achieves three main tasks: stores the acquired data in a database, keeps a Domain Name Server (DNS)[1] finally generates the user interface. The minimum systems control the actuators of the system and can achieve the acquisition of the physical variables. These variables are digitals and analogical signals, such as speed and wind direction, solar radiation, internal and external temperature of the greenhouse, internal and external humidity, rain detector, heating actuator and shadow opening (Figure 1).

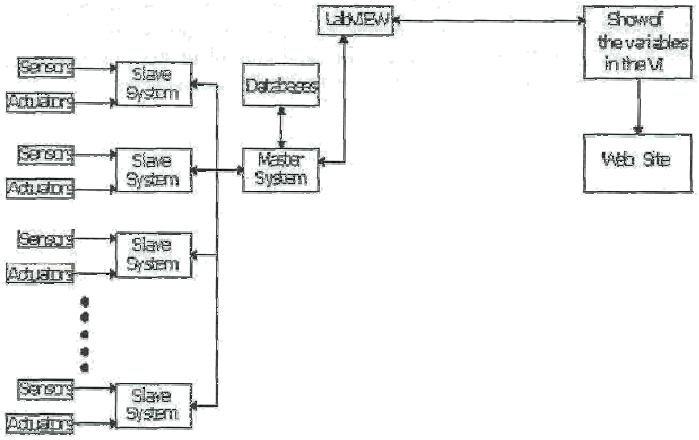



Figure 1. Block Diagram

The slave minimum systems are interconnected by a RS485 protocol to the PC through Master-Slave architecture.

We use the RS485 Serial Communication Interface, since it allows us a multipoint communication (Figure 2) and others relevant characteristics. Thus, it is necessary to realize a conversion from the RS232 Serial Communication Interface (shipped on PIC16F877) to the RS485 bus protocol.

[1] Domain Name Server is an Internet service that translates *domain names* into IP addresses. We used this server to public our own page in the Web.

| Specifications | RS-232 | RS-422 | RS-485 |
|---|---|---|---|
| Mode of Operation | Single-Ended | Differential | Differential |
| Total Number of Drivers and Receivers on One Line (One driver active at a time for RS-485 networks) | 1 Driver<br>1 Receiver | 1 Driver<br>10 Receivers | 32 Drivers<br>32 Receivers |
| Maximum Cable Length | 50 ft. (2500 pF) | 4000 ft. | 4000 ft. |
| Maximum Data Rate (40 ft.-4000 ft. for RS-422/RS-485) | 20kB/s (by spec-can be higher) | 10 Mbits/s | 10 Mbits/s |

Figure 2. Comparison of RS-232, RS-422, and RS-485 Serial Communication Interfaces

## 2. System Overview

In this section, it is shown in detail the main components and devices used in our complete system. The system consists of two main parts: the PC connected to the Internet and the Master-Slave System (Figure 3).

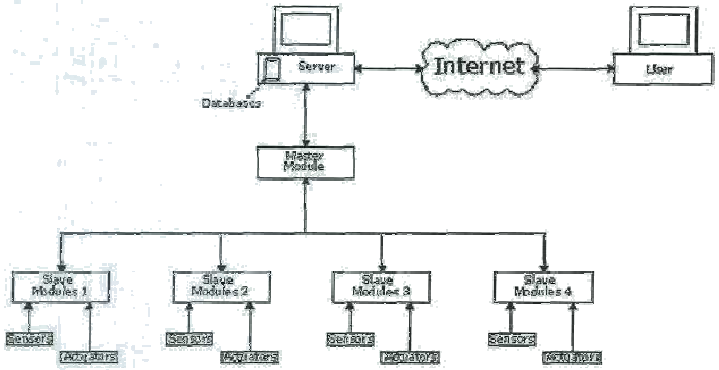



Figure 3.General Structure

The physical variables are measured by the Sensors and then sent by the Slaves Module to the Master Module and then retransmitted to the PC.

The PC stores in a database the physical variables. In addition keeps the Web server, displaying the data via Internet.

## 3. Master-Slave system

The minimum system (Figure 4) developed has eight analog inputs where the signals of the sensors are connected to be converter in digital signals using the Analog-Digital Converter Module. After the conversion the values are stored in a variable and it is ready to be sent to the master and then to the PC. The communication between devices is through the Addressable Universal Synchronous Asynchronous Receiver Transmitter (USART). In this case the USART[2] is configured as a full duplex asynchronous system and the baud rate is 9600.

[2] USART Asynchronous mode uses standard non-return-to zero (NRZ) format (one START bit, eight or nine data, bits, and one STOP bit). The USART transmits and receives the LSb first. The transmitter and receiver are functionally independent, but use the same data format and baud rate [4].

Each Slave has a nine bits direction to be identified, thus the master can send the user request to the correct slave, and then it is useful to prevent errors between information transmission and request from the master to the slaves. The communication between master and PC is within eight bits since the LabVIEW only allows eight bits transmission.

When a request arrives to the correct slave, this receives code (from the master) which indicates the information that the slave has to transmit. This information comes from the temperature sensor, humidity sensor and solar radiation sensor. Thus the user can supervise the greenhouse and control these variables by sending control signals to the actuators connected to the slaves according to the control law programmed in LabVIEW.

Moreover the analog input module, the SCI module (transmission- receive) and the digital outputs module (to the actuators), the minimum system has the necessary implementations to be used in our system. These implementations are: the keyboard, the display, one free port and the Master Synchronous Serial Module. The display it is used to locally supervise some parameters in the same module instead of going necessarily to PC. The others shipped-on modules are not been used at this moment, but they will integrated in a future development works.

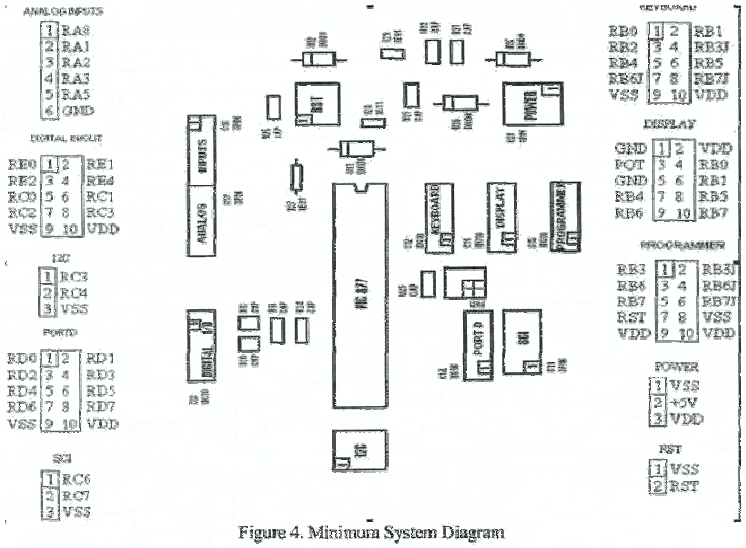



Figure 4. Minimum System Diagram

The linkage of these digital devices and components must be made by some software components. We have used some commercial packages and developed all the algorithms and VIs to have an open architecture to perform easily different control and supervision schemes.

## 4. Software Features

In order to keep some compatibility among the different devices and software components and reliability of the proposed architecture, it was chosen some commercially software as described in the following.

### 4.1 LabView

LabView is a graphical programming language that uses icons instead of lines of text to create applications; furthermore, it allows achieving different forms of communication between a PC and external digital devices. This communication can be made by a serial or a parallel port, and all of this is programmed under an easy graphic user interface. Since the master and slaves, based on the microcontroller 16F877, have two serial ports, we have chosen a LabView based on serial communication interface.

The virtual instrumentation is achieved via virtual instruments (VI). This VI, defined by LabView, seems and operates as a physical instrument, such as an oscilloscope or a multi-meter. Furthermore, it can be used a VI in another VI, it is called a subVI.

In this work, each VI is used to manipulate functions for example the VI presents data on the screen or sends it to another files or PC's. (Figure 5).

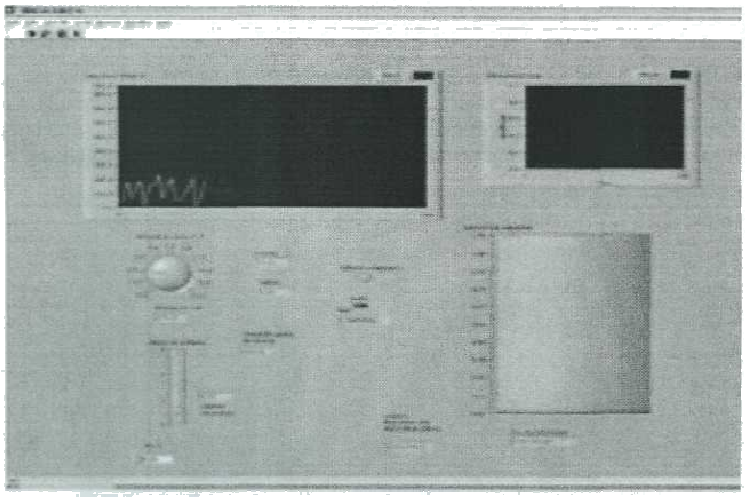


Figure 5. LabView graphic interfaces

For the database, the data came from the Master and is directly saved in an Excel file due to the LabView input output format requirement. Thus, the created file is called for another program which manages databases (MySQL).

### 4.2 MySQL and PHP

MySQL is a compact database server ideal for small applications. In addition to supporting standard SQL[3] (ANSI), it compiles on a number of platforms and has different operating systems multithreading abilities on UNIX servers[4], providing a great performance.

This software is used to import information from an Excel file to a database admitted by PHP[5] software. PHP can access most any SQL or ODBC database. It can both read and write information in the database.

### 4.3 Dreamweaver

The visual editing features in Dreamweaver[6] allow us create some pages without writing a line of code and it can view all the site elements or assets and drag them from an easy-to-use panel directly into a document. We can streamline the development workflow by creating and editing images in Macromedia Fireworks or another graphics application, then importing them directly into Dreamweaver, or by adding Macromedia Flash objects.

How it was mentioned, Dreamweaver is only a tool that helps to the design of the Web page, and can be shown in a Web navigator in any kind of platform with Internet connection.

---

[3] The SQL part of MySQL stands for "Structured Query Language" - the most common standardized language used to access databases

[4] Red Hat 9 Linux is the leading platform for open source computing.

[5] (Personal Home Page) Hypertext Preprocessor

[6] Macromedia Dreamweaver MX 2004 is a professional HTML editor for designing, coding, and developing websites, web pages, and web applications. It allows us the control of hand-coding HTML or a visual editing environment

## 5. Website

The website (Figure 6) shows pictures for the crop plantation and the greenhouse construction architectonic, the different access to read the physical variables databases, and the LabView virtual instruments to supervise the different modules.

The website involves a DNS Server and a Dynamic Host Configuration Protocol (DHCP) due to the necessity of a DNS server to have a communication with Internet, since this convert the domain name to an IP address used in the TCP/IP protocol. The DHCP allows us the generation of an IP address automatic (to the user connected in the network) in a range of IP addresses, thus it is viable to have a multipoint network instead of a point to point network.

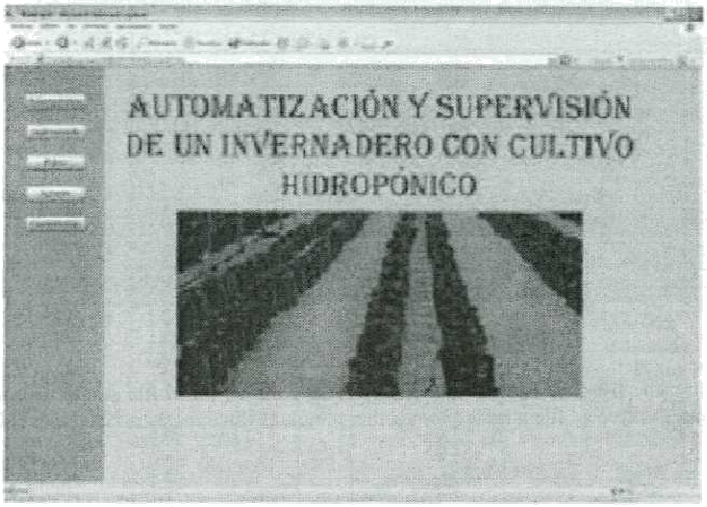



Figure 6. Web Site

## 6. Conclusions and perspectives

Based on the PIC 16F877 microcontroller, we have proposed a distributed serial communication system to supervise the external variables and control the internal variables in a greenhouse. The serial communication protocol used is RS485, since it allows us a multipoint communication and others relevant characteristics (number of receivers and cable length). Also some communication failure test has been carried out in order to check the robustness performance.

## References:

[1] LabView Basics II Course Manual, 2000, National Instrument, September 2000, Edition, Texas, USA.
[2] Using de Dreamweaver MX, 2004, Macromedia, September 2003, Ed. San Fancisco, USA.
[3] Red Hat Linux Getting Started Guide, 2003, Red Hat, Inc.
[4] LabView User Manual, 2003, National Instrument, April 2003, Edition, Texas, USA.
[5] MySQL Reference Manual, 2001.
[6] Serodio Carlos, Boaventura Cunha, J., Morais Raul, Couto Carlos, Monteiro Joao. A networked platform for agricultural management system. 2001. Computers and Electronics in Agricultural.
[7] Fuentes, J.M., Herrera, J., Arboleda, J.P., Heit, F., Casas, C., Company, J. Communication system for a distributed intelligent controller. 1999. Microprocessors and Microsystems
[8] Tipsuwan, Y., Chow Mo-Yuen. Control Methodologies in networked control system. 2003. 2003. Control Engineering Practice.
[9] Gieling, Th.H., Van Meurs, W. Th., Jansen, J. A computer network with SCADA and case tools for on-line process control in greenhouses. 1996. Pergamon.
[10] Boaventura Cunha, J., Moura Oliveira, J.P. Optimal management of greenhouses environments. 2003. EFITA 2003 Conference.